# Manual de referencia de GNU Guix

Uso del gestor de paquetes funcional GNU Guix.


## Los desarrolladores de GNU Guix

Edición c7888f5
31 May 2024

# Table of Contents

# 1 Introducción

GNU Guix[1] es una herramienta de gestión de paquetes y una distribución del sistema GNU. Guix facilita a usuarias sin privilegios la instalación, actualización o borrado de paquetes de software, la vuelta a un conjunto de paquetes previo atómicamente, la construcción de paquetes desde las fuentes, y ayuda de forma general en la creación y mantenimiento de entornos software.

Puede instalar GNU Guix sobre un sistema GNU/Linux existente, donde complementará las herramientas disponibles sin interferencias (see Chapter 2 [Instalación], page 4), o puede usarse como un sistema operativo en sí mismo, el *sistema Guix*[2]. See Section 1.2 [Distribución GNU], page 2.

## 1.1 Gestión de software con Guix

Guix provides a command-line package management interface (see Chapter 5 [Gestión de paquetes], page 35), tools to help with software development (see Chapter 7 [Desarrollo], page 79), command-line utilities for more advanced usage (see Chapter 9 [Utilidades], page 181), as well as Scheme programming interfaces (see Chapter 8 [Interfaz programática], page 101).

Su *daemon de construcción* es responsable de la construcción de paquetes en delegación de las usuarias (see Section 2.2 [Preparación del daemon], page 5) y de la descarga de binarios preconstruidos de fuentes autorizadas (see Section 5.3 [Sustituciones], page 46)

Guix incluye definiciones de paquetes para muchos paquetes GNU y no-GNU, todos los cuales respetan la libertad de computación de la usuaria (`https://www.gnu.org/philosophy/free-sw.html`). Es *extensible*: las usuarias pueden escribir sus propias definiciones de paquetes (see Section 8.2 [Definición de paquetes], page 102) y hacerlas disponibles como módulos independientes de paquetes (see Section 8.1 [Módulos de paquetes], page 101). También es *personalizable*: las usuarias pueden *derivar* definiciones de paquetes especializadas de las existentes, inclusive desde la línea de órdenes (see Section 9.1.2 [Opciones de transformación de paquetes], page 184).

En su implementación, Guix utiliza la disciplina de *gestión de paquetes funcional* en la que Nix fue pionero (see Chapter 23 [Reconocimientos], page 780). En Guix, el proceso de construcción e instalación es visto como una *función*, en el sentido matemático. Dicha función toma entradas, como los guiones de construcción, un compilador, unas bibliotecas y devuelve el paquete instalado. Como función pura, su resultado únicamente depende de sus entradas—por ejemplo, no puede hacer referencia a software o guiones que no fuesen pasados explícitamente como entrada. Una función de construcción siempre produce el mismo resultado cuando se le proporciona un conjunto de entradas dado. No puede modificar el entorno del sistema que la ejecuta de ninguna forma; por ejemplo, no puede crear, modificar o borrar archivos fuera de sus directorios de construcción e instalación. Esto

[1] "Guix" se pronuncia tal y como se escribe en castellano, "iks" en el alfabeto fonético internacional (IPA).

[2] Solíamos referirnos al sistema Guix como "Distribución de sistema Guix" o "GuixSD". Ahora consideramos que tiene más sentido agrupar todo bajo la etiqueta "Guix" ya que, después de todo, el sistema Guix está inmediatamente disponible a través de la orden `guix system`, ¡incluso cuando usa una distribución distinta por debajo!

se consigue ejecutando los procesos de construcción en entornos aislados (o *contenedores*), donde únicamente sus entradas explícitas son visibles.

El resultado de las funciones de construcción de paquetes es *almacenado en la caché* en el sistema de archivos, en un directorio especial llamado *el almacén* (see Section 8.9 [El almacén], page 157). Cada paquete se instala en un directorio propio en el almacén—por defecto, bajo `/gnu/store`. El nombre del directorio contiene el hash de todas las entradas usadas para construir el paquete; por tanto, cambiar una entrada resulta en un nombre de directorio distinto.

Esta aproximación es el cimiento de las avanzadas características de Guix: capacidad para la actualización transaccional y vuelta-atrás de paquetes, instalación en el ámbito de la usuaria y recolección de basura de paquetes (see Section 5.1 [Características], page 35).

## 1.2 Distribución GNU

Guix viene con una distribución del sistema GNU consistente en su totalidad de software libre[3]. La distribución puede instalarse independientemente (see Chapter 3 [Instalación del sistema], page 21), pero también es posible instalar Guix como un gestor de paquetes sobre un sistema GNU/Linux existente (see Chapter 2 [Instalación], page 4). Para distinguir entre las dos opciones, nos referimos a la distribución independiente como el sistema Guix.

La distribución proporciona paquetes principales de GNU como GNU libc, GCC y Binutils, así como muchas aplicaciones GNU y no-GNU. La lista completa de paquetes disponibles se puede explorar en línea (`https://www.gnu.org/software/guix/packages`) o ejecutando `guix package` (see Section 5.2 [Invocación de guix package], page 36):

```
guix package --list-available
```

Nuestro objetivo es proporcionar una distribución práctica con 100% software libre basada en Linux y otras variantes de GNU, con un enfoque en la promoción y la alta integración de componentes GNU, y un énfasis en programas y herramientas que ayuden a las usuarias a ejercitar esa libertad.

Actualmente hay paquetes disponibles para las siguientes plataformas:

`x86_64-linux`
: Arquitectura `x86_64` de Intel/AMD, con núcleo Linux-Libre.

`i686-linux`
: Arquitectura de 32-bits Intel (IA32), con núcleo Linux-Libre.

`armhf-linux`
: Arquitectura ARMv7-A con coma flotante hardware, Thumb-2 y NEON, usando la interfaz binaria de aplicaciones (ABI) EABI con coma flotante hardware, y con el núcleo Linux-Libre.

`aarch64-linux`
: procesadores ARMv8-A de 64 bits little-endian, con el núcleo Linux-Libre.

`i586-gnu` GNU/Hurd (`https://hurd.gnu.org`) en la arquitectura Intel de 32 bits (IA32).

[3] El término "libre" aquí se refiere a la libertad proporcionada a las usuarias de dicho software (`https://www.gnu.org/philosophy/free-sw.html`).

Esta configuración es experimental y se encuentra en desarrollo. La forma más fácil de probarla es configurando una instancia del servicio `hurd-vm-service-type` en su máquina GNU/Linux (see [transparent-emulation-qemu], page 546). ¡See Chapter 22 [Contribuir], page 734, para informarse sobre cómo ayudar!

`mips64el-linux (unsupported)`
: little-endian 64-bit MIPS processors, specifically the Loongson series, n32 ABI, and Linux-Libre kernel. This configuration is no longer fully supported; in particular, there is no ongoing work to ensure that this architecture still works. Should someone decide they wish to revive this architecture then the code is still available.

`powerpc-linux (unsupported)`
: big-endian 32-bit PowerPC processors, specifically the PowerPC G4 with AltiVec support, and Linux-Libre kernel. This configuration is not fully supported and there is no ongoing work to ensure this architecture works.

`powerpc64le-linux`
: little-endian 64-bit Power ISA processors, Linux-Libre kernel. This includes POWER9 systems such as the RYF Talos II mainboard (`https://www.fsf.org/news/talos-ii-mainboard-and-talos-ii-lite-mainboard-now-fsf-certified-to-resp`
This platform is available as a "technology preview": although it is supported, substitutes are not yet available from the build farm (see Section 5.3 [Sustituciones], page 46), and some packages may fail to build (see Section 22.11 [Tracking Bugs and Changes], page 765). That said, the Guix community is actively working on improving this support, and now is a great time to try it and get involved!

`riscv64-linux`
: little-endian 64-bit RISC-V processors, specifically RV64GC, and Linux-Libre kernel. This platform is available as a "technology preview": although it is supported, substitutes are not yet available from the build farm (see Section 5.3 [Sustituciones], page 46), and some packages may fail to build (see Section 22.11 [Tracking Bugs and Changes], page 765). That said, the Guix community is actively working on improving this support, and now is a great time to try it and get involved!

Con el sistema Guix, *declara* todos los aspectos de la configuración del sistema y Guix se hace cargo de instanciar la configuración de manera transaccional, reproducible y sin estado global (see Chapter 11 [Configuración del sistema], page 242). El sistema Guix usa el núcleo Linux-libre, el sistema de inicialización Shepherd (see Section "Introduction" in *The GNU Shepherd Manual*), las conocidas utilidades y herramientas de compilación GNU, así como el entorno gráfico o servicios del sistema de su elección.

Guix System is available on all the above platforms except `mips64el-linux`, `powerpc-linux`, `powerpc64le-linux` and `riscv64-linux`.

Para información sobre el transporte a otras arquitecturas o núcleos, see Chapter 21 [Transportar], page 733.

La construcción de esta distribución es un esfuerzo cooperativo, ¡y esta invitada a unirse! See Chapter 22 [Contribuir], page 734, para información sobre cómo puede ayudar.

# 2 Instalación

You can install the package management tool Guix on top of an existing GNU/Linux or GNU/Hurd system[1], referred to as a *foreign distro*. If, instead, you want to install the complete, standalone GNU system distribution, *Guix System*, see Chapter 3 [Instalación del sistema], page 21. This section is concerned only with the installation of Guix on a foreign distro.

> **Importante:** This section only applies to systems without Guix. Following it for existing Guix installations will overwrite important system files.

Cuando está instalado sobre una distribución distinta, GNU Guix complementa las herramientas disponibles sin interferencias. Sus datos radican exclusivamente en dos directorios, normalmente `/gnu/store` y `/var/guix`; otros archivos en su sistema, como `/etc`, permanecen intactos.

Una vez instalado, Guix puede ser actualizado ejecutando `guix pull` (see Section 5.7 [Invocación de guix pull], page 57.

## 2.1 Instalación binaria

This section describes how to install Guix from a self-contained tarball providing binaries for Guix and for all its dependencies. This is often quicker than installing from source, described later (see Section 22.2 [Construcción desde Git], page 735).

> **Importante:** This section only applies to systems without Guix. Following it for existing Guix installations will overwrite important system files.

Some GNU/Linux distributions, such as Debian, Ubuntu, and openSUSE provide Guix through their own package managers. The version of Guix may be older than c7888f5 but you can update it afterwards by running '`guix pull`'.

For Debian or a derivative such as Ubuntu, call:

```
sudo apt install guix
```

Likewise, on openSUSE:

```
sudo zypper install guix
```

The Guix project also provides a shell script, `guix-install.sh`, which automates the binary installation process without use of a foreign distro package manager[2]. Use of `guix-install.sh` requires Bash, GnuPG, GNU tar, wget, and Xz.

The script guides you through the following:

- Downloading and extracting the binary tarball
- Setting up the build daemon
- Making the 'guix' command available to non-root users
- Configuring substitute servers

As root, run:

```
# cd /tmp
```

[1] Hurd support is currently limited.

[2] `https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh`

```
# wget https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh
# chmod +x guix-install.sh
# ./guix-install.sh
```

**Nota:** By default, `guix-install.sh` will configure Guix to download pre-built package binaries, called *substitutes* (see Section 5.3 [Sustituciones], page 46), from the project's build farms. If you choose not to permit this, Guix will build *everything* from source, making each installation and upgrade very expensive. See Section 5.3.7 [Sobre la confianza en binarios], page 50, for a discussion of why you may want to build packages from source.

To use substitutes from `bordeaux.guix.gnu.org`, `ci.guix.gnu.org` or a mirror, you must authorize them. For example,

```
# guix archive --authorize < \
    ~root/.config/guix/current/share/guix/bordeaux.guix.gnu.org.pub
# guix archive --authorize < \
    ~root/.config/guix/current/share/guix/ci.guix.gnu.org.pub
```

When you're done installing Guix, see Section 2.4 [Configuración de la aplicación], page 17, for extra configuration you might need, and Chapter 4 [Empezando], page 32, for your first steps!

**Nota:** El archivador de la instalación binaria puede ser (re)producido y verificado simplemente ejecutando la siguiente orden en el árbol de fuentes de Guix:

```
make guix-binary.sistema.tar.xz
```

... que a su vez ejecuta:

```
guix pack -s sistema --localstatedir \
  --profile-name=current-guix guix
```

See Section 7.3 [Invocación de guix pack], page 92, para más información sobre esta útil herramienta.

Should you eventually want to uninstall Guix, run the same script with the `--uninstall` flag:

```
./guix-install.sh --uninstall
```

With `--uninstall`, the script irreversibly deletes all the Guix files, configuration, and services.

## 2.2 Preparación del daemon

Operaciones como la construcción de un paquete o la ejecución del recolector de basura son realizadas por un proceso especializado, el *daemon de construcción*, en delegación de sus clientes. Únicamente el daemon puede acceder al almacén y su base de datos asociada. Por tanto, cualquier operación que manipula el almacén se realiza a través del daemon. Por ejemplo, las herramientas de línea de órdenes como `guix package` y `guix build` se comunican con el daemon (*via* llamadas a procedimientos remotos) para indicarle qué hacer.

The following sections explain how to prepare the build daemon's environment. See Section 5.3 [Sustituciones], page 46, for how to allow the daemon to download pre-built binaries.

### 2.2.1 Configuración del entorno de construcción

En una configuración multiusuaria estándar, Guix y su daemon—el programa `guix-daemon`—son instalados por la administradora del sistema; `/gnu/store` pertenece a `root` y `guix-daemon` se ejecuta como `root`. Usuarias sin privilegios pueden usar las herramientas de Guix para construir paquetes o acceder al almacén de otro modo, y el daemon lo hará en delegación suya, asegurando que el almacén permanece en un estado consistente, y permitiendo compartir entre usuarias los paquetes construidos.

Mientras que `guix-daemon` se ejecuta como `root`, puede que no desee que los procesos de construcción de paquetes se ejecuten como `root` también, por razones de seguridad obvias. Para evitarlo, una reserva especial de *usuarias de construcción* debe ser creada para ser usada por los procesos de construcción iniciados por el daemon. Estas usuarias de construcción no necesitan tener un intérprete ni un directorio home: simplemente serán usadas cuando el daemon se deshaga de los privilegios de `root` en los procesos de construcción. Tener varias de dichas usuarias permite al daemon lanzar distintos procesos de construcción bajo UID separados, lo que garantiza que no interferirán entre ellos—una característica esencial ya que las construcciones se caracterizan como funciones puras (see Chapter 1 [Introducción], page 1).

En un sistema GNU/Linux, una reserva de usuarias de construcción puede ser creada así (usando la sintaxis de Bash y las órdenes de `shadow`):

```
# groupadd --system guixbuild
# for i in $(seq -w 1 10);
  do
    useradd -g guixbuild -G guixbuild           \
            -d /var/empty -s $(which nologin)   \
            -c "Guix build user $i" --system    \
            guixbuilder$i;
  done
```

El número de usuarias de construcción determina cuantos trabajos de construcción se pueden ejecutar en paralelo, especificado por la opción `--max-jobs` (see Section 2.3 [Invocación de guix-daemon], page 12). Para usar `guix system vm` y las órdenes relacionadas, puede necesitar añadir las usuarias de construcción al grupo `kvm` para que puedan acceder a `/dev/kvm`, usando `-G guixbuild,kvm` en vez de `-G guixbuild` (see Section 11.16 [Invocación de guix system], page 634).

The `guix-daemon` program may then be run as `root` with the following command[3]:

```
# guix-daemon --build-users-group=guixbuild
```

De este modo, el daemon inicia los procesos de construcción en un "chroot", bajo una de las usuarias `guixbuilder`. En GNU/Linux, por defecto, el entorno "chroot" contiene únicamente:

[3] If your machine uses the systemd init system, copying the *prefix*`/lib/systemd/system/guix-daemon.service` file to `/etc/systemd/system` will ensure that `guix-daemon` is automatically started. Similarly, if your machine uses the Upstart init system, copy the *prefix*`/lib/upstart/system/guix-daemon.conf` file to `/etc/init`.

- un directorio `/dev` mínimo, creado en su mayor parte independientemente del `/dev` del sistema anfitrión[4];
- el directorio `/proc`; únicamente muestra los procesos del contenedor ya que se usa un espacio de nombres de PID separado;
- `/etc/passwd` con una entrada para la usuaria actual y una entrada para la usuaria `nobody`;
- `/etc/groups` con una entrada para el grupo de la usuaria;
- `/etc/hosts` con una entrada que asocia `localhost` a `127.0.0.1`;
- un directorio `/tmp` con permisos de escritura.

The chroot does not contain a `/home` directory, and the `HOME` environment variable is set to the non-existent `/homeless-shelter`. This helps to highlight inappropriate uses of `HOME` in the build scripts of packages.

All this usually enough to ensure details of the environment do not influence build processes. In some exceptional cases where more control is needed—typically over the date, kernel, or CPU—you can resort to a virtual build machine (see [build-vm], page 548).

Puede influir en el directorio que el daemon utiliza para almacenar los árboles de construcción a través de la variable de entorno `TMPDIR`. No obstante, el árbol de construcción en el "chroot" siempre se llama `/tmp/guix-build-`*nombre*`.drv-0`, donde *nombre* es el nombre de la derivación—por ejemplo, `coreutils-8.24`. De este modo, el valor de `TMPDIR` no se escapa a los entornos de construcción, lo que evita discrepancias en caso de que los procesos de construcción capturen el nombre de su árbol de construcción.

El daemon también respeta la variable de entorno `http_proxy` y `https_proxy` para las descargas HTTP y HTTPS que realiza, ya sea para derivaciones de salida fija (see Section 8.10 [Derivaciones], page 159) o para sustituciones (see Section 5.3 [Sustituciones], page 46).

Si está instalando Guix como una usuaria sin privilegios, es posible todavía ejecutar `guix-daemon` siempre que proporcione el parámetro `--disable-chroot`. No obstante, los procesos de construcción no estarán aislados entre sí ni del resto del sistema. Por tanto, los procesos de construcción pueden interferir entre ellos y pueden acceder a programas, bibliotecas y otros archivos disponibles en el sistema—haciendo mucho más difícil verlos como funciones *puras*.

### 2.2.2 Uso de la facilidad de delegación de trabajo

Cuando así se desee, el daemon de construcción puede *delegar* construcciones de derivación a otras máquinas ejecutando Guix, usando el *procedimiento de extensión de construcción* `offload`[5]. Cuando dicha característica es activada se lee una lista de máquinas de construcción especificadas por la usuaria desde `/etc/guix/machines.scm`; cada vez que se solicita una construcción, por ejemplo via `guix build`, el daemon intenta su delegación a una de las máquinas que satisfaga las condiciones de la derivación, en particular su tipo de sistema—por ejemplo, `x86_64-linux`. Una única máquina puede usarse para múltiples

[4] "En su mayor parte" porque, mientras el conjunto de archivos que aparecen en `/dev` es fijo, la mayor parte de estos archivos solo pueden ser creados si el sistema anfitrión los tiene.

[5] Esta característica está únicamente disponible cuando Guile-SSH (`https://github.com/artyom-potsov/guile-ssh`) está presente.

tipos de sistema, ya sea porque los implemente su arquitectura de manera nativa, a través de emulación (see [transparent-emulation-qemu], page 546), o ambas. Los prerrequisitos restantes para la construcción se copian a través de SSH a la máquina seleccionada, la cual procede con la construcción; con un resultado satisfactorio la o las salidas de la construcción son copiadas de vuelta a la máquina inicial. La facilidad de descarga de trabajo incorpora una planificación básica que intenta seleccionar la mejor máquina, la cual es seleccionada entre las máquinas disponibles en base a criterios como los siguientes:

1. La disponibilidad de un puesto de construcción. Una máquina de construcción tiene el número de puestos de construcción (conexiones) que indique el valor de `parallel-builds` en su objeto `build-machine`.
2. Su velocidad relativa, a través del campo `speed` de su objeto `build-machine`.
3. Su carga de trabajo. El valor normalizado de carga debe ser menor aun valor límite, configurable a través del campo `overload-threshold` de su objeto `build-machine`.
4. El espacio disponible en el disco. Debe haber más de 100 MiB disponibles.

El archivo `/etc/guix/machines.scm` normalmente tiene un contenido de este estilo:

```
(list (build-machine
        (name "ochentayseis.example.org")
        (systems (list "x86_64-linux" "i686-linux"))
        (host-key "ssh-ed25519 AAAAC3Nza...")
        (user "rober")
        (speed 2.))     ;¡increíblemente rápida!

      (build-machine
        (name "armeight.example.org")
        (systems (list "aarch64-linux"))
        (host-key "ssh-rsa AAAAB3Nza...")
        (user "alice")

        ;; Remember 'guix offload' is spawned by
        ;; 'guix-daemon' as root.
        (private-key "/root/.ssh/identity-for-guix")))
```

En el ejemplo anterior se especifica una lista de dos máquinas de construcción, una para las arquitecturas `x86_64` y `i686`, y otra para la arquitectura `aarch64`.

De hecho, este archivo es—¡sin sorpresa ninguna!—un archivo Scheme que se evalúa cuando el procedimiento de extensión `offload` se inicia. El valor que devuelve debe ser una lista de objetos `build-machine`. Mientras que este ejemplo muestra una lista fija de máquinas de construcción, una puede imaginarse, digamos, el uso de DNS-SD para devolver una lista de máquinas de construcción potenciales descubierta en la red local (see Section "Introduction" in *Using Avahi in Guile Scheme Programs*). El tipo de datos `build-machine` se detalla a continuación.

`build-machine` [Tipo de datos]

Este tipo de datos representa las máquinas de construcción a las cuales el daemon puede delegar construcciones. Los campos importantes son:

`name` El nombre de red de la máquina remota.

`systems` Los tipos de sistema implementados por la máquina remota—por ejemplo, `(list "x86_64-linux" "i686-linux")`.

`user` The user account on the remote machine to use when connecting over SSH. Note that the SSH key pair must *not* be passphrase-protected, to allow non-interactive logins.

`host-key` Este campo debe contener la *clave pública de la máquina* de SSH en formato OpenSSH. Es usado para autentificar la máquina cuando nos conectamos a ella. Es una cadena larga más o menos así:

```
ssh-ed25519 AAAAC3NzaC...mde+UhL recordatorio@example.org
```

Si la máquina está ejecutando el daemon OpenSSH, `sshd`, la clave pública de la máquina puede encontrarse en un archivo como `/etc/ssh/ssh_host_ed25519_key.pub`.

Si la máquina está ejecutando el daemon SSH GNU lsh, `lshd`, la clave de la máquina está en `/etc/lsh/host-key.pub` o un archivo similar. Puede convertirse a formato OpenSSH usando `lsh-export-key` (see Section "Converting keys" in *LSH Manual*):

```
$ lsh-export-key --openssh < /etc/lsh/host-key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAAEOp8FoQAAAQEAs1eB46LV...
```

Ciertos número de campos opcionales pueden ser especificados:

`port` (predeterminado: `22`)
Número de puerto del servidor SSH en la máquina.

`private-key` (predeterminada: `~root/.ssh/id_rsa`)
El archivo de clave privada SSH usado para conectarse a la máquina, en formato OpenSSH. Esta clave no debe estar protegida con una contraseña.

Tenga en cuenta que el valor predeterminado es la clave privada *de la cuenta de root*. Asegúrese de que existe si usa el valor predeterminado.

`compression` (predeterminado: `"zlib@openssh.com,zlib"`)
`compression-level` (predeterminado: `3`)
Los métodos de compresión y nivel de compresión a nivel SSH solicitados.

Tenga en cuenta que la delegación de carga depende de la compresión SSH para reducir el ancho de banda usado cuando se transfieren archivos hacia y desde máquinas de construcción.

`daemon-socket` (predeterminado: `"/var/guix/daemon-socket/socket"`)
Nombre de archivo del socket de dominio Unix en el que `guix-daemon` escucha en esa máquina.

`overload-threshold` (default: `0.8`)
El límite superior de carga, el cual se usa en la planificación de delegación de tareas para descartar potenciales máquinas si superan dicho límite. Dicho valor más o menos representa el uso del procesador de la máquina, con un rango de 0.0 (0%) a 1.0 (100%). También se puede desactivar si se proporciona el valor `#f` en `overload-threshold`.

`parallel-builds` (predeterminadas: `1`)
: El número de construcciones que pueden ejecutarse en paralelo en la máquina.

`speed` (predeterminado: `1.0`)
: Un "factor de velocidad relativa". El planificador de delegaciones tenderá a preferir máquinas con un factor de velocidad mayor.

`features` (predeterminadas: `'()`)
: Una lista de cadenas denotando las características específicas permitidas por la máquina. Un ejemplo es `"kvm"` para máquinas que tienen los módulos KVM de Linux y las correspondientes características hardware. Las derivaciones pueden solicitar las características por nombre, y entonces se planificarán en las máquinas adecuadas.

**Nota:** On Guix System, instead of managing `/etc/guix/machines.scm` independently, you can choose to specify build machines directly in the `operating-system` declaration, in the `build-machines` field of `guix-configuration`. See [guix-configuration-build-machines], page 287.

El ejecutable `guix` debe estar en la ruta de búsqueda de las máquinas de construcción. Puede comprobar si es el caso ejecutando:

```
ssh build-machine guix repl --version
```

Hay una última cosa por hacer una vez `machines.scm` está en su lugar. Como se ha explicado anteriormente, cuando se delega, los archivos se transfieren en ambas direcciones entre los almacenes de las máquinas. Para que esto funcione, primero debe generar un par de claves en cada máquina para permitir al daemon exportar los archivos firmados de archivos en el almacén (see Section 5.11 [Invocación de guix archive], page 66):

```
# guix archive --generate-key
```

**Nota:** This key pair is not related to the SSH key pair that was previously mentioned in the description of the `build-machine` data type.

Cada máquina de construcción debe autorizar a la clave de la máquina maestra para que acepte elementos del almacén que reciba de la maestra:

```
# guix archive --authorize < clave-publica-maestra.txt
```

La máquina maestra debe autorizar la clave de cada máquina de construcción de la misma manera.

Todo este lío con claves está ahí para expresar las mutuas relaciones de confianza entre pares de la máquina maestra y las máquinas de construcción. Concretamente, cuando la maestra recibe archivos de una máquina de construcción (y *vice versa*), su daemon de construcción puede asegurarse de que son genuinos, no han sido modificados, y que están firmados por una clave autorizada.

Para comprobar si su configuración es operacional, ejecute esta orden en el nodo maestro:

```
# guix offload test
```

This will attempt to connect to each of the build machines specified in `/etc/guix/machines.scm`, make sure Guix is available on each machine, attempt to export to the machine and import from it, and report any error in the process.

Si quiere probar un archivo de máquinas diferente, simplemente lo debe especificar en la línea de órdenes:

```
# guix offload test otras-maquinas.scm
```

Por último, puede probar un subconjunto de máquinas cuyos nombres coincidan con una expresión regular así:

```
# guix offload test maquinas.scm '\.gnu\.org$'
```

Para mostrar la carga actual de todas las máquinas de construcción, ejecute esta orden en el nodo principal:

```
# guix offload status
```

### 2.2.3 Soporte de SELinux

Guix incluye un archivo de política SELinux en `etc/guix-daemon.cil` que puede ser instalado en un sistema donde SELinux está activado, para etiquetar los archivos Guix y especificar el comportamiento esperado del daemon. Ya que el sistema Guix no proporciona una política base de SELinux, la política del daemon no puede usarse en el sistema Guix.

#### 2.2.3.1 Instalación de la política de SELinux

> **Nota:** The `guix-install.sh` binary installation script offers to perform the steps below for you (see Section 2.1 [Instalación binaria], page 4).

Para instalar la política ejecute esta orden como root:

```
semodule -i /var/guix/profiles/per-user/root/current-guix/share/selinux/guix-daemon.ci
```

Then, as root, relabel the file system, possibly after making it writable:

```
mount -o remount,rw /gnu/store
restorecon -R /gnu /var/guix
```

At this point you can start or restart `guix-daemon`; on a distribution that uses systemd as its service manager, you can do that with:

```
systemctl restart guix-daemon
```

Una vez la política está instalada, el sistema de archivos ha sido re-etiquetado, y el daemon ha sido reiniciado, debería ejecutarse en el contexto `guix_daemon_t`. Puede confirmarlo con la siguiente orden:

```
ps -Zax | grep guix-daemon
```

Monitorice los archivos de log de SELinux mientras ejecuta una orden como `guix build hello` para convencerse que SELinux permite todas las operaciones necesarias.

#### 2.2.3.2 Limitaciones

Esta política no es perfecta. Aquí está una lista de limitaciones o comportamientos extraños que deben ser considerados al desplegar la política SELinux provista para el daemon Guix.

1. `guix_daemon_socket_t` isn't actually used. None of the socket operations involve contexts that have anything to do with `guix_daemon_socket_t`. It doesn't hurt to have this unused label, but it would be preferable to define socket rules for only this label.

2. `guix gc` cannot access arbitrary links to profiles. By design, the file label of the destination of a symlink is independent of the file label of the link itself. Although all profiles under `$localstatedir` are labelled, the links to these profiles inherit the label of the directory they are in. For links in the user's home directory this will be `user_home_t`. But for links from the root user's home directory, or `/tmp`, or the HTTP server's working directory, etc, this won't work. `guix gc` would be prevented from reading and following these links.
3. La característica del daemon de esperar conexiones TCP puede que no funcione más. Esto puede requerir reglas adicionales, ya que SELinux trata los sockets de red de forma diferente a los archivos.
4. Actualmente todos los archivos con un nombre coincidente con la expresión regular `/gnu/store.+-(gux-.+|profile)/bin/guix-daemon` tienen asignada la etiqueta `guix_daemon_exec_t`; esto significa que *cualquier* archivo con ese nombre en cualquier perfil tendrá permitida la ejecución en el dominio `guix_daemon_t`. Esto no es ideal. Una atacante podría construir un paquete que proporcione este ejecutable y convencer a la usuaria para instalarlo y ejecutarlo, lo que lo eleva al dominio `guix_daemon_t`. Llegadas a este punto, SELinux no puede prevenir que acceda a los archivos permitidos para los procesos en dicho dominio.

   You will need to relabel the store directory after all upgrades to `guix-daemon`, such as after running `guix pull`. Assuming the store is in `/gnu`, you can do this with `restorecon -vR /gnu`, or by other means provided by your operating system.

   Podríamos generar una política mucho más restrictiva en tiempo de instalación, de modo que solo el nombre *exacto* del archivo del ejecutable de `guix-daemon` actualmente instalado sea marcado como `guix_daemon_exec_t`, en vez de usar una expresión regular amplia. La desventaja es que root tendría que instalar o actualizar la política en tiempo de instalación cada vez que se actualizase el paquete de Guix que proporcione el ejecutable de `guix-daemon` realmente en ejecución.

## 2.3 Invocación de `guix-daemon`

El programa `guix-daemon` implementa toda la funcionalidad para acceder al almacén. Esto incluye iniciar procesos de construcción, ejecutar el recolector de basura, comprobar la disponibilidad de un resultado de construcción, etc. Normalmente se ejecuta como `root` así:

```
# guix-daemon --build-users-group=guixbuild
```

This daemon can also be started following the systemd "socket activation" protocol (see Section "Service De- and Constructors" in *The GNU Shepherd Manual*).

Para detalles obre como configurarlo, see Section 2.2 [Preparación del daemon], page 5.

Por defecto, `guix-daemon` inicia los procesos de construcción bajo distintos UIDs, tomados del grupo de construcción especificado con `--build-users-group`. Además, cada proceso de construcción se ejecuta en un entorno "chroot" que únicamente contiene el subconjunto del almacén del que depende el proceso de construcción, como especifica su derivación (see Chapter 8 [Interfaz programática], page 101), más un conjunto específico de directorios del sistema. Por defecto, estos directorios contienen `/dev` y `/dev/pts`. Es más, sobre GNU/Linux, el entorno de construcción es un *contenedor*: además de tener su propio árbol

del sistema de archivos, tiene un espacio de nombres de montado separado, su propio espacio de nombres de PID, de red, etc. Esto ayuda a obtener construcciones reproducibles (see Section 5.1 [Características], page 35).

Cuando el daemon realiza una construcción en delegación de la usuaria, crea un directorio de construcción bajo `/tmp` o bajo el directorio especificado por su variable de entorno `TMPDIR`. Este directorio se comparte con el contenedor durante toda la construcción, aunque dentro del contenedor el árbol de construcción siempre se llama `/tmp/guix-build-`*`nombre`*`.drv-0`.

The build directory is automatically deleted upon completion, unless the build failed and the client specified `--keep-failed` (see Section 9.1.1 [Opciones comunes de construcción], page 181).

El daemon espera conexiones y lanza un subproceso por sesión iniciada por cada cliente (una de las sub-órdenes de `guix`). La orden `guix processes` le permite tener una visión general de la actividad de su sistema mostrando clientes y sesiones activas. See Section 9.16 [Invocación de guix processes], page 237, para más información.

Se aceptan las siguientes opciones de línea de ordenes:

`--build-users-group=`*`grupo`*

Toma las usuarias de *grupo* para ejecutar los procesos de construcción (see Section 2.2 [Preparación del daemon], page 5).

`--no-substitutes`

No usa sustituciones para la construcción de productos. Esto es, siempre realiza las construcciones localmente en vez de permitir la descarga de binarios preconstruidos (see Section 5.3 [Sustituciones], page 46).

Cuando el daemon se está ejecutando con la opción `--no-substitutes`, los clientes aún pueden activar explícitamente las sustituciones a través de la llamada de procedimiento remoto `set-build-options` (see Section 8.9 [El almacén], page 157).

`--substitute-urls=`*`urls`*

Consider *urls* the default whitespace-separated list of substitute source URLs. When this option is omitted, '`https://bordeaux.guix.gnu.org https://ci.guix.gnu.org`' is used.

Esto significa que las sustituciones puede ser descargadas de *urls*, mientras estén firmadas por una firma de confianza (see Section 5.3 [Sustituciones], page 46).

See Section 5.3.3 [Obtención de sustiticiones desde otros servidores], page 48, para obtener más información sobre cómo configurar el daemon para obtener sustituciones de otros servidores.

`--no-offload`

No usa la delegación de construcciones en otras máquinas (see Section 2.2.2 [Configuración de delegación del daemon], page 7). Es decir, siempre realiza las construcciones de manera local en vez de delegar construcciones a máquinas remotas.

`--cache-failures`

Almacena en la caché los fallos de construcción. Por defecto, únicamente las construcciones satisfactorias son almacenadas en la caché.

Cuando se usa esta opción, `guix gc --list-failures` puede usarse para consultar el conjunto de elementos del almacén marcados como fallidos; `guix gc --clear-failures` borra los elementos del almacén del conjunto de fallos existentes en la caché. See Section 5.6 [Invocación de guix gc], page 53.

`--cores=`*n*

`-c` *n* Usa *n* núcleos de la CPU para construir cada derivación; `0` significa tantos como haya disponibles.

El valor predeterminado es `0`, pero puede ser sobreescrito por los clientes, como la opción `--cores` de `guix build` (see Section 9.1 [Invocación de guix build], page 181).

El efecto es definir la variable de entorno `NIX_BUILD_CORES` en el proceso de construcción, el cual puede usarla para explotar el paralelismo interno—por ejemplo, ejecutando `make -j$NIX_BUILD_CORES`.

`--max-jobs=`*n*

`-M` *n* Permite como máximo *n* trabajos de construcción en paralelo. El valor predeterminado es `1`. Fijarlo a `0` significa que ninguna construcción se realizará localmente; en vez de eso, el daemon delegará las construcciones (see Section 2.2.2 [Configuración de delegación del daemon], page 7), o simplemente fallará.

`--max-silent-time=`*segundos*

Cuando la construcción o sustitución permanece en silencio más de *segundos*, la finaliza e informa de un fallo de construcción.

The default value is `3600` (one hour).

El valor especificado aquí puede ser sobreescrito por clientes (see Section 9.1.1 [Opciones comunes de construcción], page 181).

`--timeout=`*segundos*

Del mismo modo, cuando el proceso de construcción o sustitución dura más de *segundos*, lo termina e informa un fallo de construcción.

The default value is 24 hours.

El valor especificado aquí puede ser sobreescrito por los clientes (see Section 9.1.1 [Opciones comunes de construcción], page 181).

`--rounds=`*N*

Construye cada derivación *n* veces seguidas, y lanza un error si los resultados de las construcciones consecutivas no son idénticos bit-a-bit. Fíjese que esta configuración puede ser sobreescrita por clientes como `guix build` (see Section 9.1 [Invocación de guix build], page 181).

Cuando se usa conjuntamente con `--keep-failed`, la salida que difiere se mantiene en el almacén, bajo `/gnu/store/...-check`. Esto hace fácil buscar diferencias entre los dos resultados.

`--debug` Produce salida de depuración.

Esto es útil para depurar problemas en el arranque del daemon, pero su comportamiento puede cambiarse en cada cliente, por ejemplo con la opción `--verbosity` de `guix build` (see Section 9.1 [Invocación de guix build], page 181).

`--chroot-directory=`*dir*
: Añade *dir* al chroot de construcción.

  Hacer esto puede cambiar el resultado del proceso de construcción—por ejemplo si usa dependencias opcionales, que se encuentren en *dir*, cuando están disponibles, y no de otra forma. Por esa razón, no se recomienda hacerlo. En vez de eso, asegúrese que cada derivación declara todas las entradas que necesita.

`--disable-chroot`
: Desactiva la construcción en un entorno chroot.

  No se recomienda el uso de esta opción ya que, de nuevo, podría permitir a los procesos de construcción ganar acceso a dependencias no declaradas. Es necesario, no obstante, cuando `guix-daemon` se ejecuta bajo una cuenta de usuaria sin privilegios.

`--log-compression=`*tipo*
: Comprime los logs de construcción de acuerdo a *tipo*, que puede ser `gzip`, `bzip2` o `none`.

  Unless `--lose-logs` is used, all the build logs are kept in the *localstatedir*. To save space, the daemon automatically compresses them with gzip by default.

`--discover[=yes|no]`
: Whether to discover substitute servers on the local network using mDNS and DNS-SD.

  This feature is still experimental. However, here are a few considerations.

  1. It might be faster/less expensive than fetching from remote servers;
  2. There are no security risks, only genuine substitutes will be used (see Section 5.3.4 [Verificación de sustituciones], page 49);
  3. An attacker advertising `guix publish` on your LAN cannot serve you malicious binaries, but they can learn what software you're installing;
  4. Servers may serve substitute over HTTP, unencrypted, so anyone on the LAN can see what software you're installing.

  It is also possible to enable or disable substitute server discovery at run-time by running:

  ```
  herd discover guix-daemon on
  herd discover guix-daemon off
  ```

`--disable-deduplication`
: Desactiva la "deduplicación" automática en el almacén.

  Por defecto, los archivos se añaden al almacén "deduplicados" automáticamente: si un nuevo archivo añadido es idéntico a otro que ya se encuentra en el almacén, el daemon introduce el nuevo archivo como un enlace duro al otro archivo. Esto puede reducir notablemente el uso del disco, a expensas de una carga de entrada/salida ligeramente incrementada al finalizar un proceso de construcción. Esta opción desactiva dicha optimización.

`--gc-keep-outputs[=yes|no]`
: Determina si el recolector de basura (GC) debe mantener salidas de las derivaciones vivas.

  Cuando se usa `yes`, el recolector de basura mantendrá las salidas de cualquier derivación viva disponible en el almacén—los archivos `.drv`. El valor predeterminado es `no`, lo que significa que las salidas de las derivaciones se mantienen únicamente si son alcanzables desde alguna raíz del recolector de basura. See Section 5.6 [Invocación de guix gc], page 53, para más información sobre las raíces del recolector de basura.

`--gc-keep-derivations[=yes|no]`
: Determina si el recolector de basura (GC) debe mantener derivaciones correspondientes a salidas vivas.

  Cuando se usa `yes`, como es el caso predeterminado, el recolector de basura mantiene derivaciones—es decir, archivos `.drv`—mientras al menos una de sus salidas está viva. Esto permite a las usuarias seguir la pista de los orígenes de los elementos en el almacén. El uso de `no` aquí ahorra un poco de espacio en disco.

  De este modo, usar `--gc-keep-derivations` con valor `yes` provoca que la vitalidad fluya de salidas a derivaciones, y usar `--gc-keep-outputs` con valor `yes` provoca que la vitalidad fluya de derivaciones a salidas. Cuando ambas tienen valor `yes`, el efecto es mantener todos los prerrequisitos de construcción (las fuentes, el compilador, las bibliotecas y otras herramientas de tiempo de construcción) de los objetos vivos del almacén, independientemente de que esos prerrequisitos sean alcanzables desde una raíz del recolector de basura. Esto es conveniente para desarrolladoras ya que evita reconstrucciones o descargas.

`--impersonate-linux-2.6`
: En sistemas basados en Linux, suplanta a Linux 2.6. Esto significa que la llamada del sistema `uname` del núcleo indicará 2.6 como el número de versión de la publicación.

  Esto puede ser útil para construir programas que (habitualmente de forma incorrecta) dependen en el número de versión del núcleo.

`--lose-logs`
: No guarda logs de construcción. De manera predeterminada se almacenan en el directorio *`localstatedir`*`/guix/log`.

`--system=`*`sistema`*
: Asume *sistema* como el tipo actual de sistema. Por defecto es el par de arquitectura/núcleo encontrado durante la configuración, como `x86_64-linux`.

`--listen=`*`destino`*
: Espera conexiones en *destino*. *destino* se interpreta como el nombre del archivo del socket de dominio Unix si comienza on `/` (barra a la derecha). En otro caso, *destino* se interpreta como un nombre de máquina o un nombre de máquina y puerto a escuchar. Aquí van unos pocos ejemplos:

`--listen=/gnu/var/daemon`
: Espera conexiones en el socket de dominio Unix `/gnu/var/daemon`, se crea si es necesario.

`--listen=localhost`
: Espera conexiones TCP en la interfaz de red correspondiente a `localhost`, en el puerto 44146.

`--listen=128.0.0.42:1234`
: Espera conexiones TCP en la interfaz de red correspondiente a `128.0.0.42`, en el puerto 1234.

Esta opción puede repetirse múltiples veces, en cuyo caso `guix-daemon` acepta conexiones en todos los destinos especificados. Las usuarias pueden indicar a los clientes a qué destino conectarse proporcionando el valor deseado a la variable de entorno `GUIX_DAEMON_SOCKET` (see Section 8.9 [El almacén], page 157).

> **Nota:** El protocolo del daemon `no está autentificado ni cifrado`. El uso de `--listen=`*`dirección`* es aceptable en redes locales, como clusters, donde únicamente los nodos de confianza pueden conectarse al daemon de construcción. En otros casos donde el acceso remoto al daemon es necesario, recomendamos usar sockets de dominio Unix junto a SSH.

Cuando se omite `--listen`, `guix-daemon` escucha conexiones en el socket de dominio Unix que se encuentra en *`localstatedir`*`/guix/daemon-socket/socket`.

## 2.4 Configuración de la aplicación

Cuando se usa Guix sobre una distribución GNU/Linux distinta al sistema Guix—una *distribución distinta*—unos pocos pasos adicionales son necesarios para tener todo preparado. Aquí están algunos de ellos.

### 2.4.1 Localizaciones

Los paquetes instalados a través de Guix no usarán los datos de localización del sistema anfitrión. En vez de eso, debe instalar primero uno de los paquetes de localización disponibles con Guix y después definir la variable de entorno `GUIX_LOCPATH`:

```
$ guix install glibc-locales
$ export GUIX_LOCPATH=$HOME/.guix-profile/lib/locale
```

Note that the `glibc-locales` package contains data for all the locales supported by the GNU libc and weighs in at around 930 MiB[6]. If you only need a few locales, you can define your custom locales package via the `make-glibc-utf8-locales` procedure from the `(gnu packages base)` module. The following example defines a package containing the various Canadian UTF-8 locales known to the GNU libc, that weighs around 14 MiB:

```
(use-modules (gnu packages base))

(define my-glibc-locales
```

[6] The size of the `glibc-locales` package is reduced down to about 213 MiB with store deduplication and further down to about 67 MiB when using a zstd-compressed Btrfs file system.

```
(make-glibc-utf8-locales
 glibc
 #:locales (list "en_CA" "fr_CA" "ik_CA" "iu_CA" "shs_CA")
 #:name "glibc-canadian-utf8-locales"))
```

La variable `GUIX_LOCPATH` juega un rol similar a `LOCPATH` (see Section "Locale Names" in *The GNU C Library Reference Manual*). No obstante, hay dos diferencias importantes:

1. `GUIX_LOCPATH` es respetada únicamente por la libc dentro de Guix, y no por la libc que proporcionan las distribuciones distintas. Por tanto, usar `GUIX_LOCPATH` le permite asegurarse de que los programas de la distribución distinta no cargarán datos de localización incompatibles.
2. libc añade un sufijo a cada entrada de `GUIX_LOCPATH` con `/X.Y`, donde `X.Y` es la versión de libc—por ejemplo, `2.22`. Esto significa que, en caso que su perfil Guix contenga una mezcla de programas enlazados contra diferentes versiones de libc, cada versión de libc únicamente intentará cargar datos de localización en el formato correcto.

Esto es importante porque el formato de datos de localización usado por diferentes versiones de libc puede ser incompatible.

### 2.4.2 Selector de servicios de nombres

Cuando se usa Guix en una distribución distinta, *recomendamos encarecidamente* que el sistema ejecute el *daemon de caché del servicio de nombres* de la biblioteca de C de GNU, `ncsd`, que debe escuchar en el socket `/var/run/nscd/socket`. En caso de no hacerlo, las aplicaciones instaladas con Guix pueden fallar al buscar nombres de máquinas o cuentas de usuaria, o incluso pueden terminar abruptamente. Los siguientes párrafos explican por qué.

La biblioteca de C de GNU implementa un *selector de servicios de nombres* (NSS), que es un mecanismo extensible para "búsquedas de nombres" en general: resolución de nombres de máquinas, cuentas de usuaria y más (see Section "Name Service Switch" in *The GNU C Library Reference Manual*).

Al ser extensible, NSS permite el uso de *módulos*, los cuales proporcionan nuevas implementaciones de búsqueda de nombres: por ejemplo, el módulo `nss-mdns` permite la resolución de nombres de máquina `.local`, el módulo `nis` permite la búsqueda de cuentas de usuaria usando el servicio de información de red (NIS), etc. Estos "servicios de búsqueda" extra se configuran para todo el sistema en `/etc/nsswitch.conf`, y todos los programas en ejecución respetan esta configuración (see Section "NSS Configuration File" in *The GNU C Reference Manual*).

Cuando se realiza una búsqueda de nombres—por ejemplo, llamando a la función `getaddrinfo` en C—las aplicaciones primero intentarán conectar con nscd; en caso satisfactorio, nscd realiza la búsqueda de nombres en delegación suya. Si nscd no está ejecutándose, entonces realizan la búsqueda por ellas mismas, cargando los servicios de búsqueda de nombres en su propio espacio de direcciones y ejecutándola. Estos servicios de búsqueda de nombres—los archivos `libnss_*.so`—son abiertos con `dlopen`, pero pueden venir de la biblioteca de C del sistema, en vez de la biblioteca de C contra la que la aplicación está enlazada (la biblioteca de C que viene en Guix).

Y aquí es donde está el problema: si su aplicación está enlazada contra la biblioteca de C de Guix (digamos, glibc 2.24) e intenta cargar módulos de otra biblioteca de C (digamos,

`libnss_mdns.so` para glibc 2.22), probablemente terminará abruptamente o sus búsquedas de nombres fallarán inesperadamente.

Ejecutar `nscd` en el sistema, entre otras ventajas, elimina este problema de incompatibilidad binaria porque esos archivos `libnss_*.so` se cargan en el proceso `nscd`, no en la aplicación misma.

### 2.4.3 Tipografías X11

The majority of graphical applications use Fontconfig to locate and load fonts and perform X11-client-side rendering. The `fontconfig` package in Guix looks for fonts in `$HOME/.guix-profile` by default. Thus, to allow graphical applications installed with Guix to display fonts, you have to install fonts with Guix as well. Essential font packages include `font-ghostscript`, `font-dejavu`, and `font-gnu-freefont`.

Una vez que haya instalado o borrado tipografías, o cuando se de cuenta de que una aplicación no encuentra las tipografías, puede que necesite instalar Fontconfig y forzar una actualización de su caché de tipografías ejecutando:

```
guix install fontconfig
fc-cache -rv
```

Para mostrar texto escrito en lenguas chinas, Japonés o Coreano en aplicaciones gráficas, considere instalar `font-adobe-source-han-sans` o `font-wqy-zenhei`. La anterior tiene múltiples salidas, una por familia de lengua (see Section 5.4 [Paquetes con múltiples salidas], page 51). Por ejemplo, la siguiente orden instala tipografías para lenguas chinas:

```
guix install font-adobe-source-han-sans:cn
```

Programas más antiguos como `xterm` no usan Fontconfig sino que dependen en el lado del servidor para realizar el renderizado de tipografías. Dichos programas requieren especificar un nombre completo de tipografía usando XLFD (Descripción lógica de tipografías X), como esta:

```
-*-dejavu sans-medium-r-normal-*-*-100-*-*-*-*-*-1
```

Para ser capaz de usar estos nombres completos para las tipografías TrueType instaladas en su perfil Guix, necesita extender la ruta de fuentes del servidor X:

```
xset +fp $(dirname $(readlink -f ~/.guix-profile/share/fonts/truetype/fonts.dir))
```

Después de eso, puede ejecutar `xlsfonts` (del paquete `xlsfonts`) para asegurarse que sus tipografías TrueType se enumeran aquí.

### 2.4.4 Certificados X.509

El paquete `nss-certs` proporciona certificados X.509, que permiten a los programas verificar los servidores accedidos por HTTPS.

Cuando se usa Guix en una distribución distinta, puede instalar este paquete y definir las variables de entorno relevantes de modo que los paquetes sepan dónde buscar los certificados. See Section 11.12 [Certificados X.509], page 621, para información detallada.

### 2.4.5 Paquetes Emacs

Cuando instale paquetes de Emacs con Guix los archivos de Elisp se encuentran en el directorio `share/emacs/site-lisp/` del perfil en el que se instalen. Las bibliotecas de

Elisp se ponen a disposición de Emacs a través de la variable de entorno `EMACSLOADPATH`, a la cual se le asigna un valor cuando se instale el propio Emacs.

Additionally, autoload definitions are automatically evaluated at the initialization of Emacs, by the Guix-specific `guix-emacs-autoload-packages` procedure. This procedure can be interactively invoked to have newly installed Emacs packages discovered, without having to restart Emacs. If, for some reason, you want to avoid auto-loading the Emacs packages installed with Guix, you can do so by running Emacs with the `--no-site-file` option (see Section "Init File" in *The GNU Emacs Manual*).

> **Nota:** Emacs can now compile packages natively. Under the default configuration, this means that Emacs packages will now be just-in-time (JIT) compiled as you use them, and the results stored in a subdirectory of your `user-emacs-directory`.
>
> Furthermore, the build system for Emacs packages transparently supports native compilation, but note, that `emacs-minimal`—the default Emacs for building packages—has been configured without native compilation. To natively compile your emacs packages ahead of time, use a transformation like `--with-input=emacs-minimal=emacs`.

## 2.5 Actualizar Guix

Para actualizar Guix ejecute:

```
guix pull
```

See Section 5.7 [Invocación de guix pull], page 57, para más información.

En una distribución distinta puede actualizar el daemon de construcción ejecutando:

```
sudo -i guix pull
```

seguido de (asumiendo que su distribución usa la herramienta de gestión de servicios systemd):

```
systemctl restart guix-daemon.service
```

En el Sistema Guix, la actualización del daemon se lleva a cabo con la reconfiguración el sistema (see Section 11.16 [Invocación de guix system], page 634).

# 3 Instalación del sistema

Esta sección explica cómo instalar el sistema Guix en una máquina. Guix, como gestor de paquetes, puede instalarse sobre un sistema GNU/Linux en ejecución, see Chapter 2 [Instalación], page 4.

## 3.1 Limitaciones

Consideramos que el sistema Guix está listo para un amplio rango de casos de uso, tanto de servidor como de escritorio. Las garantías que proporciona—actualizaciones transaccionales y vuelta atrás atómica, reproducibilidad—lo convierten en un cimiento sólido.

No obstante, antes de que proceda con la instalación, sea consciente de las siguientes limitaciones apreciables que se conocen en la versión c7888f5:

- Se proporcionan más y más servicios del sistema (see Section 11.10 [Servicios], page 275), pero pueden faltar algunos.
- Están disponibles GNOME, Xfce, LXDE y Enlightenment (see Section 11.10.9 [Servicios de escritorio], page 363), así como un número de gestores de ventanas X11. No obstante, actualmente falta KDE.

Más que una descarga de responsabilidades es una invitación a informar de problemas (¡e historias satisfactorias!), y para unirse a nosotras en su mejora. See Chapter 22 [Contribuir], page 734, para más información.

## 3.2 Consideraciones sobre el hardware

GNU Guix se enfoca en respetar la libertad de computación de las usuarias. Se construye sobre el núcleo Linux-libre, lo que significa que únicamente funciona hardware para el que existen controladores y firmware libres. Hoy en día, un amplio rango del hardware común funciona con GNU/Linux-libre—desde teclados a tarjetas gráficas a escáneres y controladoras Ethernet. Desafortunadamente, todavía hay áreas donde los fabricantes de hardware deniegan a las usuarias el control de su propia computación, y dicho hardware no funciona en el sistema Guix.

Una de las áreas principales donde faltan controladores o firmware libre son los dispositivos WiFi. Los dispositivos WiFi que se sabe que funcionan incluyen aquellos que usan los chips Atheros (AR9271 y AR7010), que corresponden al controlador `ath9k` de Linux-libre, y aquellos que usan los chips Broadcom/AirForce (BCM43xx con Wireless-Core Revisión 5), que corresponden al controlador `b43-open` de Linux-libre. Existe firmware libre para ambos, y está disponible por defecto en el sistema Guix, como parte de `%base-firmware` (see Section 11.3 [Referencia de operating-system], page 253).

The installer warns you early on if it detects devices that are known *not* to work due to the lack of free firmware or free drivers.

La Fundación del Software Libre (`https://www.fsf.org/`) patrocina *Respeta Su Libertad* (`https://www.fsf.org/ryf`) (RYF), un programa de certificación para productos hardware que respetan su libertad y su privacidad y se aseguran de que usted tenga el control sobre su dispositivo. Le recomendamos que compruebe la lista de dispositivos certificados RYF.

Otro recurso útil es el sitio web H-Node (`https://wwww.h-node.org/`). Contiene un catálogo de dispositivos hardware con información acerca su funcionalidad con GNU/Linux.

## 3.3 Instalación desde memoria USB y DVD

Se puede descargar una imagen de instalación ISO-9660 que puede ser escrita en una memoria USB o grabada en un DVD desde ‘`https://ftp.gnu.org/gnu/guix/guix-system-install-c7888f5.x86_`
donde puede sustituir `x86_64-linux` con uno de los siguientes valores:

`x86_64-linux`
: para un sistema GNU/Linux en CPUs compatibles con la arquitectura de 64-bits de Intel/AMD;

`i686-linux`
: para un sistema GNU/Linux en CPUs compatibles con la arquitectura de 32-bits de Intel.

Asegúrese de descargar el archivo `.sig` asociado y de verificar la autenticidad de la imagen contra él, más o menos así:

```
$ wget https://ftp.gnu.org/gnu/guix/guix-system-install-c7888f5.x86_64-linux.iso.sig
$ gpg --verify guix-system-install-c7888f5.x86_64-linux.iso.sig
```

Si la orden falla porque no dispone de la clave pública necesaria, entonces ejecute esta otra orden para importarla:

```
$ wget https://sv.gnu.org/people/viewgpg.php?user_id=15145 \
      -qO - | gpg --import -
```

y vuelva a ejecutar la orden `gpg --verify`.

Tenga en cuenta que un aviso del tipo “Esta clave no esta certificada con una firma de confianza” es normal.

Esta imagen contiene las herramientas necesarias para una instalación. Está pensada ara ser copiada *tal cual* a una memoria USB o DVD con espacio suficiente.

### Copiado en una memoria USB

Conecte una memoria USB de 1 GiB o más a su máquina, y determine su nombre de dispositivo. Asumiendo que la memoria USB es `/dev/sdX` copie la imagen con:

```
dd if=guix-system-install-c7888f5.x86_64-linux.iso of=/dev/sdX
sync
```

El acceso a `/dev/sdX` normalmente necesita privilegios de root.

### Grabación en un DVD

Introduzca un DVD en su máquina para grabarlo, y determine el nombre del dispositivo. Asumiendo que la unidad DVD es `/dev/srX`, copie la imagen con:

```
growisofs -dvd-compat -Z /dev/srX=guix-system-install-c7888f5.x86_64-linux.iso
```

El acceso a `/dev/srX` normalmente necesita privilegios de root.

### Arranque

Once this is done, you should be able to reboot the system and boot from the USB stick or DVD. The latter usually requires you to get in the BIOS or UEFI boot menu, where you can choose to boot from the USB stick. In order to boot from Libreboot, switch to the command mode by pressing the `c` key and type `search_grub usb`.

Sadly, on some machines, the installation medium cannot be properly booted and you only see a black screen after booting even after you waited for ten minutes. This may indicate that your machine cannot run Guix System; perhaps you instead want to install Guix on a foreign distro (see Section 2.1 [Instalación binaria], page 4). But don't give up just yet; a possible workaround is pressing the `e` key in the GRUB boot menu and appending `nomodeset` to the Linux bootline. Sometimes the black screen issue can also be resolved by connecting a different display.

See Section 3.8 [Instalación de Guix en una máquina virtual], page 30, si, en vez de esto, desea instalar el sistema Guix en una máquina virtual (VM).

## 3.4 Preparación para la instalación

Una vez que haya arrancado, puede usar el instalador gráfico guiado, el cual facilita la introducción al sistema (see Section 3.5 [Instalación gráfica guiada], page 23). Alternativamente, si ya es está familiarizada con GNU/Linux y desea más control que el que proporciona el instalador gráfico, puede seleccionar el proceso de instalación "manual" (see Section 3.6 [Instalación manual], page 25).

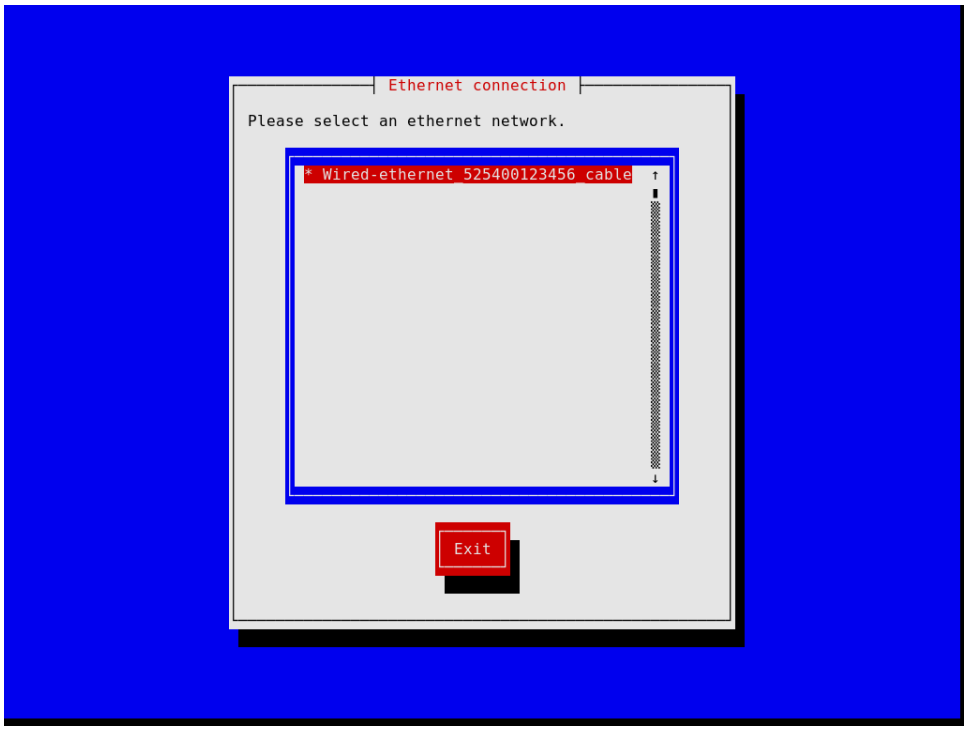
El instalador gráfico está disponible en TTY1. Puede obtener consolas de administración ("root") en los TTY 3 a 6 pulsando *`ctrl-alt-f3`*, *`ctrl-alt-f4`*, etc. TTY2 muestra esta documentación y se puede cambiar a dicha consola con *`ctrl-alt-f2`*. La documentación es explorable usando las órdenes del lector Info (see *Stand-alone GNU Info*). El sistema de instalación ejecuta el daemon GPM para ratones, el cual le permite seleccionar texto con el botón izquierdo y pegarlo con el botón central.

> **Nota:** La instalación requiere acceso a Internet de modo que cualquier dependencia de su configuración de sistema no encontrada pueda ser descargada. Véase la sección "Red" más adelante.

## 3.5 Instalación gráfica guiada

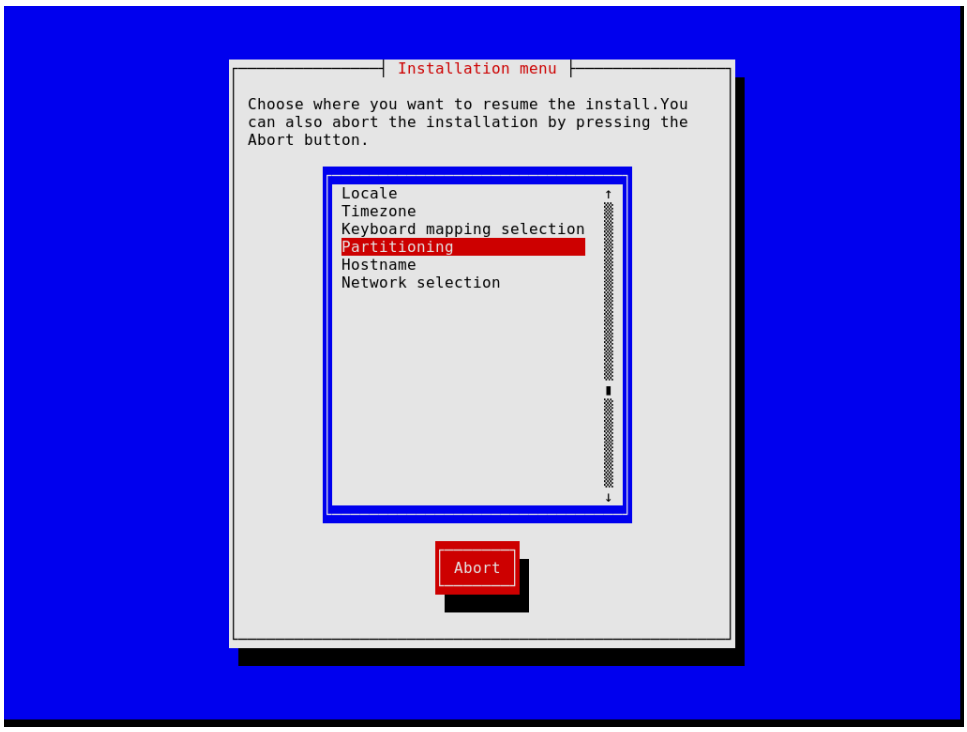
El instalador gráfico es una interfaz de usuaria basada en texto. Le guiará, con cajas de diálogo, a través de los pasos necesarios para instalar el sistema GNU Guix.

Las primeras cajas de diálogo le permiten configurar el sistema mientras lo usa durante la instalación: puede seleccionar el idioma, la distribución del teclado y configurar la red, la cual se usará durante la instalación. La siguiente imagen muestra el diálogo de configuración de red.




Los siguientes pasos le permitirán particionar su disco duro, como se muestra en la siguiente imagen, elegir si se usarán o no sistemas de archivos cifrados, introducir el nombre de la máquina, la contraseña de root y crear cuentas adicionales, entre otras cosas.

Tenga en cuenta que, en cualquier momento, el instalador le permite salir de la instalación actual y retomarla en un paso previo, como se muestra en la siguiente imagen.




Una vez haya finalizado, el instalador produce una configuración de sistema operativo y la muestra (see Section 11.2 [Uso de la configuración del sistema], page 244). En este punto puede pulsar "OK" y la instalación procederá. En caso de finalización satisfactoria, puede reiniciar con el nuevo sistema y disfrutarlo. ¡See Section 3.7 [Tras la instalación del sistema], page 30, para ver cómo proceder a continuación!

## 3.6 Instalación manual

Esta sección describe como podría instalar "manualmente" el sistema GNU Guix en su máquina. Esta opción requiere familiaridad con GNU/Linux, con el intérprete y con las herramientas de administración comunes. Si piensa que no es para usted, considere el uso del instalador gráfico guiado (see Section 3.5 [Instalación gráfica guiada], page 23).

The installation system provides root shells on TTYs 3 to 6; press *`ctrl-alt-f3`*, *`ctrl-alt-f4`*, and so on to reach them. It includes many common tools needed to install the system, but is also a full-blown Guix System. This means that you can install additional packages, should you need it, using `guix package` (see Section 5.2 [Invocación de guix package], page 36).

### 3.6.1 Distribución de teclado, red y particionado

Antes de instalar el sistema, puede desear ajustar la distribución del teclado, configurar la red y particionar el disco duro deseado. Esta sección le guiará durante este proceso.

#### 3.6.1.1 Distribución de teclado

La imagen de instalación usa la distribución de teclado QWERTY de los EEUU. Si desea cambiarla, puede usar la orden `loadkeys`. Por ejemplo, la siguiente orden selecciona la distribución de teclado para el castellano:

```
loadkeys es
```

Véanse los archivos bajo `/run/current-system/profile/share/keymaps` para la obtención de una lista de distribuciones de teclado disponibles. Ejecute `man loadkeys` para más información.

#### 3.6.1.2 Red

Ejecute la siguiente orden para ver los nombres asignados a sus interfaces de red:

```
ifconfig -a
```

. . . o, usando la orden específica de GNU/Linux `ip`:

```
ip address
```

El nombre de las interfaces de cable comienza con '`e`'; por ejemplo, la interfaz que corresponde a la primera controladora Ethernet en la placa se llama '`eno1`'. El nombre de las interfaces inalámbricas comienza con '`w`', como '`w1p2s0`'.

Conexión por cable

Para configurar una red por cable ejecute la siguiente orden, substituyendo *interfaz* con el nombre de la interfaz de cable que desea usar.

```
ifconfig interfaz up
```

. . . o, usando la orden específica de GNU/Linux `ip`:

```
ip link set interfaz up
```

Conexión sin cable

Para configurar una red inalámbrica, puede crear un archivo de configuración para la herramienta de configuración `wpa_supplicant` (su ruta no es importante) usando uno de los editores de texto disponibles como `nano`:

```
nano wpa_supplicant.conf
```

Como un ejemplo, la siguiente plantilla puede colocarse en este archivo y funcionará para muchas redes inalámbricas, siempre que se proporcione el SSID y la contraseña reales de la red a la que se va a conectar:

```
network={
  ssid="mi-ssid"
  key_mgmt=WPA-PSK
  psk="la contraseña de la red"
}
```

Inicie el servicio inalámbrico y lance su ejecución en segundo plano con la siguiente orden (sustituya *interfaz* por el nombre de la interfaz de red que desea usar):

```
wpa_supplicant -c wpa_supplicant.conf -i interfaz -B
```

Ejecute `man wpa_supplicant` para más información.

En este punto, necesita obtener una dirección IP. En una red donde las direcciones IP se asignan automáticamente mediante DHCP, puede ejecutar:

```
dhclient -v interfaz
```

Intente hacer ping a un servidor para comprobar si la red está funcionando correctamente:

```
ping -c 3 gnu.org
```

Configurar el acceso por red es casi siempre un requisito debido a que la imagen no contiene todo el software y las herramientas que puedan ser necesarias.

Si necesita que el acceso a HTTP y HTTPS se produzca a través de una pasarela ("proxy"), ejecute la siguiente orden:

```
herd set-http-proxy guix-daemon URL
```

donde *URL* es la URL de la pasarela, por ejemplo `http://example.org:8118`.

Si lo desea, puede continuar la instalación de forma remota iniciando un servidor SSH:

```
herd start ssh-daemon
```

Asegúrese de establecer una contraseña con `passwd`, o configure la verificación de clave pública de OpenSSH antes de ingresar al sistema.

#### 3.6.1.3 Particionado de discos

A menos que se haya realizado previamente, el siguiente paso es el particionado, y después dar formato a la/s partición/es deseadas.

La imagen de instalación contiene varias herramientas de particionado, incluyendo Parted (see Section "Overview" in *GNU Parted User Manual*), `fdisk` y `cfdisk`. Invoque su ejecución y configure el mapa de particiones deseado en su disco:

```
cfdisk
```

Si su disco usa el formato de tabla de particiones GUID (GPT) y tiene pensado instalar GRUB basado en BIOS (la opción predeterminada), asegúrese de tener una partición de arranque BIOS disponible (see Section "BIOS installation" in *GNU GRUB manual*).

Si en vez de eso desea GRUB basado en EFI, se requiere una *Partición del Sistema EFI* (ESP) con formato FAT32. Esta partición puede montarse en `/boot/efi` y debe tener la opción `esp` activa. Por ejemplo, en `parted`:

```
parted /dev/sda set 1 esp on
```

**Nota:**

¿No esta segura si usar GRUB basado en EFI o en BIOS? Si el directorio `/sys/firmware/efi` existe en la imagen de instalación, probablemente debería realizar una instalación EFI, usando `grub-efi-bootloader`. En otro caso, debe usar GRUB basado en BIOS, conocido como `grub-bootloader`. See Section 11.15 [Configuración del gestor de arranque], page 627, para más información sobre cargadores de arranque.

Once you are done partitioning the target hard disk drive, you have to create a file system on the relevant partition(s)[1]. For the ESP, if you have one and assuming it is `/dev/sda1`, run:

```
mkfs.fat -F32 /dev/sda1
```

[1] Currently Guix System only supports ext4, btrfs, JFS, F2FS, and XFS file systems. In particular, code that reads file system UUIDs and labels only works for these file system types.

El formato de sistema de archivos ext4 es el formato más ampliamente usado para el sistema de archivos raíz. Otros sistemas de archivos, como por ejemplo Btrfs, implementan compresión, la cual complementa adecuadamente la deduplicación de archivos que el daemon realiza de manera independiente al sistema de archivos(see Section 2.3 [Invocación de guix-daemon], page 12).

Preferentemente, asigne una etiqueta a los sistemas de archivos de modo que pueda referirse a ellos de forma fácil y precisa en las declaraciones `file-system` (see Section 11.4 [Sistemas de archivos], page 257). Esto se consigue habitualmente con la opción `-L` de `mkfs.ext4` y las ordenes relacionadas. Por tanto, asumiendo que la partición de la raíz es `/dev/sda2`, se puede crear un sistema de archivos con la etiqueta `mi-raiz` de esta manera:

```
mkfs.ext4 -L mi-raiz /dev/sda2
```

If you are instead planning to encrypt the root partition, you can use the Cryptsetup/LUKS utilities to do that (see `man cryptsetup` for more information).

> **Aviso:** While efforts are in progress to extend support to LUKS2, please note that Guix only supports devices of type LUKS1 at the moment. You can verify that your existing LUKS device is of the right type by running `cryptsetup luksDump` *`device`*. Alternatively, you can create a new LUKS1 device with `cryptsetup luksFormat --type luks1` *`device`*.

Assuming you want to store the root partition on `/dev/sda2`, the command sequence to format it as a LUKS1 partition would be along these lines:

```
cryptsetup luksFormat --type luks1 /dev/sda2
cryptsetup open /dev/sda2 my-partition
mkfs.ext4 -L my-root /dev/mapper/my-partition
```

Una vez hecho esto, monte el sistema de archivos deseado bajo `/mnt` con una orden como (de nuevo, asumiendo que `mi-raiz` es la etiqueta del sistema de archivos raíz):

```
mount LABEL=mi-raiz /mnt
```

Monte también cualquier otro sistema de archivos que desee usar en el sistema resultante relativamente a esta ruta. Si ha optado por `/boot/efi` como el punto de montaje de EFI, por ejemplo, ahora debe ser montada en `/mnt/boot/efi` para que `guix system init` pueda encontrarla más adelante.

Finally, if you plan to use one or more swap partitions (see Section 11.6 [Swap Space], page 265), make sure to initialize them with `mkswap`. Assuming you have one swap partition on `/dev/sda3`, you would run:

```
mkswap /dev/sda3
swapon /dev/sda3
```

De manera alternativa, puede usar un archivo de intercambio. Por ejemplo, asumiendo que en el nuevo sistema desea usar el archivo `/archivo-de-intercambio` como tal, ejecutaría[2]:

```
# Esto son 10GiB de espacio de intercambio. Ajuste "count" para
# cambiar el tamaño.
```

[2] Este ejemplo funcionará para muchos tipos de sistemas de archivos (por ejemplo, ext4). No obstante, para los sistemas de archivos con mecanismos de copia-durante-escritura (por ejemplo, btrfs) los pasos pueden ser diferentes. Para obtener más detalles, véanse las páginas de manual para `mkswap` y `swapon`.

```
dd if=/dev/zero of=/mnt/swapfile bs=1MiB count=10240
# Por seguridad, se le conceden permisos de lectura y escritura
# únicamente a root.
chmod 600 /mnt/swapfile
mkswap /mnt/swapfile
swapon /mnt/swapfile
```

Fíjese que si ha cifrado la partición raíz y creado un archivo de intercambio en su sistema de archivos como se ha descrito anteriormente, el cifrado también protege al archivo de intercambio, como a cualquier archivo en dicho sistema de archivos.

### 3.6.2 Procedimiento de instalación

Con las particiones deseadas listas y la raíz deseada montada en `/mnt`, estamos preparadas para empezar. Primero, ejecute:

```
herd start cow-store /mnt
```

Esto activa la copia-durante-escritura en `/gnu/store`, de modo que los paquetes que se añadan durante la fase de instalación se escriban en el disco montado en `/mnt` en vez de permanecer en memoria. Esto es necesario debido a que la primera fase de la orden `guix system init` (vea más adelante) implica descargas o construcciones en `/gnu/store`, el cual, inicialmente, está un sistema de archivos en memoria.

Next, you have to edit a file and provide the declaration of the operating system to be installed. To that end, the installation system comes with three text editors. We recommend GNU nano (see *GNU nano Manual*), which supports syntax highlighting and parentheses matching; other editors include mg (an Emacs clone), and nvi (a clone of the original BSD `vi` editor). We strongly recommend storing that file on the target root file system, say, as `/mnt/etc/config.scm`. Failing to do that, you will have lost your configuration file once you have rebooted into the newly-installed system.

See Section 11.2 [Uso de la configuración del sistema], page 244, para hacerse una idea del archivo de configuración. Las configuraciones de ejemplo mencionadas en esa sección están disponibles bajo `/etc/configuration` en la imagen de instalación. Por tanto, para empezar con una configuración del sistema que proporcione un servidor gráfico (un sistema de "escritorio"), puede ejecutar algo parecido a estas órdenes:

```
# mkdir /mnt/etc
# cp /etc/configuration/desktop.scm /mnt/etc/config.scm
# nano /mnt/etc/config.scm
```

Debe prestar atención a lo que su archivo de configuración contiene, y en particular:

- Make sure the `bootloader-configuration` form refers to the targets you want to install GRUB on. It should mention `grub-bootloader` if you are installing GRUB in the legacy way, or `grub-efi-bootloader` for newer UEFI systems. For legacy systems, the `targets` field contain the names of the devices, like `(list "/dev/sda")`; for UEFI systems it names the paths to mounted EFI partitions, like `(list "/boot/efi")`; do make sure the paths are currently mounted and a `file-system` entry is specified in your configuration.
- Asegúrese que las etiquetas de su sistema de archivos corresponden con el valor de sus campos `device` respectivos en su configuración `file-system`, asumiendo que su

configuración `file-system` usa el procedimiento `file-system-label` en su campo `device`.

- Si hay particiones cifradas o en RAID, asegúrese de añadir un campo `mapped-devices` para describirlas (see Section 11.5 [Dispositivos traducidos], page 263).

Una vez haya terminado de preparar el archivo de configuración, el nuevo sistema debe ser inicializado (recuerde que el sistema de archivos raíz deseado está montado bajo `/mnt`):

```
guix system init /mnt/etc/config.scm /mnt
```

Esto copia todos los archivos necesarios e instala GRUB en `/dev/sdX`, a menos que proporcione la opción `--no-bootloader`. Para más información, see Section 11.16 [Invocación de guix system], page 634. Esta orden puede desencadenar descargas o construcciones de paquetes no encontrados, lo cual puede tomar algún tiempo.

Una vez que la orden se complete—¡y, deseablemente, de forma satisfactoria!—puede ejecutar `reboot` y arrancar con el nuevo sistema. La contraseña de `root` en el nuevo sistema está vacía inicialmente; otras contraseñas de usuarias tienen que ser inicializadas ejecutando la orden `passwd` como `root`, a menos que en su configuración se especifique de otra manera (see [user-account-password], page 269). ¡See Section 3.7 [Tras la instalación del sistema], page 30, para proceder a continuación!

## 3.7 Tras la instalación del sistema

Success, you've now booted into Guix System! You can upgrade the system whenever you want by running:

```
guix pull
sudo guix system reconfigure /etc/config.scm
```

This builds a new system *generation* with the latest packages and services.

Now, see Section 11.1 [Getting Started with the System], page 242, and join us on `#guix` on the Libera.Chat IRC network or on `guix-devel@gnu.org` to share your experience!

## 3.8 Instalación de Guix en una máquina virtual

Si desea instalar el sistema Guix en una máquina virtual (VM) o en un servidor privado virtual (VPS) en vez de en su preciada máquina, esta sección es para usted.

Si quiere arrancar una VM QEMU (`https://qemu.org/`) para instalar el sistema Guix en una imagen de disco, siga estos pasos:

1. Primero, obtenga y descomprima la imagen de instalación del sistema Guix como se ha descrito previamente (see Section 3.3 [Instalación desde memoria USB y DVD], page 22).
2. Cree una imagen de disco que contendrá el sistema instalado. Para crear una imagen de disco con formato qcow2, use la orden `qemu-img`:

   ```
   qemu-img create -f qcow2 guix-system.img 50G
   ```

   El archivo que obtenga será mucho menor de 50GB (típicamente menos de 1MB), pero crecerá cuando el dispositivo de almacenamiento virtualizado se vaya llenando.
3. Arranque la imagen de instalación USB en una máquina virtual:

   ```
   qemu-system-x86_64 -m 1024 -smp 1 -enable-kvm \
   ```

```
-nic user,model=virtio-net-pci -boot menu=on,order=d \
-drive file=guix-system.img \
-drive media=cdrom,readonly=on,file=guix-system-install-c7888f5.system.iso
```

`-enable-kvm` es opcional, pero mejora el rendimiento significativamente, see Section 11.18 [Ejecutar Guix en una máquina virtual], page 647.

4. Ahora es root en la VM, prosiga con el procedimiento de instalación. See Section 3.4 [Preparación para la instalación], page 23, y siga las instrucciones.

Una vez complete la instalación, puede arrancar el sistema que está en la imagen `guix-system.img`. See Section 11.18 [Ejecutar Guix en una máquina virtual], page 647, para información sobre cómo hacerlo.

## 3.9 Construcción de la imagen de instalación

La imagen de instalación descrita anteriormente se construyó usando la orden `guix system`, específicamente:

```
guix system image -t iso9660 gnu/system/install.scm
```

Eche un vistazo a `gnu/system/install.scm` en el árbol de fuentes, y vea también Section 11.16 [Invocación de guix system], page 634, para más información acerca de la imagen de instalación.

## 3.10 Construcción de la imagen de instalación para placas ARM

Muchos dispositivos con procesador ARM necesitan una variante específica del cargador de arranque U-Boot (`https://www.denx.de/wiki/U-Boot/`).

Si construye una imagen de disco y el cargador de arranque no está disponible de otro modo (en otra unidad de arranque, etc.), es recomendable construir una imagen que incluya el cargador, específicamente:

```
guix system image --system=armhf-linux -e '((@ (gnu system install) os-with-u-boot) (@
```

`A20-OLinuXino-Lime2` es el nombre de la placa. Si especifica una placa no válida, una lista de placas posibles será mostrada.

# 4 Empezando

Es probable que haya llegado a esta sección o bien porque haya instalado Guix sobre otra distribución (see Chapter 2 [Instalación], page 4), o bien porque haya instalado el sistema Guix completo (see Chapter 3 [Instalación del sistema], page 21). Es hora de dar sus primeros pasos con Guix; esta sección intenta ser de ayuda con estos primeros pasos y proporcionar una primera impresión sobre Guix.

Guix está orientado a instalar software, por lo que probablemente la primera cosa que deseará hacer es mirar el software disponible. Digamos, por ejemplo, que busca un editor de texto. Para ello puede ejecutar:

```
guix search texto editor
```

Esta orden le muestra cierto número de *paquetes* asociados a dicha búsqueda, mostrando con cada uno de ellos su nombre, versión, una descripción e información adicional. Una vez que ha encontrado el que quiere usar, digamos Emacs (je je je), puede seguir adelante e instalarlo (ejecute esta orden con su cuenta de usuaria habitual, *¡no necesita privilegios de administración o usar la cuenta "root"!*):

```
guix install emacs
```

You've installed your first package, congrats! The package is now visible in your default *profile*, `$HOME/.guix-profile`—a profile is a directory containing installed packages. In the process, you've probably noticed that Guix downloaded pre-built binaries; or, if you explicitly chose to *not* use pre-built binaries, then probably Guix is still building software (see Section 5.3 [Sustituciones], page 46, for more info).

Si no está usando el sistema Guix, la orden `guix install` debe haber mostrado este consejo:

```
consejo: Considere proporcionar valor a las variables de entorno necesarias ejecutando
     GUIX_PROFILE="$HOME/.guix-profile"
     . "$GUIX_PROFILE/etc/profile"

Alternativamente, véase `guix package --search-paths -p "$HOME/.guix-profile"'.
```

Indeed, you must now tell your shell where `emacs` and other programs installed with Guix are to be found. Pasting the two lines above will do just that: it will add `$HOME/.guix-profile/bin`—which is where the installed package is—to the `PATH` environment variable. You can paste these two lines in your shell so they take effect right away, but more importantly you should add them to `~/.bash_profile` (or equivalent file if you do not use Bash) so that environment variables are set next time you spawn a shell. You only need to do this once and other search paths environment variables will be taken care of similarly—e.g., if you eventually install `python` and Python libraries, `GUIX_PYTHONPATH` will be defined.

Puede continuar instalando paquetes cuando quiera. Para enumerar los paquetes instalados, ejecute:

```
guix package --list-installed
```

Para borrar un paquete, deberá ejecutar `guix remove`. Una característica distintiva es la capacidad de *revertir* cualquier operación que haya realizado—instalación, borrado, actualización—simplemente escribiendo:

```
guix package --roll-back
```

Esto es debido al hecho de que cada operación en realidad es una *transacción* que crea una nueva *generación*. Estas generaciones y la diferencia entre ellas puede ser visualizada mediante la ejecución de:

```
guix package --list-generations
```

¡Ahora ya conoce lo básico sobre gestión de paquetes!

> **Más allá:** See Chapter 5 [Gestión de paquetes], page 35, para más información sobre gestión de paquetes. Puede que le guste la gestión *declarativa* de paquetes con `guix package --manifest`, la gestión de distintos *perfiles* con `--profile`, el borrado de generaciones antiguas, la recolección de basura y otras interesantes características que le serán útiles una vez que se familiarice con Guix. Si es desarrolladora, see Chapter 7 [Desarrollo], page 79, para herramientas adicionales. Y si simplemente tiene curiosidad, see Section 5.1 [Características], page 35, para echar un vistazo a su funcionamiento interno.
>
> You can also manage the configuration of your entire *home environment*—your user "dot files", services, and packages—using Guix Home. See Chapter 13 [Home Configuration], page 671, to learn more about it!

Una vez que haya instalado un conjunto de paquetes, deseará *actualizarlos* periódicamente a la última y mejor versión. Para hacerlo, primero debe obtener la última revisión de Guix y su colección de paquetes:

```
guix pull
```

El resultado final es un nuevo ejecutable `guix`, que se encuentra en `~/.config/guix/current/bin`. A menos que use el sistema Guix, asegúrese de seguir el consejo que la orden muestra y, al igual que vimos previamente, pegue estas dos líneas en su terminal y en `.bash_profile`:

```
GUIX_PROFILE="$HOME/.config/guix/current"
. "$GUIX_PROFILE/etc/profile"
```

También le debe indicar a su intérprete que haga referencia a este nuevo `guix`:

```
hash guix
```

En este momento ya está ejecutando una nueva instalación de Guix. Por lo tanto puede continuar y actualizar todos los paquetes que haya instalado previamente:

```
guix upgrade
```

Mientras se ejecuta esta orden verá que se descargan binarios (o quizá que algunos paquetes se construyen) y tras cierto tiempo obtendrá los paquetes actualizados. ¡Recuerde que siempre puede revertir esta acción en caso de que uno de los paquetes no sea de su agrado!

Puede mostrar la revisión exacta de Guix que está ejecutando mediante la orden:

```
guix describe
```

La información que muestra contiene *todo lo necesario para reproducir exactamente el mismo Guix*, ya sea en otro momento o en una máquina diferente.

> **Más allá:** See Section 5.7 [Invocación de guix pull], page 57, para más información. See Chapter 6 [Canales], page 69, para saber cómo especificar *canales* adicionales de los que obtener paquetes, cómo replicar Guix y más información.

La orden `time-machine` también le puede ser de utilidad (see Section 5.8 [Invocación de guix time-machine], page 61).

Si ha instalado el sistema Guix, una de las primeras cosas que probablemente desee hacer es actualizar su sistema. Una vez que haya ejecutado `guix pull` con la última versión de Guix, puede actualizar su sistema de este modo:

```
sudo guix system reconfigure /etc/config.scm
```

Upon completion, the system runs the latest versions of its software packages. Just like for packages, you can always *roll back* to a previous generation *of the whole system*. See Section 11.1 [Getting Started with the System], page 242, to learn how to manage your system.

¡Ya sabe lo suficiente para empezar!

**Recursos:** El resto de este manual proporciona una referencia para Guix al completo. Esta es una lista de recursos adicionales que le pueden resultar útiles:

- See *The GNU Guix Cookbook*, que contiene una lista de recetas tipo "cómo se hace" para una variedad de casos.
- La tarjeta de referencia de GNU Guix (`https://guix.gnu.org/guix-refcard.pdf`) enumera en dos páginas la mayor parte de las órdenes y opciones que pueda necesitar en cualquier momento.
- La página web contiene medios audiovisuales instructivos (`https://guix.gnu.org/es/videos/`) sobre temas como el uso diario de Guix, cómo obtener ayuda y cómo contribuir.
- See Chapter 14 [Documentación], page 707, para aprender cómo acceder a la documentación en su máquina.

¡Esperamos que disfrute Guix tanto como la comunidad disfruta en su construcción!

# 5 Gestión de paquetes

El propósito de GNU Guix es permitir a las usuarias instalar, actualizar y borrar fácilmente paquetes de software, sin tener que conocer acerca de sus procedimientos de construcción o dependencias. Guix también va más allá de este conjunto obvio de características.

Este capítulo describe las principales características de Guix, así como las herramientas de gestión de paquetes que ofrece. Junto a la interfaz de línea de órdenes descrita a continuación (see Section 5.2 [Invocación de guix package], page 36, también puede usar la interfaz Emacs-Guix (see *The Emacs Guix Reference Manual*), tras la instalación del paquete `emacs-guix` (ejecute la orden *`M-x guix-help`* para iniciarse en su uso):

```
guix install emacs-guix
```

## 5.1 Características

Se asume que ya ha dado sus primeros pasos con Guix (see Chapter 4 [Empezando], page 32) y desea obtener información general sobre cómo funciona la implementación internamente.

Cuando se usa Guix, cada paquete se encuentra en el *almacén de paquetes*, en su propio directorio—algo que se asemeja a `/gnu/store/xxx-paquete-1.2`, donde `xxx` es una cadena en base32.

En vez de referirse a estos directorios, las usuarias tienen su propio *perfil*, el cual apunta a los paquetes que realmente desean usar. Estos perfiles se almacenan en el directorio de cada usuaria, en `$HOME/.guix-profile`.

Por ejemplo, `alicia` instala GCC 4.7.2. Como resultado, `/home/alicia/.guix-profile/bin/gcc` apunta a `/gnu/store/...-gcc-4.7.2/bin/gcc`. Ahora, en la misma máquina, `rober` ha instalado ya GCC 4.8.0. El perfil de `rober` simplemente sigue apuntando a `/gnu/store/...-gcc-4.8.0/bin/gcc`—es decir, ambas versiones de GCC pueden coexistir en el mismo sistema sin ninguna interferencia.

La orden `guix package` es la herramienta central para gestión de paquetes (see Section 5.2 [Invocación de guix package], page 36). Opera en los perfiles de usuaria, y puede ser usada *con privilegios de usuaria normal*.

La orden proporciona las operaciones obvias de instalación, borrado y actualización. Cada invocación es en realidad una *transacción*: o bien la operación especificada se realiza satisfactoriamente, o bien nada sucede. Por tanto, si el proceso `guix package` es finalizado durante una transacción, o un fallo eléctrico ocurre durante la transacción, el perfil de usuaria permanece en su estado previo, y permanece usable.

In addition, any package transaction may be *rolled back*. So, if, for example, an upgrade installs a new version of a package that turns out to have a serious bug, users may roll back to the previous instance of their profile, which was known to work well. Similarly, the global system configuration on Guix is subject to transactional upgrades and roll-back (see Section 11.1 [Getting Started with the System], page 242).

Todos los paquetes en el almacén de paquetes pueden ser *eliminados por el recolector de basura*. Guix puede determinar a qué paquetes hacen referencia todavía los perfiles de usuarias, y eliminar aquellos que, de forma demostrable, no se haga referencia en ningún perfil (see Section 5.6 [Invocación de guix gc], page 53). Las usuarias pueden también

borrar explícitamente generaciones antiguas de su perfil para que los paquetes a los que hacen referencia puedan ser recolectados.

Guix toma una aproximación *puramente funcional* en la gestión de paquetes, como se describe en la introducción (see Chapter 1 [Introducción], page 1). Cada nombre de directorio de paquete en `/gnu/store` contiene un hash de todas las entradas que fueron usadas para construir el paquete—compilador, bibliotecas, guiones de construcción, etc. Esta correspondencia directa permite a las usuarias asegurarse que una instalación dada de un paquete corresponde al estado actual de su distribución. Esto también ayuda a maximizar la *reproducibilidad de la construcción*: gracias al uso de entornos aislados de construcción, una construcción dada probablemente generará archivos idénticos bit-a-bit cuando se realice en máquinas diferentes (see Section 2.3 [Invocación de guix-daemon], page 12).

Estos cimientos permiten a Guix ofrecer *despliegues transparentes de binarios/fuentes*. Cuando un binario pre-construido para un elemento de `/gnu/store` está disponible para descarga de una fuente externa—una *sustitución*, Guix simplemente lo descarga y desempaqueta; en otro caso construye el paquete de las fuentes, localmente (see Section 5.3 [Sustituciones], page 46). Debido a que los resultados de construcción son normalmente reproducibles bit-a-bit, las usuarias no tienen que confiar en los servidores que proporcionan sustituciones: pueden forzar una construcción local y *retar* a las proveedoras (see Section 9.12 [Invocación de guix challenge], page 230).

Control over the build environment is a feature that is also useful for developers. The `guix shell` command allows developers of a package to quickly set up the right development environment for their package, without having to manually install the dependencies of the package into their profile (see Section 7.1 [Invoking guix shell], page 79).

Todo Guix y sus definiciones de paquetes están bajo control de versiones, y `guix pull` le permite "viajar en el tiempo" por la historia del mismo Guix (see Section 5.7 [Invocación de guix pull], page 57). Esto hace posible replicar una instancia de Guix en una máquina diferente o en un punto posterior del tiempo, lo que a su vez le permite *replicar entornos de software completos*, mientras que mantiene un preciso *seguimiento de la procedencia* del software.

## 5.2 Invocación de `guix package`

The `guix package` command is the tool that allows users to install, upgrade, and remove packages, as well as rolling back to previous configurations. These operations work on a user *profile*—a directory of installed packages. Each user has a default profile in `$HOME/.guix-profile`. The command operates only on the user's own profile, and works with normal user privileges (see Section 5.1 [Características], page 35). Its syntax is:

```
guix package opciones
```

Primariamente, *opciones* especifica las operaciones a ser realizadas durante la transacción. Al completarse, un nuevo perfil es creado, pero las *generaciones* previas del perfil permanecen disponibles, en caso de que la usuaria quisiera volver atrás.

Por ejemplo, para borrar `lua` e instalar `guile` y `guile-cairo` en una única transacción:

```
guix package -r lua -i guile guile-cairo
```

Para su conveniencia, también se proporcionan los siguientes alias:

- `guix search` es un alias de `guix package -s`,

- `guix install` es un alias de `guix package -i`,
- `guix remove` es un alias de `guix package -r`,
- `guix upgrade` es un alias de `guix package -u`
- y `guix show` es un alias de `guix package --show=`.

Estos alias tienen menos capacidad expresiva que `guix package` y proporcionan menos opciones, por lo que en algunos casos es probable que desee usar `guix package` directamente.

`guix package` también proporciona una *aproximación declarativa*, donde la usuaria especifica el conjunto exacto de paquetes a poner disponibles y la pasa a través de la opción `--manifest` (see [profile-manifest], page 40).

Para cada usuaria, un enlace simbólico al perfil predeterminado de la usuaria es creado en `$HOME/.guix-profile`. Este enlace simbólico siempre apunta a la generación actual del perfil predeterminado de la usuaria. Por lo tanto, las usuarias pueden añadir `$HOME/.guix-profile/bin` a su variable de entorno `PATH`, etcétera.

Si no está usando el sistema Guix, considere la adición de las siguientes líneas en su `~/.bash_profile` (see Section "Bash Startup Files" in *The GNU Bash Reference Manual*) de manera que los nuevos intérpretes que ejecute obtengan todas las definiciones correctas de las variables de entorno:

```
GUIX_PROFILE="$HOME/.guix-profile" ; \
source "$GUIX_PROFILE/etc/profile"
```

En una configuración multiusuaria, los perfiles de usuaria se almacenan en un lugar registrado como una *raíz del sistema de archivos*, a la que apunta `$HOME/.guix-profile` (see Section 5.6 [Invocación de guix gc], page 53). Ese directorio normalmente es *localstatedir*`/guix/profiles/per-user/`*usuaria*, donde *localstatedir* es el valor pasado a `configure` como `--localstatedir` y *usuaria* es el nombre de usuaria. El directorio `per-user` se crea cuando se lanza `guix-daemon`, y el subdirectorio *usuaria* es creado por `guix package`.

Las *opciones* pueden ser las siguientes:

`--install=`*paquete* `...`
`-i` *paquete* `...`

Instala los *paquete*s especificados.

Each *package* may specify a simple package name, such as `guile`, optionally followed by an at-sign and version number, such as `guile@3.0.7` or simply `guile@3.0`. In the latter case, the newest version prefixed by `3.0` is selected.

If no version number is specified, the newest available version will be selected. In addition, such a *package* specification may contain a colon, followed by the name of one of the outputs of the package, as in `gcc:doc` or `binutils@2.22:lib` (see Section 5.4 [Paquetes con múltiples salidas], page 51).

Packages with a corresponding name (and optionally version) are searched for among the GNU distribution modules (see Section 8.1 [Módulos de paquetes], page 101).

Alternatively, a *package* can directly specify a store file name such as `/gnu/store/...-guile-3.0.7`, as produced by, e.g., `guix build`.

A veces los paquetes tienen *entradas propagadas*: estas son las dependencias que se instalan automáticamente junto al paquete requerido (see [package-propagated-inputs], page 107, para información sobre las entradas propagadas en las definiciones de paquete).

Un ejemplo es la biblioteca GNU MPC: sus archivos de cabecera C hacen referencia a los de la biblioteca GNU MPFR, que a su vez hacen referencia a los de la biblioteca GMP. Por tanto, cuando se instala MPC, las bibliotecas MPFR y GMP también se instalan en el perfil; borrar MPC también borra MPFR y GMP—a menos que también se hayan instalado explícitamente por la usuaria.

Por otra parte, los paquetes a veces dependen de la definición de variables de entorno para sus rutas de búsqueda (véase a continuación la explicación de `--seach-paths`). Cualquier definición de variable de entorno que falte o sea posiblemente incorrecta se informa aquí.

`--install-from-expression=`*exp*
`-e` *exp*
: Instala el paquete al que *exp* evalúa.

  *exp* must be a Scheme expression that evaluates to a `<package>` object. This option is notably useful to disambiguate between same-named variants of a package, with expressions such as `(@ (gnu packages commencement) guile-final)`.

  Fíjese que esta opción instala la primera salida del paquete especificado, lo cual puede ser insuficiente cuando se necesita una salida específica de un paquete con múltiples salidas.

`--install-from-file=`*archivo*
`-f` *archivo*
: Instala el paquete que resulta de evaluar el código en *archivo*.

  Como un ejemplo, *archivo* puede contener una definición como esta (see Section 8.2 [Definición de paquetes], page 102):

```
(use-modules (guix)
             (guix build-system gnu)
             (guix licenses))

(package
  (name "hello")
  (version "2.10")
  (source (origin
            (method url-fetch)
            (uri (string-append "mirror://gnu/hello/hello-" version
                                ".tar.gz"))
            (sha256
             (base32
              "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i"))))
  (build-system gnu-build-system)
  (synopsis "Hello, GNU world: An example GNU package")
  (description "Guess what GNU Hello prints!")
  (home-page "http://www.gnu.org/software/hello/")
```

```
        (license gpl3+))
```

Developers may find it useful to include such a `guix.scm` file in the root of their project source tree that can be used to test development snapshots and create reproducible development environments (see Section 7.1 [Invoking guix shell], page 79).

El *archivo* puede contener también una representación en JSON de una o más definiciones de paquetes. Ejecutar `guix package -f` en `hello.json` con el contenido mostrado a continuación provocará la instalación del paquete `greeter` tras la construcción de `myhello`:

```
[
  {
    "name": "myhello",
    "version": "2.10",
    "source": "mirror://gnu/hello/hello-2.10.tar.gz",
    "build-system": "gnu",
    "arguments": {
      "tests?": false
    },
    "home-page": "https://www.gnu.org/software/hello/",
    "synopsis": "Hello, GNU world: An example GNU package",
    "description": "GNU Hello prints a greeting.",
    "license": "GPL-3.0+",
    "native-inputs": ["gettext"]
  },
  {
    "name": "greeter",
    "version": "1.0",
    "source": "mirror://gnu/hello/hello-2.10.tar.gz",
    "build-system": "gnu",
    "arguments": {
      "test-target": "foo",
      "parallel-build?": false
    },
    "home-page": "https://example.com/",
    "synopsis": "Greeter using GNU Hello",
    "description": "This is a wrapper around GNU Hello.",
    "license": "GPL-3.0+",
    "inputs": ["myhello", "hello"]
  }
]
```

`--remove=`*paquete* `...`
`-r` *paquete* `...`

Borra los *paquete*s especificados.

Como en `--install`, cada *paquete* puede especificar un número de versión y/o un nombre de salida además del nombre del paquete. Por ejemplo, `-r glibc:debug` eliminaría la salida `debug` de `glibc`.

`--upgrade[=`*regexp* `...]`
`-u [`*regexp* `...]`

Actualiza todos los paquetes instalados. Si se especifica una o más expresiones regular *regexp*, actualiza únicamente los paquetes instalados cuyo nombre es aceptado por *regexp*. Véase también la opción `--do-not-upgrade` más adelante.

Tenga en cuenta que esto actualiza los paquetes a la última versión encontrada en la distribución instalada actualmente. Para actualizar su distribución, debe ejecutar regularmente `guix pull` (see Section 5.7 [Invocación de guix pull], page 57).

Al actualizar, las transformaciones que se aplicaron originalmente al crear el perfil se aplican de nuevo de manera automática (see Section 9.1.2 [Opciones de transformación de paquetes], page 184). Por ejemplo, asumiendo que hubiese instalado Emacs a partir de la última revisión de su rama de desarrollo con:

```
guix install emacs-next --with-branch=emacs-next=master
```

La próxima vez que ejecute `guix upgrade` Guix obtendrá de nuevo la última revisión de la rama de desarrollo de Emacs y construirá `emacs-next` a partir de ese código.

Tenga en cuenta que las opciones de transformación, como por ejemplo `--with-branch` y `--with-source`, dependen de un estado externo; es su responsabilidad asegurarse de que funcionen de la manera esperada. También puede deshacer las transformaciones aplicadas a un paquete con la siguiente orden:

```
guix install paquete
```

`--do-not-upgrade[=`*regexp* `...]`

Cuando se usa junto a la opción `--upgrade`, *no* actualiza ningún paquete cuyo nombre sea aceptado por *regexp*. Por ejemplo, para actualizar todos los paquetes en el perfil actual excepto aquellos que contengan la cadena "emacs":

```
$ guix package --upgrade . --do-not-upgrade emacs
```

`--manifest=`*archivo*
`-m` *archivo*

Crea una nueva generación del perfil desde el objeto de manifiesto devuelto por el código Scheme en *archivo*. Esta opción puede repetirse varias veces, en cuyo caso los manifiestos se concatenan.

Esto le permite *declarar* los contenidos del perfil en vez de construirlo a través de una secuencia de `--install` y órdenes similares. La ventaja es que *archivo* puede ponerse bajo control de versiones, copiarse a máquinas diferentes para reproducir el mismo perfil, y demás.

*archivo* debe devolver un objeto *manifest*, que es básicamente una lista de paquetes:

```
(use-package-modules guile emacs)

(packages->manifest
 (list emacs
       guile-2.0
```

```
;; Usa una salida específica del paquete.
(list guile-2.0 "debug")))
```

See Section 8.4 [Writing Manifests], page 119, for information on how to write a manifest. See [export-manifest], page 45, to learn how to obtain a manifest file from an existing profile.

`--roll-back`

Vuelve a la *generación* previa del perfil—es decir, revierte la última transacción.

Cuando se combina con opciones como `--install`, la reversión atrás ocurre antes que cualquier acción.

Cuando se vuelve atrás en la primera generación que realmente contiene paquetes instalados, se hace que el perfil apunte a la *generación cero*, la cual no contiene ningún archivo a excepción de sus propios metadatos.

Después de haber vuelto atrás, instalar, borrar o actualizar paquetes sobreescribe las generaciones futuras previas. Por tanto, la historia de las generaciones en un perfil es siempre linear.

`--switch-generation=`*patrón*

`-S` *patrón* Cambia a una generación particular definida por el *patrón*.

*patrón* puede ser tanto un número de generación como un número prefijado con "+" o "-". Esto último significa: mueve atrás/hacia delante el número especificado de generaciones. Por ejemplo, si quiere volver a la última generación antes de `--roll-back`, use `--switch-generation=+1`.

La diferencia entre `--roll-back` y `--switch-generation=-1` es que `--switch-generation` no creará una generación cero, así que si la generación especificada no existe, la generación actual no se verá cambiada.

`--search-paths[=`*tipo*`]`

Informa de variables de entorno, en sintaxis Bash, que pueden necesitarse para usar el conjunto de paquetes instalado. Estas variables de entorno se usan para especificar las *rutas de búsqueda* para archivos usadas por algunos de los paquetes.

For example, GCC needs the `CPATH` and `LIBRARY_PATH` environment variables to be defined so it can look for headers and libraries in the user's profile (see Section "Environment Variables" in *Using the GNU Compiler Collection (GCC)*). If GCC and, say, the C library are installed in the profile, then `--search-paths` will suggest setting these variables to *`profile`*`/include` and *`profile`*`/lib`, respectively (see Section 8.8 [Search Paths], page 154, for info on search path specifications associated with packages.)

El caso de uso típico es para definir estas variables de entorno en el intérprete de consola:

```
$ eval $(guix package --search-paths)
```

*tipo* puede ser `exact`, `prefix` o `suffix`, lo que significa que las definiciones de variables de entorno devueltas serán respectivamente las configuraciones exactas, prefijos o sufijos del valor actual de dichas variables. Cuando se omite, el valor predeterminado de *tipo* es `exact`.

Esta opción puede usarse para calcular las rutas de búsqueda *combinadas* de varios perfiles. Considere este ejemplo:

```
$ guix package -p foo -i guile
$ guix package -p bar -i guile-json
$ guix package -p foo -p bar --search-paths
```

La última orden informa sobre la variable `GUILE_LOAD_PATH`, aunque, tomada individualmente, ni `foo` ni `bar` hubieran llevado a esa recomendación.

`--profile=`*perfil*
`-p` *perfil* Usa *perfil* en vez del perfil predeterminado de la usuaria.

*perfil* debe ser el nombre de un archivo que se creará tras completar las tareas. Concretamente, *perfil* sera simplemente un enlace simbólico ("symlink") que apunta al verdadero perfil en el que se instalan los paquetes:

```
$ guix install hello -p ~/código/mi-perfil
...
$ ~/código/mi-perfil/bin/hello
¡Hola mundo!
```

Todo lo necesario para deshacerse del perfil es borrar dicho enlace simbólico y sus enlaces relacionados que apuntan a generaciones específicas:

```
$ rm ~/código/mi-perfil ~/código/mi-perfil-*-link
```

`--list-profiles`
Enumera los perfiles de la usuaria:

```
$ guix package --list-profiles
/home/carlos/.guix-profile
/home/carlos/código/mi-perfil
/home/carlos/código/perfil-desarrollo
/home/carlos/tmp/prueba
```

Cuando se ejecuta como root, enumera todos los perfiles de todas las usuarias.

`--allow-collisions`
Permite colisiones de paquetes en el nuevo perfil. ¡Úselo bajo su propio riesgo!

Por defecto, `guix package` informa como un error las *colisiones* en el perfil. Las colisiones ocurren cuando dos o más versiones diferentes o variantes de un paquete dado se han seleccionado para el perfil.

`--bootstrap`
Use el Guile usado para el lanzamiento para construir el perfil. Esta opción es util únicamente a las desarrolladoras de la distribución.

Además de estas acciones, `guix package` acepta las siguientes opciones para consultar el estado actual de un perfil, o la disponibilidad de paquetes:

`--search=`*regexp*
`-s` *regexp* Enumera los paquetes disponibles cuyo nombre, sinopsis o descripción corresponde con *regexp* (sin tener en cuenta la capitalización), ordenados por relevancia. Imprime todos los metadatos de los paquetes coincidentes en formato `recutils` (see *GNU recutils manual*).

Esto permite extraer campos específicos usando la orden `recsel`, por ejemplo:

```
$ guix package -s malloc | recsel -p name,version,relevance
name: jemalloc
version: 4.5.0
relevance: 6

name: glibc
version: 2.25
relevance: 1

name: libgc
version: 7.6.0
relevance: 1
```

De manera similar, para mostrar el nombre de todos los paquetes disponibles bajo los términos de la GNU LGPL versión 3:

```
$ guix package -s "" | recsel -p name -e 'license ~ "LGPL 3"'
name: elfutils

name: gmp
...
```

También es posible refinar los resultados de búsqueda mediante el uso de varias opciones `-s`, o varios parámetros a `guix search`. Por ejemplo, la siguiente orden devuelve un lista de juegos de mesa[1] (esta vez mediante el uso del alias `guix search`:

```
$ guix search '\<board\>' game | recsel -p name
name: gnubg
...
```

Si omitimos `-s game`, también obtendríamos paquetes de software que tengan que ver con placas de circuitos impresos (`"circuit board"` en inglés); borrar los signos mayor y menor alrededor de `board` añadiría paquetes que tienen que ver con teclados (keyboard en inglés).

Y ahora para un ejemplo más elaborado. La siguiente orden busca bibliotecas criptográficas, descarta bibliotecas Haskell, Perl, Python y Ruby, e imprime el nombre y la sinopsis de los paquetes resultantes:

```
$ guix search crypto library | \
    recsel -e '! (name ~ "^(ghc|perl|python|ruby)")' -p name,synopsis
```

See Section "Selection Expressions" in *GNU recutils manual*, para más información en *expresiones de selección* para `recsel -e`.

`--show=`*paquete*

Muestra los detalles del *paquete*, tomado de la lista disponible de paquetes, en formato `recutils` (see *GNU recutils manual*).

```
$ guix package --show=guile | recsel -p name,version
```

[1] NdT: board en inglés.

```
name: guile
version: 3.0.5

name: guile
version: 3.0.2

name: guile
version: 2.2.7
...
```

También puede especificar el nombre completo de un paquete para únicamente obtener detalles sobre una versión específica (esta vez usando el alias `guix show`):

```
$ guix show guile@3.0.5 | recsel -p name,version
name: guile
version: 3.0.5
```

`--list-installed[=`*regexp*`]`
`-I [`*regexp*`]`

Enumera los paquetes actualmente instalados en el perfil especificado, con los últimos paquetes instalados mostrados al final. Cuando se especifica *regexp*, enumera únicamente los paquetes instalados cuyos nombres son aceptados por *regexp*.

Por cada paquete instalado, imprime los siguientes elementos, separados por tabuladores: el nombre del paquete, la cadena de versión, la parte del paquete que está instalada (por ejemplo, `out` para la salida predeterminada, `include` para sus cabeceras, etc.), y la ruta de este paquete en el almacén.

`--list-available[=`*regexp*`]`
`-A [`*regexp*`]`

Enumera los paquetes disponibles actualmente en la distribución para este sistema (see Section 1.2 [Distribución GNU], page 2). Cuando se especifica *regexp*, enumera únicamente paquetes disponibles cuyo nombre coincide con *regexp*.

Por cada paquete, imprime los siguientes elementos separados por tabuladores: su nombre, su cadena de versión, las partes del paquete (see Section 5.4 [Paquetes con múltiples salidas], page 51) y la dirección de las fuentes de su definición.

`--list-generations[=`*patrón*`]`
`-l [`*patrón*`]`

Devuelve una lista de generaciones junto a sus fechas de creación; para cada generación, muestra los paquetes instalados, con los paquetes instalados más recientemente mostrados los últimos. Fíjese que la generación cero nunca se muestra.

Por cada paquete instalado, imprime los siguientes elementos, separados por tabuladores: el nombre de un paquete, su cadena de versión, la parte del paquete que está instalada (see Section 5.4 [Paquetes con múltiples salidas], page 51), y la ruta de este paquete en el almacén.

Cuando se usa *patrón*, la orden devuelve únicamente las generaciones que se ajustan al patrón. Entre los patrones adecuados se encuentran:

- *Enteros y enteros separados por comas.* Ambos patrones denotan números de generación. Por ejemplo, `--list-generations=1` devuelve la primera.

  Y `--list-generations=1,8,2` devuelve las tres generaciones en el orden especificado. No se permiten ni espacios ni una coma al final.

- *Rangos.* `--list-generations=2..9` imprime las generaciones especificadas y todas las intermedias. Fíjese que el inicio de un rango debe ser menor a su fin.

  También es posible omitir el destino final. Por ejemplo, `--list-generations=2..` devuelve todas las generaciones empezando por la segunda.

- *Duraciones.* Puede también obtener los últimos *N* días, semanas, o meses pasando un entero junto a la primera letra de la duración. Por ejemplo, `--list-generations=20d` enumera las generaciones que tienen hasta 20 días de antigüedad.

`--delete-generations[=`*patrón*`]`
`-d [`*patrón*`]`

Cuando se omite *patrón*, borra todas las generaciones excepto la actual.

Esta orden acepta los mismos patrones que `--list-generations`. Cuando se especifica un *patrón*, borra las generaciones coincidentes. Cuando el *patrón* especifica una duración, las generaciones *más antiguas* que la duración especificada son las borradas. Por ejemplo, `--delete-generations=1m` borra las generaciones de más de un mes de antigüedad.

Si la generación actual entra en el patrón, *no* es borrada. Tampoco la generación cero es borrada nunca.

Preste atención a que el borrado de generaciones previas impide la reversión a su estado. Consecuentemente esta orden debe ser usada con cuidado.

`--export-manifest`

Write to standard output a manifest suitable for `--manifest` corresponding to the chosen profile(s).

This option is meant to help you migrate from the "imperative" operating mode—running `guix install`, `guix upgrade`, etc.—to the declarative mode that `--manifest` offers.

Be aware that the resulting manifest *approximates* what your profile actually contains; for instance, depending on how your profile was created, it can refer to packages or package versions that are not exactly what you specified.

Keep in mind that a manifest is purely symbolic: it only contains package names and possibly versions, and their meaning varies over time. If you wish to "pin" channels to the revisions that were used to build the profile(s), see `--export-channels` below.

`--export-channels`

Write to standard output the list of channels used by the chosen profile(s), in a format suitable for `guix pull --channels` or `guix time-machine --channels` (see Chapter 6 [Canales], page 69).

Together with `--export-manifest`, this option provides information allowing you to replicate the current profile (see Section 6.3 [Replicación de Guix], page 70).

However, note that the output of this command *approximates* what was actually used to build this profile. In particular, a single profile might have been built from several different revisions of the same channel. In that case, `--export-manifest` chooses the last one and writes the list of other revisions in a comment. If you really need to pick packages from different channel revisions, you can use inferiors in your manifest to do so (see Section 5.9 [Inferiores], page 62).

Together with `--export-manifest`, this is a good starting point if you are willing to migrate from the "imperative" model to the fully declarative model consisting of a manifest file along with a channels file pinning the exact channel revision(s) you want.

Finally, since `guix package` may actually start build processes, it supports all the common build options (see Section 9.1.1 [Opciones comunes de construcción], page 181). It also supports package transformation options, such as `--with-source`, and preserves them across upgrades (see Section 9.1.2 [Opciones de transformación de paquetes], page 184).

## 5.3 Sustituciones

Guix permite despliegues transparentes de fuentes/binarios, lo que significa que puede tanto construir cosas localmente, como descargar elementos preconstruidos de un servidor, o ambas. Llamamos a esos elementos preconstruidos *sustituciones*—son sustituciones de los resultados de construcciones locales. En muchos casos, descargar una sustitución es mucho más rápido que construirla localmente.

Las sustituciones pueden ser cualquier cosa que resulte de una construcción de una derivación (see Section 8.10 [Derivaciones], page 159). Por supuesto, en el caso común, son paquetes binarios preconstruidos, pero los archivos de fuentes, por ejemplo, que también resultan de construcciones de derivaciones, pueden estar disponibles como sustituciones.

### 5.3.1 Official Substitute Servers

`bordeaux.guix.gnu.org` and `ci.guix.gnu.org` are both front-ends to official build farms that build packages from Guix continuously for some architectures, and make them available as substitutes. These are the default source of substitutes; which can be overridden by passing the `--substitute-urls` option either to `guix-daemon` (see [`guix-daemon --substitute-urls`], page 13) or to client tools such as `guix package` (see [client `--substitute-urls` option], page 182).

Las URLs de sustituciones pueden ser tanto HTTP como HTTPS. Se recomienda HTTPS porque las comunicaciones están cifradas; de modo contrario, usar HTTP hace visibles todas las comunicaciones para alguien que las intercepte, quien puede usar la información obtenida para determinar, por ejemplo, si su sistema tiene vulnerabilidades de seguridad sin parchear.

Substitutes from the official build farms are enabled by default when using Guix System (see Section 1.2 [Distribución GNU], page 2). However, they are disabled by default when using Guix on a foreign distribution, unless you have explicitly enabled them via one of the recommended installation steps (see Chapter 2 [Instalación], page 4). The following

paragraphs describe how to enable or disable substitutes for the official build farm; the same procedure can also be used to enable substitutes for any other substitute server.

### 5.3.2 Autorización de servidores de sustituciones

To allow Guix to download substitutes from `bordeaux.guix.gnu.org`, `ci.guix.gnu.org` or a mirror, you must add the relevant public key to the access control list (ACL) of archive imports, using the `guix archive` command (see Section 5.11 [Invocación de guix archive], page 66). Doing so implies that you trust the substitute server to not be compromised and to serve genuine substitutes.

> **Nota:** If you are using Guix System, you can skip this section: Guix System authorizes substitutes from `bordeaux.guix.gnu.org` and `ci.guix.gnu.org` by default.

The public keys for each of the project maintained substitute servers are installed along with Guix, in *prefix*`/share/guix/`, where *prefix* is the installation prefix of Guix. If you installed Guix from source, make sure you checked the GPG signature of `guix-c7888f5.tar.gz`, which contains this public key file. Then, you can run something like this:

```
# guix archive --authorize < prefix/share/guix/bordeaux.guix.gnu.org.pub
# guix archive --authorize < prefix/share/guix/ci.guix.gnu.org.pub
```

Una vez esté autorizada, la salida de una orden como `guix build` debería cambiar de algo como:

```
$ guix build emacs --dry-run
Se construirían las siguientes derivaciones:
   /gnu/store/yr7bnx8xwcayd6j95r2clmkdl1qh688w-emacs-24.3.drv
   /gnu/store/x8qsh1hlhgjx6cwsjyvybnfv2i37z23w-dbus-1.6.4.tar.gz.drv
   /gnu/store/1ixwp12fl950d15h2cj11c73733jay0z-alsa-lib-1.0.27.1.tar.bz2.drv
   /gnu/store/nlma1pw0p603fpfiqy7kn4zm105r5dmw-util-linux-2.21.drv
...
```

a algo así:

```
$ guix build emacs --dry-run
Se descargarían 112.3 MB:
   /gnu/store/pk3n22lbq6ydamyymqkkz7i69wiwjiwi-emacs-24.3
   /gnu/store/2ygn4ncnhrpr61rssa6z0d9x22si0va3-libjpeg-8d
   /gnu/store/71yz6lgx4dazma9dwn2mcjxaah9w77jq-cairo-1.12.16
   /gnu/store/7zdhgp0n1518lvfn8mb96sxqfmvqrl7v-libxrender-0.9.7
...
```

The text changed from "The following derivations would be built" to "112.3 MB would be downloaded". This indicates that substitutes from the configured substitute servers are usable and will be downloaded, when possible, for future builds.

El mecanismo de sustituciones puede ser desactivado globalmente ejecutando `guix-daemon` con `--no-subsitutes` (see Section 2.3 [Invocación de guix-daemon], page 12). También puede ser desactivado temporalmente pasando la opción `--no-substitutes` a `guix package`, `guix build` y otras herramientas de línea de órdenes.

### 5.3.3 Obtención de sustiticiones desde otros servidores

Guix puede buscar y obtener sustituciones a partir de varios servidores. Esto es útil cuando se usan paquetes de canales adicionales para los que el servidor oficial no proporciona sustituciones pero otros servidores sí. Otra situación donde puede esta característica puede ser útil es en el caso de que prefiera realizar las descargas desde el servidor de sustituciones de su organización, accediendo al servidor oficial únicamente como mecanismo de salvaguarda o no usándolo en absoluto.

Puede proporcionarle a Guix una lista de URL de servidores de los que obtener sustituciones y las comprobará en el orden especificado. También es necesario que autorice explícitamente las claves públicas de los servidores de sustituciones para que Guix acepte las sustituciones firmadas por dichos claves.

En el sistema Guix esto se consigue modificando la configuración del servicio `guix`. Puesto que el servicio `guix` es parte de las listas de servicios predeterminadas, `%base-services` y `%desktop-services`, puede usar `modify-services` para cambiar su configuración y añadir las URL y claves para sustituciones que desee (see Section 11.19.3 [Referencia de servicios], page 652).

As an example, suppose you want to fetch substitutes from `guix.example.org` and to authorize the signing key of that server, in addition to the default `bordeaux.guix.gnu.org` and `ci.guix.gnu.org`. The resulting operating system configuration will look something like:

```
(operating-system
  ;; ...
  (services
    ;; Se asume que se parte de '%desktop-services'. Debe sustituirse
    ;; por la lista de servicios que use en realidad.
    (modify-services %desktop-services
      (guix-service-type config =>
                         (guix-configuration
                           (inherit config)
                           (substitute-urls
                             (append (list "https://guix.example.org")
                                     %default-substitute-urls))
                           (authorized-keys
                             (append (list (local-file "./clave.pub"))
                                     %default-authorized-guix-keys)))))))
```

Esto asume que el archivo `clave.pub` contiene la clave de firma de `guix.example.org`. Cuando haya realizado este cambio en el archivo de configuración de su sistema operativo (digamos `/etc/config.scm`), puede reconfigurar y reiniciar el servicio `guix-daemon` o reiniciar la máquina para que los cambios se hagan efectivos:

```
$ sudo guix system reconfigure /etc/config.scm
$ sudo herd restart guix-daemon
```

Si por el contrario ejecuta Guix sobre una distribución distinta, deberá llevar a cabo los siguientes pasos para obtener sustituciones de servidores adicionales:

1. Edite el archivo de configuración para el servicio de `guix-daemon`; cuando use systemd normalmente se trata de `/etc/systemd/system/guix-daemon.service`. Añada

la opción `--substitute-urls` en la línea de ordenes de `guix-daemon` y la lista de URL que desee (see [daemon-substitute-urls], page 13):

```
... --substitute-urls='https://guix.example.org https://bordeaux.guix.gnu.org
https://ci.guix.gnu.org'
```

2. Reinicie el daemon. Con systemd estos son los pasos:

```
systemctl daemon-reload
systemctl restart guix-daemon.service
```

3. Autorice la clave del nuevo servidor (see Section 5.11 [Invocación de guix archive], page 66):

```
guix archive --authorize < clave.pub
```

   De nuevo se asume que `clave.pub` contiene la clave pública usada por `guix.example.org` para firmar las sustituciones.

Now you're all set! Substitutes will be preferably taken from `https://guix.example.org`, using `bordeaux.guix.gnu.org` then `ci.guix.gnu.org` as fallback options. Of course you can list as many substitute servers as you like, with the caveat that substitute lookup can be slowed down if too many servers need to be contacted.

**Troubleshooting:** To diagnose problems, you can run `guix weather`. For example, running:

```
guix weather coreutils
```

not only tells you which of the currently-configured servers has substitutes for the `coreutils` package, it also reports whether one of these servers is unauthorized. See Section 9.15 [Invocación de guix weather], page 234, for more information.

Tenga en cuenta que existen situaciones en las que se puede desear añadir la URL de un servidor de sustitucines *sin* autorizar su clave. See Section 5.3.4 [Verificación de sustituciones], page 49, para entender este caso específico.

### 5.3.4 Verificación de sustituciones

Guix detecta y emite errores cuando se intenta usar una sustitución que ha sido adulterado. Del mismo modo, ignora las sustituciones que no están firmadas, o que no están firmadas por una de las firmas enumeradas en la ACL.

No obstante hay una excepción: si un servidor no autorizado proporciona sustituciones que son *idénticas bit-a-bit* a aquellas proporcionadas por un servidor autorizado, entonces el servidor no autorizado puede ser usado para descargas. Por ejemplo, asumiendo que hemos seleccionado dos servidores de sustituciones con esta opción:

```
--substitute-urls="https://a.example.org https://b.example.org"
```

Si la ACL contiene únicamente la clave para '`b.example.org`', y si '`a.example.org`' resulta que proporciona *exactamente las mismas* sustituciones, Guix descargará sustituciones de '`a.example.org`' porque viene primero en la lista y puede ser considerado un espejo de '`b.example.org`'. En la práctica, máquinas de construcción independientes producen habitualmente los mismos binarios, gracias a las construcciones reproducibles bit-a-bit (véase a continuación).

Cuando se usa HTTPS, el certificado X.509 del servidor *no* se valida (en otras palabras, el servidor no está verificado), lo contrario del comportamiento habitual de los navegadores Web. Esto es debido a que Guix verifica la información misma de las sustituciones, como se ha explicado anteriormente, lo cual nos concierne (mientras que los certificados X.509 tratan de verificar las conexiones entre nombres de dominio y claves públicas).

### 5.3.5 Configuración de la pasarela.

Substitutes are downloaded over HTTP or HTTPS. The `http_proxy` and `https_proxy` environment variables can be set in the environment of `guix-daemon` and are honored for downloads of substitutes. Note that the value of those environment variables in the environment where `guix build`, `guix package`, and other client commands are run has *absolutely no effect*.

### 5.3.6 Fallos en las sustituciones

Incluso cuando una sustitución de una derivación está disponible, a veces el intento de sustitución puede fallar. Esto puede suceder por varias razones: el servidor de sustituciones puede estar desconectado, la sustitución puede haber sido borrada, la conexión puede interrumpirse, etc.

Cuando las sustituciones están activadas y una sustitución para una derivación está disponible, pero el intento de sustitución falla, Guix intentará construir la derivación localmente dependiendo si se proporcionó la opción `--fallback` (see [opción común de construcción `--fallback`], page 182). Específicamente, si no se pasó `--fallback`, no se realizarán construcciones locales, y la derivación se considera se considera fallida. No obstante, si se pasó `--fallback`, Guix intentará construir la derivación localmente, y el éxito o fracaso de la derivación depende del éxito o fracaso de la construcción local. Fíjese que cuando las sustituciones están desactivadas o no hay sustituciones disponibles para la derivación en cuestión, la construcción local se realizará *siempre*, independientemente de si se pasó la opción `--fallback`.

Para hacerse una idea de cuantas sustituciones hay disponibles en este momento, puede intentar ejecutar la orden `guix weather` (see Section 9.15 [Invocación de guix weather], page 234). Esta orden proporciona estadísticas de las sustituciones proporcionadas por un servidor.

### 5.3.7 Sobre la confianza en binarios

Today, each individual's control over their own computing is at the mercy of institutions, corporations, and groups with enough power and determination to subvert the computing infrastructure and exploit its weaknesses. While using substitutes can be convenient, we encourage users to also build on their own, or even run their own build farm, such that the project run substitute servers are less of an interesting target. One way to help is by publishing the software you build using `guix publish` so that others have one more choice of server to download substitutes from (see Section 9.11 [Invocación de guix publish], page 226).

Guix tiene los cimientos para maximizar la reproducibilidad de las construcciones (see Section 5.1 [Características], page 35). En la mayor parte de los casos, construcciones independientes de un paquete o derivación dada deben emitir resultados idénticos bit a bit. Por tanto, a través de un conjunto diverso de construcciones independientes de paquetes,

podemos reforzar la integridad de nuestros sistemas. La orden `guix challenge` intenta ayudar a las usuarias en comprobar servidores de sustituciones, y asiste a las desarrolladoras encontrando construcciones no deterministas de paquetes (see Section 9.12 [Invocación de guix challenge], page 230). Similarmente, la opción `--check` de `guix build` permite a las usuarias si las sustituciones previamente instaladas son genuinas mediante su reconstrucción local (see [build-check], page 193).

En el futuro, queremos que Guix permita la publicación y obtención de binarios hacia/desde otras usuarias, entre pares (P2P). En caso de interesarle hablar sobre este proyecto, unase a nosotras en `guix-devel@gnu.org`.

## 5.4 Paquetes con múltiples salidas

Habitualmente, los paquetes definidos en Guix tienen una *salida* única—es decir, el paquete de fuentes proporcionará exactamente un directorio en el almacén. Cuando se ejecuta `guix install glibc`, se instala la salida predeterminada del paquete GNU libc; la salida predeterminada se llama `out`, pero su nombre puede omitirse como se mostró en esta orden. En este caso particular, la salida predeterminada de `glibc` contiene todos archivos de cabecera C, bibliotecas dinámicas, bibliotecas estáticas, documentación Info y otros archivos auxiliares.

A veces es más apropiado separar varios tipos de archivos producidos por un paquete único de fuentes en salidas separadas. Por ejemplo, la biblioteca C GLib (usada por GTK+ y paquetes relacionados) instala más de 20 MiB de documentación de referencia como páginas HTML. Para ahorrar espacio para usuarias que no la necesiten, la documentación va a una salida separada, llamada `doc`. Para instalar la salida principal de GLib, que contiene todo menos la documentación, se debe ejecutar:

```
guix install glib
```

La orden que instala su documentación es:

```
guix install glib:doc
```

While the colon syntax works for command-line specification of package outputs, it will not work when using a package *variable* in Scheme code. For example, to add the documentation of `glib` to the globally installed packages of an `operating-system` (see Section 11.3 [Referencia de operating-system], page 253), a list of two items, the first one being the package *variable* and the second one the name of the output to select (a string), must be used instead:

```
(use-modules (gnu packages glib))
;; glib-with-documentation is the Guile symbol for the glib package
(operating-system
 ...
 (packages
  (append
   (list (list glib-with-documentation "doc"))
         %base-packages)))
```

Algunos paquetes instalan programas con diferentes “huellas de dependencias”. Por ejemplo, el paquete WordNet instala tanto herramientas de línea de órdenes como interfaces gráficas de usuaria (IGU). Las primeras dependen únicamente de la biblioteca de C, mientras

que las últimas dependen en Tcl/Tk y las bibliotecas de X subyacentes. En este caso, dejamos las herramientas de línea de órdenes en la salida predeterminada, mientras que las IGU están en una salida separada. Esto permite a las usuarias que no necesitan una IGU ahorrar espacio. La orden `guix size` puede ayudar a exponer estas situaciones (see Section 9.9 [Invocación de guix size], page 219). `guix graph` también puede ser útil (see Section 9.10 [Invocación de guix graph], page 221).

Hay varios de estos paquetes con salida múltiple en la distribución GNU. Otros nombres de salida convencionales incluyen `lib` para bibliotecas y posiblemente archivos de cabecera, `bin` para programas independientes y `debug` para información de depuración (see Chapter 17 [Instalación de archivos de depuración], page 718). La salida de los paquetes se enumera en la tercera columna del resultado de `guix package --list-available` (see Section 5.2 [Invocación de guix package], page 36).

## 5.5 Invoking `guix locate`

There's so much free software out there that sooner or later, you will need to search for packages. The `guix search` command that we've seen before (see Section 5.2 [Invocación de guix package], page 36) lets you search by keywords:

```
guix search video editor
```

Sometimes, you instead want to find which package provides a given file, and this is where `guix locate` comes in. Here is how you can find which package provides the `ls` command:

```
$ guix locate ls
coreutils@9.1       /gnu/store/...-coreutils-9.1/bin/ls
```

Of course the command works for any file, not just commands:

```
$ guix locate unistr.h
icu4c@71.1          /gnu/store/.../include/unicode/unistr.h
libunistring@1.0    /gnu/store/.../include/unistr.h
```

You may also specify *glob patterns* with wildcards. For example, here is how you would search for packages providing `.service` files:

```
$ guix locate -g '*.service'
man-db@2.11.1        .../lib/systemd/system/man-db.service
wpa-supplicant@2.10  .../system-services/fi.w1.wpa_supplicant1.service
```

The `guix locate` command relies on a database that maps file names to package names. By default, it automatically creates that database if it does not exist yet by traversing packages available *locally*, which can take a few minutes (depending on the size of your store and the speed of your storage device).

> **Nota:** For now, `guix locate` builds its database based on purely local knowledge—meaning that you will not find packages that never reached your store. Eventually it will support downloading a pre-built database so you can potentially find more packages.

By default, `guix locate` first tries to look for a system-wide database, usually under `/var/cache/guix/locate`; if it does not exist or is too old, it falls back to the per-user database, by default under `~/.cache/guix/locate`. On a multi-user system, administrators may want to periodically update the system-wide database so that all users can benefit from

it, for instance by setting up `package-database-service-type` (see Section 11.10.11 [File Search Services], page 384).

La sintaxis general es:

```
guix locate [options...] file...
```

... where *file* is the name of a file to search for (specifically, the "base name" of the file: files whose parent directories are called *file* are not matched).

Las opciones disponibles son las siguientes:

`-gglob` Interpret *file*... as *glob patterns*—patterns that may include wildcards, such as '`*.scm`' to denote all files ending in '`.scm`'.

`--stats` Display database statistics.

`--update`
`-u` Update the file database.

By default, the database is automatically updated when it is too old.

`--clear` Clear the database and re-populate it.

This option lets you start anew, ensuring old data is removed from the database, which also avoids having an endlessly growing database. By default `guix locate` automatically does that periodically, though infrequently.

`--database=`*file*
Use *file* as the database, creating it if necessary.

By default, `guix locate` picks the database under `~/.cache/guix` or `/var/cache/guix`, whichever is the most recent one.

`--method=`*method*
`-m` *method* Use *method* to select the set of packages to index. Possible values are:

`manifests`
This is the default method: it works by traversing profiles on the machine and recording packages it encounters—packages you or other users of the machine installed, directly or indirectly. It is fast but it can miss other packages available in the store but not referred to by any profile.

`almacén` This is a slower but more exhaustive method: it checks among all the existing packages those that are available in the store and records them.

## 5.6 Invocación de `guix gc`

Los paquetes instalados, pero no usados, pueden ser *recolectados*. La orden `guix gc` permite a las usuarias ejecutar explícitamente el recolector de basura para reclamar espacio del directorio `/gnu/store`—¡borrar archivos o directorios manualmente puede dañar el almacén sin reparación posible!

El recolector de basura tiene un conjunto de *raíces* conocidas: cualquier archivo en `/gnu/store` alcanzable desde una raíz se considera *vivo* y no puede ser borrado; cualquier otro archivo se considera *muerto* y puede ser borrado. El conjunto de raíces del recolector de

basura ("raíces del GC" para abreviar) incluye los perfiles predeterminados de las usuarias; por defecto los enlaces bajo `/var/guix/gcroots` representan dichas raíces. Por ejemplo, nuevas raíces del GC pueden añadirse con `guix build --root` (see Section 9.1 [Invocación de guix build], page 181). La orden `guix gc --list-roots` las enumera.

Antes de ejecutar `guix gc --collect-garbage` para liberar espacio, habitualmente es útil borrar generaciones antiguas de los perfiles de usuaria; de ese modo, las construcciones antiguas de paquetes a las que dichas generaciones hacen referencia puedan ser reclamadas. Esto se consigue ejecutando `guix package --delete-generations` (see Section 5.2 [Invocación de guix package], page 36).

Nuestra recomendación es ejecutar una recolección de basura periódicamente, o cuando tenga poco espacio en el disco. Por ejemplo, para garantizar que al menos 5 GB están disponibles en su disco, simplemente ejecute:

```
guix gc -F 5G
```

Es completamente seguro ejecutarla como un trabajo periódico no-interactivo (see Section 11.10.2 [Ejecución de tareas programadas], page 297, para la configuración de un trabajo de ese tipo). La ejecución de `guix gc` sin ningún parámetro recolectará tanta basura como se pueda, pero eso es no es normalmente conveniente: puede encontrarse teniendo que reconstruir o volviendo a bajar software que está "muerto" desde el punto de vista del recolector pero que es necesario para construir otras piezas de software—por ejemplo, la cadena de herramientas de compilación.

La orden `guix gc` tiene tres modos de operación: puede ser usada para recolectar archivos muertos (predeterminado), para borrar archivos específicos (la opción `--delete`), para mostrar información sobre la recolección de basura o para consultas más avanzadas. Las opciones de recolección de basura son las siguientes:

`--collect-garbage[=`*min*`]`
`-C [`*min*`]`
: Recolecta basura—es decir, archivos no alcanzables de `/gnu/store` y subdirectorios. Esta operación es la predeterminada cuando no se especifican opciones.

  Cuando se proporciona *min*, para una vez que *min* bytes han sido recolectados. *min* puede ser un número de bytes, o puede incluir una unidad como sufijo, como `MiB` para mebibytes y `GB` para gigabytes (see Section "Block size" in *GNU Coreutils*).

  Cuando se omite *min*, recolecta toda la basura.

`--free-space=`*libre*
`-F` *libre*
: Recolecta basura hasta que haya espacio *libre* bajo `/gnu/store`, si es posible: *libre* denota espacio de almacenamiento, por ejemplo `500MiB`, como se ha descrito previamente.

  Cuando *libre* o más está ya disponible en `/gnu/store`, no hace nada y sale inmediatamente.

`--delete-generations[=`*duración*`]`
`-d [`*duración*`]`
: Before starting the garbage collection process, delete all the generations older than *duration*, for all the user profiles and home environment generations; when run as root, this applies to all the profiles *of all the users*.

Por ejemplo, esta orden borra todas las generaciones de todos sus perfiles que tengan más de 2 meses de antigüedad (excepto generaciones que sean las actuales), y una vez hecho procede a liberar espacio hasta que al menos 10 GiB estén disponibles:

```
guix gc -d 2m -F 10G
```

`--delete`
`-D`
: Intenta borrar todos los archivos del almacén y directorios especificados como parámetros. Esto falla si alguno de los archivos no están en el almacén, o todavía están vivos.

`--list-failures`
: Enumera los elementos del almacén correspondientes a construcciones fallidas existentes en la caché.

  Esto no muestra nada a menos que el daemon se haya ejecutado pasando `--cache-failures` (see Section 2.3 [Invocación de guix-daemon], page 12).

`--list-roots`
: Enumera las raíces del recolector de basura poseídas por la usuaria; cuando se ejecuta como root, enumera *todas* las raíces del recolector de basura.

`--list-busy`
: Enumera los elementos del almacén que actualmente están siendo usados por procesos en ejecución. Estos elementos del almacén se consideran de manera efectiva raíces del recolector de basura: no pueden borrarse.

`--clear-failures`
: Borra los elementos especificados del almacén de la caché de construcciones fallidas.

  De nuevo, esta opción únicamente tiene sentido cuando el daemon se inicia con `--cache-failures`. De otro modo, no hace nada.

`--list-dead`
: Muestra la lista de archivos y directorios muertos todavía presentes en el almacén—es decir, archivos y directorios que ya no se pueden alcanzar desde ninguna raíz.

`--list-live`
: Muestra la lista de archivos y directorios del almacén vivos.

Además, las referencias entre los archivos del almacén pueden ser consultadas:

`--references`
`--referrers`
: Enumera las referencias (o, respectivamente, los referentes) de los archivos del almacén pasados como parámetros.

`--requisites`
`-R`
: Enumera los requisitos los archivos del almacén pasados como parámetros. Los requisitos incluyen los mismos archivos del almacén, sus referencias, las referencias de estas, recursivamente. En otras palabras, la lista devuelta es la *clausura transitiva* de los archivos del almacén.

See Section 9.9 [Invocación de guix size], page 219, para una herramienta que perfila el tamaño de la clausura de un elemento. See Section 9.10 [Invocación de guix graph], page 221, para una herramienta de visualización del grafo de referencias.

`--derivers`
: Devuelve la/s derivación/es que conducen a los elementos del almacén dados (see Section 8.10 [Derivaciones], page 159).

  Por ejemplo, esta orden:

  ```
  guix gc --derivers $(guix package -I ^emacs$ | cut -f4)
  ```

  devuelve el/los archivo/s `.drv` que conducen al paquete `emacs` instalado en su perfil.

  Fíjese que puede haber cero archivos `.drv` encontrados, por ejemplo porque estos archivos han sido recolectados. Puede haber más de un archivo `.drv` encontrado debido a derivaciones de salida fija.

Por último, las siguientes opciones le permiten comprobar la integridad del almacén y controlar el uso del disco.

`--verify[=`*opciones*`]`
: Verifica la integridad del almacén.

  Por defecto, comprueba que todos los elementos del almacén marcados como válidos en la base de datos del daemon realmente existen en `/gnu/store`.

  Cuando se proporcionan, *opciones* debe ser una lista separada por comas que contenga uno o más valores `contents` and `repair`.

  Cuando se usa `--verify=contents`, el daemon calcula el hash del contenido de cada elemento del almacén y lo compara contra el hash de su base de datos. Las incongruencias se muestran como corrupciones de datos. Debido a que recorre *todos los archivos del almacén*, esta orden puede tomar mucho tiempo, especialmente en sistemas con una unidad de disco lenta.

  El uso de `--verify=repair` o `--verify=contents,repair` hace que el daemon intente reparar elementos corruptos del almacén obteniendo sustituciones para dichos elementos (see Section 5.3 [Sustituciones], page 46). Debido a que la reparación no es atómica, y por tanto potencialmente peligrosa, está disponible únicamente a la administradora del sistema. Una alternativa ligera, cuando sabe exactamente qué elementos del almacén están corruptos, es `guix build --repair` (see Section 9.1 [Invocación de guix build], page 181).

`--optimize`
: Optimiza el almacén sustituyendo archivos idénticos por enlaces duros—esto es la *deduplicación*.

  El daemon realiza la deduplicación después de cada construcción satisfactoria o importación de archivos, a menos que se haya lanzado con la opción `--disable-deduplication` (see Section 2.3 [Invocación de guix-daemon], page 12). Por tanto, esta opción es útil principalmente cuando el daemon se estaba ejecutando con `--disable-deduplication`.

`--vacuum-database`

Guix uses an sqlite database to keep track of the items in (see Section 8.9 [El almacén], page 157). Over time it is possible that the database may grow to a large size and become fragmented. As a result, one may wish to clear the freed space and join the partially used pages in the database left behind from removed packages or after running the garbage collector. Running `sudo guix gc --vacuum-database` will lock the database and `VACUUM` the store, defragmenting the database and purging freed pages, unlocking the database when it finishes.

## 5.7 Invocación de `guix pull`

Packages are installed or upgraded to the latest version available in the distribution currently available on your local machine. To update that distribution, along with the Guix tools, you must run `guix pull`: the command downloads the latest Guix source code and package descriptions, and deploys it. Source code is downloaded from a Git (`https://git-scm.com/book/en/`) repository, by default the official GNU Guix repository, though this can be customized. `guix pull` ensures that the code it downloads is *authentic* by verifying that commits are signed by Guix developers.

Specifically, `guix pull` downloads code from the *channels* (see Chapter 6 [Canales], page 69) specified by one of the following, in this order:

1. la opción `--channels`;
2. the user's `~/.config/guix/channels.scm` file, unless `-q` is passed;
3. the system-wide `/etc/guix/channels.scm` file, unless `-q` is passed (on Guix System, this file can be declared in the operating system configuration, see [guix-configuration-channels], page 286);
4. los canales predeterminados en código especificados en la variable `%default-channels`.

Una vez completada, `guix package` usará paquetes y versiones de paquetes de esta copia recién obtenida de Guix. No solo eso, sino que todas las órdenes de Guix y los módulos Scheme también se tomarán de la última versión. Nuevas sub-órdenes `guix` incorporadas por la actualización también estarán disponibles.

Cualquier usuaria puede actualizar su copia de Guix usando `guix pull`, y el efecto está limitado a la usuaria que ejecute `guix pull`. Por ejemplo, cuando la usuaria `root` ejecuta `guix pull`, dicha acción no produce ningún efecto en la versión del Guix que la usuaria `alicia` ve, y viceversa.

El resultado de ejecutar `guix pull` es un *perfil* disponible bajo `~/.config/guix/current` conteniendo el último Guix. Por tanto, asegúrese de añadirlo al inicio de sus rutas de búsqueda de modo que use la última versión, de modo similar para el manual Info(see Chapter 14 [Documentación], page 707).

```
export PATH="$HOME/.config/guix/current/bin:$PATH"
export INFOPATH="$HOME/.config/guix/current/share/info:$INFOPATH"
```

Las opciones `--list-generations` o `-l` enumeran las generaciones pasadas producidas por `guix pull`, junto a detalles de su procedencia:

```
$ guix pull -l
Generación 1 10 jun 2018 00:18:18
```

```
  guix 65956ad
    URL del repositorio: https://git.savannah.gnu.org/git/guix.git
    rama: origin/master
    revisión: 65956ad3526ba09e1f7a40722c96c6ef7c0936fe

Generation 2 Jun 11 2018 11:02:49
  guix e0cc7f6
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: origin/master
    commit: e0cc7f669bec22c37481dd03a7941c7d11a64f1d

Generation 3 Jun 13 2018 23:31:07 (current)
  guix 844cc1c
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: origin/master
    commit: 844cc1c8f394f03b404c5bb3aee086922373490c
```

Section 5.10 [Invocación de guix describe], page 64, para otras formas de describir el estado actual de Guix.

This `~/.config/guix/current` profile works exactly like the profiles created by `guix package` (see Section 5.2 [Invocación de guix package], page 36). That is, you can list generations, roll back to the previous generation—i.e., the previous Guix—and so on:

```
$ guix pull --roll-back
se pasó de la generación 3 a la 2
$ guix pull --delete-generations=1
borrando /var/guix/profiles/per-user/carlos/current-guix-1-link
```

También puede usar `guix package` (see Section 5.2 [Invocación de guix package], page 36) para gestionar el perfil proporcionando su nombre de manera específica:

```
$ guix package -p ~/.config/guix/current --roll-back
se pasó de la generación 3 a la 2
$ guix package -p ~/.config/guix/current --delete-generations=1
borrando /var/guix/profiles/per-user/carlos/current-guix-1-link
```

La orden `guix pull` se invoca habitualmente sin parámetros, pero permite las siguientes opciones:

`--url=`*url*
`--commit=`*revisión*
`--branch=`*rama*

> Download code for the `guix` channel from the specified *url*, at the given *commit* (a valid Git commit ID represented as a hexadecimal string or the name of a tag), or *branch*.
>
> Estas opciones se proporcionan por conveniencia, pero también puede especificar su configuración en el archivo `~/.config/guix/channels.scm` o usando la opción `--channels` (vea más adelante).

`--channels=`*archivo*
`-C` *archivo*
: Lee la lista de canales de *archivo* en vez de `~/.config/guix/channels.scm` o `/etc/guix/channels.scm`. *archivo* debe contener código Scheme que evalúe a una lista de objetos "channel". See Chapter 6 [Canales], page 69, para más información.

`--no-channel-files`
`-q`
: Inhibit loading of the user and system channel files, `~/.config/guix/channels.scm` and `/etc/guix/channels.scm`.

`--news`
`-N`
: Display news written by channel authors for their users for changes made since the previous generation (see Chapter 6 [Canales], page 69). When `--details` is passed, additionally display new and upgraded packages.

  You can view that information for previous generations with `guix pull -l`.

`--list-generations[=`*patrón*`]`
`-l [`*patrón*`]`
: Enumera todas las generaciones de `~/.config/guix/current` o, si se proporciona un *patrón*, el subconjunto de generaciones que correspondan con el *patrón*. La sintaxis de *patrón* es la misma que `guix package --list-generations` (see Section 5.2 [Invocación de guix package], page 36).

  By default, this prints information about the channels used in each revision as well as the corresponding news entries. If you pass `--details`, it will also print the list of packages added and upgraded in each generation compared to the previous one.

`--details`
: Instruct `--list-generations` or `--news` to display more information about the differences between subsequent generations—see above.

`--roll-back`
: Vuelve a la *generación* previa de `~/.config/guix/current`—es decir, deshace la última transacción.

`--switch-generation=`*patrón*
`-S` *patrón*
: Cambia a una generación particular definida por el *patrón*.

  *patrón* puede ser tanto un número de generación como un número prefijado con "+" o "-". Esto último significa: mueve atrás/hacia delante el número especificado de generaciones. Por ejemplo, si quiere volver a la última generación antes de `--roll-back`, use `--switch-generation=+1`.

`--delete-generations[=`*patrón*`]`
`-d [`*patrón*`]`
: Cuando se omite *patrón*, borra todas las generaciones excepto la actual.

  Esta orden acepta los mismos patrones que `--list-generations`. Cuando se especifica un *patrón*, borra las generaciones coincidentes. Cuando el *patrón* especifica una duración, las generaciones *más antiguas* que la duración especificada son las borradas. Por ejemplo, `--delete-generations=1m` borra las generaciones de más de un mes de antigüedad.

Si la generación actual entra en el patrón, *no* será borrada.

Preste atención a que el borrado de generaciones previas impide la reversión a su estado. Consecuentemente esta orden debe ser usada con cuidado.

Section 5.10 [Invocación de guix describe], page 64, para una forma de mostrar información sobre únicamente la generación actual.

`--profile=`*perfil*
`-p` *perfil* Usa *perfil* en vez de `~/.config/guix/current`.

`--dry-run`
`-n` Muestra qué revisión/es del canal serían usadas y qué se construiría o sustituiría, sin efectuar ninguna acción real.

`--allow-downgrades`
Permite obtener revisiones de los canales más antiguas o no relacionadas con aquellas que se encuentran en uso actualmente.

De manera predeterminada `guix pull` proteje contra los llamados "ataques de versión anterior" en los que el repositorio Git de un canal se reinicia a una revisión anterior o no relacionada de sí mismo, provocando potencialmente la instalación de versiones más antiguas y con vulnerabilidades conocidas de paquetes de software.

> **Nota:** Asegúrese de entender las implicaciones de seguridad antes de usar la opción `--allow-downgrades`.

`--disable-authentication`
Permite obtener código de un canal sin verificarlo.

De manera predeterminada, `guix pull` valida el código que descarga de los canales verificando que sus revisiones están firmadas por desarrolladoras autorizadas, y emite un error si no es el caso. Esta opción le indica que no debe realizar ninguna de esas verificaciones.

> **Nota:** Asegúrese de entender las implicaciones de seguridad antes de usar la opción `--disable-authentication`.

`--system=`*sistema*
`-s` *sistema*
Intenta construir paquetes para *sistema*—por ejemplo, `x86_64-linux`—en vez del tipo de sistema de la máquina de construcción.

`--bootstrap`
Use el Guile usado para el lanzamiento para construir el último Guix. Esta opción es útil para las desarrolladoras de Guix únicamente.

El mecanismo de *canales* le permite instruir a `guix pull` de qué repositorio y rama obtener los datos, así como repositorios *adicionales* que contengan módulos de paquetes que deben ser desplegados. See Chapter 6 [Canales], page 69, para más información.

Además, `guix pull` acepta todas las opciones de construcción comunes (see Section 9.1.1 [Opciones comunes de construcción], page 181).

## 5.8 Invocación de `guix time-machine`

La orden `guix time-machine` proporciona acceso a otras revisiones de Guix, por ejemplo para instalar versiones antiguas de un paquete, o para reproducir una computación en un entorno idéntico. La revisión de Guix que se usará se define por el identificador de una revisión o por un archivo de descripción de canales creado con `guix describe` (see Section 5.10 [Invocación de guix describe], page 64).

Let's assume that you want to travel to those days of November 2020 when version 1.2.0 of Guix was released and, once you're there, run the `guile` of that time:

```
guix time-machine --commit=v1.2.0 -- \
  environment -C --ad-hoc guile -- guile
```

The command above fetches Guix 1.2.0 (and possibly other channels specified by your `channels.scm` configuration files—see below) and runs its `guix environment` command to spawn an environment in a container running `guile` (`guix environment` has since been subsumed by `guix shell`; see Section 7.1 [Invoking guix shell], page 79). It's like driving a DeLorean[2]! The first `guix time-machine` invocation can be expensive: it may have to download or even build a large number of packages; the result is cached though and subsequent commands targeting the same commit are almost instantaneous.

As for `guix pull`, in the absence of any options, `time-machine` fetches the latest commits of the channels specified in `~/.config/guix/channels.scm`, `/etc/guix/channels.scm`, or the default channels; the `-q` option lets you ignore these configuration files. The command:

```
guix time-machine -q -- build hello
```

will thus build the package `hello` as defined in the main branch of Guix, without any additional channel, which is in general a newer revision of Guix than you have installed. Time travel works in both directions!

> **Nota:** The history of Guix is immutable and `guix time-machine` provides the exact same software as they are in a specific Guix revision. Naturally, no security fixes are provided for old versions of Guix or its channels. A careless use of `guix time-machine` opens the door to security vulnerabilities. See Section 5.7 [Invocación de guix pull], page 57.

`guix time-machine` raises an error when attempting to travel to commits older than "v0.16.0" (commit '`4a0b87f0`'), dated Dec. 2018. This is one of the oldest commits supporting the channel mechanism that makes "time travel" possible.

> **Nota:** Although it should technically be possible to travel to such an old commit, the ease to do so will largely depend on the availability of binary substitutes. When traveling to a distant past, some packages may not easily build from source anymore. One such example are old versions of OpenSSL whose tests would fail after a certain date. This particular problem can be worked around by running a *virtual build machine* with its clock set to the right time (see [build-vm], page 548).

La sintaxis general es:

```
guix time-machine opciones... -- orden param...
```

[2] If you don't know what a DeLorean is, consider traveling back to the 1980's. (Back to the Future (1985) (`https://www.imdb.com/title/tt0088763/`))

donde *orden* and *param...* se proporcionan sin modificar a la orden `guix` de la revisión especificada. Las *opciones* que definen esta revisión son las mismas que se usan con `guix pull` (see Section 5.7 [Invocación de guix pull], page 57):

`--url=`*url*
`--commit=`*revisión*
`--branch=`*rama*
: Use the `guix` channel from the specified *url*, at the given *commit* (a valid Git commit ID represented as a hexadecimal string or the name of a tag), or *branch*.

`--channels=`*archivo*
`-C` *archivo*
: Lee la lista de canales de *archivo*. *archivo* debe contener código Scheme que evalúe a una lista de objetos "channel". See Chapter 6 [Canales], page 69, para más información.

`--no-channel-files`
`-q`
: Inhibit loading of the user and system channel files, `~/.config/guix/channels.scm` and `/etc/guix/channels.scm`.

  Thus, `guix time-machine -q` is equivalent to the following Bash command, using the "process substitution" syntax (see Section "Process Substitution" in *The GNU Bash Reference Manual*):

  ```
  guix time-machine -C <(echo %default-channels) ...
  ```

Tenga en cuenta que `guix time-machine` puede desencadenar construcciones de canales y sus dependencias, y que pueden controlarse mediante las opciones de construcción estándar (see Section 9.1.1 [Opciones comunes de construcción], page 181).

## 5.9 Inferiores

> **Nota:** La funcionalidad descrita aquí es una "versión de evaluación tecnológica" en la versión c7888f5. Como tal, la interfaz está sujeta a cambios.

A veces necesita mezclar paquetes de revisiones de la revisión de Guix que está ejecutando actualmente con paquetes disponibles en una revisión diferente. Los *inferiores* de Guix le permiten conseguirlo componiendo diferentes revisiones de Guix de modo arbitrario.

Técnicamente, un "inferior" es esencialmente un proceso Guix separado conectado con su Guix principal a través de una sesión interactiva (see Section 8.13 [Invocación de guix repl], page 177). El módulo `(guix inferior)` le permite crear inferiores y comunicarse con ellos. También proporciona una interfaz de alto nivel para buscar y manipular los paquetes que un inferior proporciona—*paquetes de inferiores*.

When combined with channels (see Chapter 6 [Canales], page 69), inferiors provide a simple way to interact with a separate revision of Guix. For example, let's assume you want to install in your profile the current `guile` package, along with the `guile-json` as it existed in an older revision of Guix—perhaps because the newer `guile-json` has an incompatible API and you want to run your code against the old API. To do that, you could write a manifest for use by `guix package --manifest` (see Section 8.4 [Writing Manifests], page 119); in that manifest, you would create an inferior for that old Guix revision you care about, and you would look up the `guile-json` package in the inferior:

```
(use-modules (guix inferior) (guix channels)
```

```
                (srfi srfi-1))   ;para 'first'

    (define channels
      ;; Esta es la revisión antigua de donde queremos
      ;; extraer guile-json.
      (list (channel
             (name 'guix)
             (url "https://git.savannah.gnu.org/git/guix.git")
             (commit
              "65956ad3526ba09e1f7a40722c96c6ef7c0936fe"))))

    (define inferior
      ;; Un inferior que representa la revisión previa.
      (inferior-for-channels channels))

    ;; Ahora crea un manifiesto con el paquete "guile" actual
    ;; y el antiguo paquete "guile-json".
    (packages->manifest
     (list (first (lookup-inferior-packages inferior "guile-json"))
           (specification->package "guile")))
```

En su primera ejecución, `guix package --manifest` puede tener que construir el canal que especificó antes de crear el inferior; las siguientes ejecuciones serán mucho más rápidas porque la revisión de Guix estará en la caché.

El módulo `(guix inferior)` proporciona los siguientes procedimientos para abrir un inferior:

`inferior-for-channels` *canales* [*#:cache-directory*] [*#:ttl*] [Procedimiento]
: Devuelve un inferior para *canales*, una lista de canales. Usa la caché en *cache-directory*, donde las entradas pueden ser reclamadas después de *ttl* segundos. Este procedimiento abre una nueva conexión al daemon de construcción.

  Como efecto secundario, este procedimiento puede construir o sustituir binarios para *canales*, lo cual puede tomar cierto tiempo.

`open-inferior` *directorio* [*#:command* "*bin/guix*"] [Procedimiento]
: Abre el Guix inferior en *directorio*, ejecutando `directorio/command repl` o su equivalente. Devuelve `#f` si el inferior no pudo ser ejecutado.

Los procedimientos enumerados a continuación le permiten obtener y manipular paquetes de inferiores.

`inferior-packages` *inferior* [Procedimiento]
: Devuelve la lista de paquetes conocida por *inferior*.

`lookup-inferior-packages` *inferior nombre* [*versión*] [Procedimiento]
: Devuelve la lista ordenada de paquetes del inferior que corresponden con *nombre* en *inferior*, con los números de versión más altos primero. Si *versión* tiene un valor verdadero, devuelve únicamente paquetes con un número de versión cuyo prefijo es *versión*.

`inferior-package?` *obj* [Procedimiento]
Devuelve verdadero si *obj* es un paquete inferior.

`inferior-package-name` *paquete* [Procedimiento]
`inferior-package-version` *paquete* [Procedimiento]
`inferior-package-synopsis` *paquete* [Procedimiento]
`inferior-package-description` *paquete* [Procedimiento]
`inferior-package-home-page` *paquete* [Procedimiento]
`inferior-package-location` *paquete* [Procedimiento]
`inferior-package-inputs` *paquete* [Procedimiento]
`inferior-package-native-inputs` *paquete* [Procedimiento]
`inferior-package-propagated-inputs` *paquete* [Procedimiento]
`inferior-package-transitive-propagated-inputs` *paquete* [Procedimiento]
`inferior-package-native-search-paths` *paquete* [Procedimiento]
`inferior-package-transitive-native-search-paths` *paquete* [Procedimiento]
`inferior-package-search-paths` *paquete* [Procedimiento]
Estos procedimientos son la contraparte de los accesos a los registros de paquete (see Section 8.2.1 [Referencia de package], page 105). La mayor parte funcionan interrogando al inferior del que *paquete* viene, por lo que el inferior debe estar vivo cuando llama a dichos procedimientos.

Inferior packages can be used transparently like any other package or file-like object in G-expressions (see Section 8.12 [Expresiones-G], page 167). They are also transparently handled by the `packages->manifest` procedure, which is commonly used in manifests (see Section 5.2 [Invocación de guix package], page 36). Thus you can insert an inferior package pretty much anywhere you would insert a regular package: in manifests, in the `packages` field of your `operating-system` declaration, and so on.

## 5.10 Invocación de `guix describe`

A menudo desea responder a preguntas como: "¿Qué revisión de Guix estoy usando?" o "¿Qué canales estoy usando?" Esto es una información muy útil en muchas situaciones: si quiere *replicar* un entorno en una máquina diferente o cuenta de usuaria, si desea informar de un error o determinar qué cambio en los canales que usa lo causó, o si quiere almacenar el estado de su sistema por razones de reproducibilidad. La orden `guix describe` responde a estas preguntas.

Cuando se ejecuta desde un `guix` bajado con `guix pull`, `guix describe` muestra el/los canal/es desde el/los que se construyó, incluyendo la URL de su repositorio y los IDs de las revisiones (see Chapter 6 [Canales], page 69):

```
$ guix describe
Generation 10 Sep 03 2018 17:32:44 (current)
  guix e0fa68c
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: master
    commit: e0fa68c7718fffd33d81af415279d6ddb518f727
```

Si está familiarizado con el sistema de control de versiones Git, esto es similar a `git describe`; la salida también es similar a la de `guix pull --list-generations`, pero limi-

tada a la generación actual (see Section 5.7 [Invocación de guix pull], page 57). Debido a que el ID de revisión Git mostrado antes refiere sin ambigüedades al estado de Guix, esta información es todo lo necesario para describir la revisión de Guix que usa, y también para replicarla.

Para facilitar la replicación de Guix, también se le puede solicitar a `guix describe` devolver una lista de canales en vez de la descripción legible por humanos mostrada antes:

```
$ guix describe -f channels
(list (channel
        (name 'guix)
        (url "https://git.savannah.gnu.org/git/guix.git")
        (commit
          "e0fa68c7718fffd33d81af415279d6ddb518f727")
        (introduction
          (make-channel-introduction
            "9edb3f66fd807b096b48283debdcddccfea34bad"
            (openpgp-fingerprint
              "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA")))))
```

Puede almacenar esto en un archivo y se lo puede proporcionar a `guix pull -C` en otra máquina o en un momento futuro, lo que instanciará *esta revisión exacta de Guix* (see Section 5.7 [Invocación de guix pull], page 57). De aquí en adelante, ya que puede desplegar la misma revisión de Guix, puede también *replicar un entorno completo de software*. Nosotras humildemente consideramos que esto es *impresionante*, ¡y esperamos que le guste a usted también!

Los detalles de las opciones aceptadas por `guix describe` son las siguientes:

`--format=`*formato*
`-f` *formato*
: Produce salida en el *formato* especificado, uno de:

  `human`
  : produce salida legible por humanos;

  `channels`
  : produce una lista de especificaciones de canales que puede ser pasada a `guix pull -C` o instalada como `~/.config/guix/channels.scm` (see Section 5.7 [Invocación de guix pull], page 57);

  `channels-sans-intro`
  : como `channels`, pero se omite el campo `introduction`; se puede usar para producir una especificación de canal adecuada para la versión 1.1.0 de Guix y versiones anteriores—el campo `introduction` está relacionado con la verificación de canales (see Chapter 6 [Canales], page 69) y no está implementado en dichas versiones;

  `json`
  : produce una lista de especificaciones de canales en formato JSON;

  `recutils`
  : produce una lista de especificaciones de canales en formato Recutils.

`--list-formats`
: Muestra los formatos disponibles para la opción `--format`.

`--profile=`*perfil*
`-p` *perfil* Muestra información acerca del *perfil*.

## 5.11 Invocación de `guix archive`

La orden `guix archive` permite a las usuarias *exportar* archivos del almacén en un único archivador, e *importarlos* posteriormente en una máquina que ejecute Guix. En particular, permite que los archivos del almacén sean transferidos de una máquina al almacén de otra máquina.

> **Nota:** Si está buscando una forma de producir archivos en un formato adecuado para herramientas distintas a Guix, see Section 7.3 [Invocación de guix pack], page 92.

Para exportar archivos del almacén como un archivo por la salida estándar, ejecute:

```
guix archive --export opciones especificaciones...
```

*especificaciones* deben ser o bien nombres de archivos del almacén o especificaciones de paquetes, como las de `guix package` (see Section 5.2 [Invocación de guix package], page 36). Por ejemplo, la siguiente orden crea un archivo que contiene la salida `gui` del paquete `git` y la salida principal de `emacs`:

```
guix archive --export git:gui /gnu/store/...-emacs-24.3 > great.nar
```

Si los paquetes especificados no están todavía construidos, `guix archive` los construye automáticamente. El proceso de construcción puede controlarse mediante las opciones de construcción comunes (see Section 9.1.1 [Opciones comunes de construcción], page 181).

Para transferir el paquete `emacs` a una máquina conectada por SSH, se ejecutaría:

```
guix archive --export -r emacs | ssh otra-maquina guix archive --import
```

De manera similar, un perfil de usuaria completo puede transferirse de una máquina a otra de esta manera:

```
guix archive --export -r $(readlink -f ~/.guix-profile) | \
  ssh otra-maquina guix archive --import
```

No obstante, fíjese que, en ambos ejemplos, todo `emacs` y el perfil como también todas sus dependencias son transferidas (debido a la `-r`), independiente de lo que estuviese ya disponible en el almacén de la máquina objetivo. La opción `--missing` puede ayudar a esclarecer qué elementos faltan en el almacén objetivo. La orden `guix copy` simplifica y optimiza este proceso completo, así que probablemente es lo que debería usar en este caso (see Section 9.13 [Invocación de guix copy], page 233).

Cada elemento del almacén se escribe en formato de *archivo normalizado* o *nar* (descrito a continuación), y la salida de `guix archive --export` (y entrada de `guix archive --import`) es un *empaquetado nar*.

El formato "nar" es comparable a 'tar' en el espíritu, pero con diferencias que lo hacen más apropiado para nuestro propósito. Primero, en vez de almacenar todos los metadatos Unix de cada archivo, el formato nar solo menciona el tipo de archivo (normal, directorio o enlace simbólico); los permisos Unix y el par propietario/grupo se descartan. En segundo lugar, el orden en el cual las entradas de directorios se almacenan siempre siguen el orden de los nombres de archivos de acuerdo a la ordenación de cadenas en la localización C. Esto hace la producción del archivo completamente determinista.

El formato del empaquetado nar es esencialmente una concatenación de cero o más nar junto a metadatos para cada elemento del almacén que contiene: su nombre de archivo, referencias, derivación correspondiente y firma digital.

Durante la exportación, el daemon firma digitalmente los contenidos del archivo, y la firma digital se adjunta. Durante la importación, el daemon verifica la firma y rechaza la importación en caso de una firma inválida o si la clave firmante no está autorizada.

Las opciones principales son:

`--export` Exporta los archivos del almacén o paquetes (véase más adelante). Escribe el archivo resultante a la salida estándar.

Las dependencias *no* están incluidas en la salida, a menos que se use `--recursive`.

`-r`
`--recursive`
Cuando se combina con `--export`, instruye a `guix archive` para incluir las dependencias de los elementos dados en el archivo. Por tanto, el archivo resultante está auto-contenido: contiene la clausura de los elementos exportados del almacén.

`--import` Lee un archivo de la entrada estándar, e importa los archivos enumerados allí en el almacén. La operación se aborta si el archivo tiene una firma digital no válida, o si está firmado por una clave pública que no está entre las autorizadas (vea `--authorize` más adelante).

`--missing`
Lee una lista de nombres de archivos del almacén de la entrada estándar, uno por línea, y escribe en la salida estándar el subconjunto de estos archivos que faltan en el almacén.

`--generate-key[=`*parámetros*`]`
Genera un nuevo par de claves para el daemon. Esto es un prerrequisito antes de que los archivos puedan ser exportados con `--export`. Esta operación es habitualmente instantánea pero puede tomar tiempo si la piscina de entropía necesita tiene que rellenarse. En el sistema Guix `guix-service-type` se encarga de generar este par de claves en el primer arranque.

El par de claves generado se almacena típicamente bajo `/etc/guix`, en `signing-key.pub` (clave pública) y `signing-key.sec` (clave privada, que se debe mantener secreta). Cuando *parámetros* se omite, se genera una clave ECDSA usando la curva Ed25519, o, en versiones de Libgcrypt previas a la 1.6.0, es una clave RSA de 4096 bits. De manera alternativa, los *parámetros* pueden especificar parámetros `genkey` adecuados para Libgcrypt (see Section "General public-key related Functions" in *The Libgcrypt Reference Manual*).

`--authorize`
Autoriza importaciones firmadas con la clave pública pasada por la entrada estándar. La clave pública debe estar en el "formato avanzado de expresiones-s"—es decir, el mismo formato que el archivo `signing-key.pub`.

La lista de claves autorizadas se mantiene en el archivo editable por personas `/etc/guix/acl`. El archivo contiene "expresiones-s en formato avanzado"

(`https://people.csail.mit.edu/rivest/Sexp.text`) y está estructurado como una lista de control de acceso en el formato Infraestructura Simple de Clave Pública (SPKI) (`https://theworld.com/~cme/spki.txt`).

`--extract=`*directorio*
`-x` *directorio*

Lee un único elemento del archivo como es ofrecido por los servidores de sustituciones (see Section 5.3 [Sustituciones], page 46) y lo extrae a *directorio*. Esta es una operación de bajo nivel necesitada únicamente para casos muy concretos; véase a continuación.

For example, the following command extracts the substitute for Emacs served by `bordeaux.guix.gnu.org` to `/tmp/emacs`:

```
$ wget -O - \
  https://bordeaux.guix.gnu.org/nar/gzip/...-emacs-24.5 \
  | gunzip | guix archive -x /tmp/emacs
```

Los archivos de un único elemento son diferentes de los archivos de múltiples elementos producidos por `guix archive --export`; contienen un único elemento del almacén, y *no* embeben una firma. Por tanto esta operación *no* verifica la firma y su salida debe considerarse insegura.

El propósito primario de esta operación es facilitar la inspección de los contenidos de un archivo que provenga probablemente de servidores de sustituciones en los que no se confía (see Section 9.12 [Invocación de guix challenge], page 230).

`--list`
`-t`

Lee un único elemento del archivo como es ofrecido por los servidores de sustituciones (see Section 5.3 [Sustituciones], page 46) e imprime la lista de archivos que contiene, como en este ejemplo:

```
$ wget -O - \
  https://bordeaux.guix.gnu.org/nar/lzip/...-emacs-26.3 \
  | lzip -d | guix archive -t
```

# 6 Canales

Guix and its package collection are updated by running `guix pull`. By default `guix pull` downloads and deploys Guix itself from the official GNU Guix repository. This can be customized by providing a file specifying the set of *channels* to pull from (see Section 5.7 [Invocación de guix pull], page 57). A channel specifies the URL and branch of a Git repository to be deployed, and `guix pull` can be instructed to pull from one or more channels. In other words, channels can be used to *customize* and to *extend* Guix, as we will see below. Guix is able to take into account security concerns and deal with authenticated updates.

## 6.1 Especificación de canales adicionales

You can specify *additional channels* to pull from. To use a channel, write `~/.config/guix/channels.scm` to instruct `guix pull` to pull from it *in addition* to the default Guix channel(s):

```
;; Añade variaciones de paquetes sobre los que proporciona Guix.
(cons (channel
        (name 'paquetes-personalizados)
        (url "https://example.org/paquetes-personalizados.git"))
      %default-channels)
```

Fíjese que el fragmento previo es (¡como siempre!) código Scheme; usamos `cons` para añadir un canal a la lista de canales a la que la variable `%default-channels` hace referencia (see Section "Pairs" in *GNU Guile Reference Manual*). Con el archivo en este lugar, `guix pull` no solo construye Guix sino también los módulos de paquetes de su propio repositorio. El resultado en `~/.config/guix/current` es la unión de Guix con sus propios módulos de paquetes:

```
$ guix describe
Generation 19 Aug 27 2018 16:20:48
  guix d894ab8
    repository URL: https://git.savannah.gnu.org/git/guix.git
    branch: master
    commit: d894ab8e9bfabcefa6c49d9ba2e834dd5a73a300
  variant-packages dd3df5e
    repository URL: https://example.org/variant-packages.git
    branch: master
    commit: dd3df5e2c8818760a8fc0bd699e55d3b69fef2bb
```

The output of `guix describe` above shows that we're now running Generation 19 and that it includes both Guix and packages from the `variant-packages` channel (see Section 5.10 [Invocación de guix describe], page 64).

## 6.2 Uso de un canal de Guix personalizado

El canal llamado `guix` especifica de dónde debe descargarse el mismo Guix—sus herramientas de línea de órdenes y su colección de paquetes—. Por ejemplo, suponga que quiere actualizar de otra copia del repositorio Guix en `example.org`, y específicamente la rama

super-hacks, para ello puede escribir en ~/.config/guix/channels.scm esta especificación:

```
;; Le dice a 'guix pull' que use mi propio repositorio.
(list (channel
        (name 'guix)
        (url "https://example.org/otro-guix.git")
        (branch "super-hacks")))
```

From there on, guix pull will fetch code from the super-hacks branch of the repository at example.org. The authentication concern is addressed below (see Section 6.5 [Verificación de canales], page 72).

Note that you can specify a local directory on the url field above if the channel that you intend to use resides on a local file system. However, in this case guix checks said directory for ownership before any further processing. This means that if the user is not the directory owner, but wants to use it as their default, they will then need to set it as a safe directory in their global git configuration file. Otherwise, guix will refuse to even read it. Supposing your system-wide local directory is at /src/guix.git, you would then create a git configuration file at ~/.gitconfig with the following contents:

```
[safe]
        directory = /src/guix.git
```

This also applies to the root user unless when called with sudo by the directory owner.

## 6.3 Replicación de Guix

The guix describe command shows precisely which commits were used to build the instance of Guix we're using (see Section 5.10 [Invocación de guix describe], page 64). We can replicate this instance on another machine or at a different point in time by providing a channel specification "pinned" to these commits that looks like this:

```
;; Despliega unas revisiones específicas de mis canales de interés.
(list (channel
       (name 'guix)
       (url "https://git.savannah.gnu.org/git/guix.git")
       (commit "d894ab8e9bfabcefa6c49d9ba2e834dd5a73a300"))
      (channel
       (name 'paquetes-personalizados)
       (url "https://example.org/paquetes-personalizados.git")
       (branch "dd3df5e2c8818760a8fc0bd699e55d3b69fef2bb")))
```

To obtain this pinned channel specification, the easiest way is to run guix describe and to save its output in the channels format in a file, like so:

```
guix describe -f channels > channels.scm
```

The resulting channels.scm file can be passed to the -C option of guix pull (see Section 5.7 [Invocación de guix pull], page 57) or guix time-machine (see Section 5.8 [Invocación de guix time-machine], page 61), as in this example:

```
guix time-machine -C channels.scm -- shell python -- python3
```

Given the channels.scm file, the command above will always fetch the *exact same Guix instance*, then use that instance to run the exact same Python (see Section 7.1 [Invoking

guix shell], page 79). On any machine, at any time, it ends up running the exact same binaries, bit for bit.

Pinned channels address a problem similar to "lock files" as implemented by some deployment tools—they let you pin and reproduce a set of packages. In the case of Guix though, you are effectively pinning the entire package set as defined at the given channel commits; in fact, you are pinning all of Guix, including its core modules and command-line tools. You're also getting strong guarantees that you are, indeed, obtaining the exact same software.

Esto le proporciona superpoderes, lo que le permite seguir la pista de la procedencia de los artefactos binarios con un grano muy fino, y reproducir entornos de software a su voluntad—un tipo de capacidad de "meta-reproducibilidad", si lo desea. See Section 5.9 [Inferiores], page 62, para otro modo de tomar ventaja de estos superpoderes.

## 6.4 Customizing the System-Wide Guix

If you're running Guix System or building system images with it, maybe you will want to customize the system-wide `guix` it provides—specifically, `/run/current-system/profile/bin/guix`. For example, you might want to provide additional channels or to pin its revision.

This can be done using the `guix-for-channels` procedure, which returns a package for the given channels, and using it as part of your operating system configuration, as in this example:

```
(use-modules (guix channels))

(define my-channels
  ;; Channels that should be available to
  ;; /run/current-system/profile/bin/guix.
  (append
   (list (channel
          (name 'guix-science)
          (url "https://github.com/guix-science/guix-science")
          (branch "master")))
   %default-channels))

(operating-system
  ;; ...
  (services
    ;; Change the package used by 'guix-service-type'.
    (modify-services %base-services
      (guix-service-type
       config => (guix-configuration
                  (inherit config)
                  (channels my-channels)
                  (guix (guix-for-channels my-channels)))))))
```

The resulting operating system will have both the `guix` and the `guix-science` channels visible by default. The `channels` field of `guix-configuration` above further ensures that

`/etc/guix/channels.scm`, which is used by `guix pull`, specifies the same set of channels (see [guix-configuration-channels], page 286).

The `(gnu packages package-management)` module exports the `guix-for-channels` procedure, described below.

`guix-for-channels` *channels* [Procedure]

Return a package corresponding to *channels*.

The result is a "regular" package, which can be used in `guix-configuration` as shown above or in any other place that expects a package.

## 6.5 Verificación de canales

Las órdenes `guix pull` y `guix time-machine` *verifican* el código obtenido de los canales: se aseguran de que cada commit que se obtenga se encuentre firmado por una desarrolladora autorizada. El objetivo es proteger de modificaciones no-autorizadas al canal que podrían provocar que las usuarias ejecuten código pernicioso.

Como usuaria, debe proporcionar una *presentación del canal* en su archivo de canales de modo que Guix sepa como verificar su primera revision. La especificación de una canal, incluyendo su introducción, es más o menos así:

```
(channel
  (name 'un-canal)
  (url "https://example.org/un-canal.git")
  (introduction
   (make-channel-introduction
    "6f0d8cc0d88abb59c324b2990bfee2876016bb86"
    (openpgp-fingerprint
     "CABB A931 C0FF EEC6 900D  0CFB 090B 1199 3D9A EBB5"))))
```

La especificación previa muestra el nombre y la URL del canal. La llamada a `make-channel-introduction` especifica que la identificación de este canal empieza en la revisión `6f0d8cc...`, que está firmada por la clave de OpenPGP que tiene la huella `CABB A931...`.

En el canal principal, llamado `guix`, obtiene esta información de manera automática desde su instalación de Guix. Para otros canales, incluya la presentación del canal proporcionada por sus autoras en su archivo `channels.scm`. Asegúrese de obtener la presentación del canal desde una fuente confiable ya que es la raíz de su confianza.

Si tiene curiosidad sobre los mecanismos de identificación y verificación, ¡siga leyendo!

## 6.6 Channels with Substitutes

When running `guix pull`, Guix will first compile the definitions of every available package. This is an expensive operation for which substitutes (see Section 5.3 [Sustituciones], page 46) may be available. The following snippet in `channels.scm` will ensure that `guix pull` uses the latest commit with available substitutes for the package definitions: this is done by querying the continuous integration server at `https://ci.guix.gnu.org`.

```
(use-modules (guix ci))

(list (channel-with-substitutes-available
```

```
%default-guix-channel
"https://ci.guix.gnu.org"))
```

Note that this does not mean that all the packages that you will install after running `guix pull` will have available substitutes. It only ensures that `guix pull` will not try to compile package definitions. This is particularly useful when using machines with limited resources.

## 6.7 Creación de un canal

Let's say you have a bunch of custom package variants or personal packages that you think would make little sense to contribute to the Guix project, but would like to have these packages transparently available to you at the command line. By creating a *channel*, you can use and publish such a package collection. This involves the following steps:

1. A channel lives in a Git repository so the first step, when creating a channel, is to create its repository:

   ```
   mkdir my-channel
   cd my-channel
   git init
   ```

2. The next step is to create files containing package modules (see Section 8.1 [Módulos de paquetes], page 101), each of which will contain one or more package definitions (see Section 8.2 [Definición de paquetes], page 102). A channel can provide things other than packages, such as build systems or services; we're using packages as it's the most common use case.

   For example, Alice might want to provide a module called `(alice packages greetings)` that will provide her favorite "hello world" implementations. To do that Alice will create a directory corresponding to that module name.

   ```
   mkdir -p alice/packages
   $EDITOR alice/packages/greetings.scm
   git add alice/packages/greetings.scm
   ```

   You can name your package modules however you like; the main constraint to keep in mind is to avoid name clashes with other package collections, which is why our hypothetical Alice wisely chose the `(alice packages ...)` name space.

   Note that you can also place modules in a sub-directory of the repository; see Section 6.8 [Módulos de paquetes en un subdirectorio], page 74, for more info on that.

3. With this first module in place, the next step is to test the packages it provides. This can be done with `guix build`, which needs to be told to look for modules in the Git checkout. For example, assuming `(alice packages greetings)` provides a package called `hi-from-alice`, Alice will run this command from the Git checkout:

   ```
   guix build -L. hi-from-alice
   ```

   ... where `-L.` adds the current directory to Guile's load path (see Section "Load Paths" in *GNU Guile Reference Manual*).

4. It might take Alice a few iterations to obtain satisfying package definitions. Eventually Alice will commit this file:

   ```
   git commit
   ```

Como autora de un canal, considere adjuntar el material para la identificación a su canal de modo que las usuarias puedan verificarlo. See Section 6.5 [Verificación de canales], page 72, y Section 6.10 [Especificación de autorizaciones del canal], page 75, para obtener información sobre cómo hacerlo.

5. To use Alice's channel, anyone can now add it to their channel file (see Section 6.1 [Especificación de canales adicionales], page 69) and run `guix pull` (see Section 5.7 [Invocación de guix pull], page 57):

   ```
   $EDITOR ~/.config/guix/channels.scm
   guix pull
   ```

   Guix will now behave as if the root directory of that channel's Git repository had been permanently added to the Guile load path. In this example, `(alice packages greetings)` will automatically be found by the `guix` command.

Voilà!

**Aviso:** Before you publish your channel, we would like to share a few words of caution:

- Antes de publicar un canal, por favor considere contribuir sus definiciones de paquete al propio Guix (see Chapter 22 [Contribuir], page 734). Guix como proyecto es abierto a software libre de todo tipo, y los paquetes en el propio Guix están disponibles para todas las usuarias de Guix y se benefician del proceso de gestión de calidad del proyecto.
- Package modules and package definitions are Scheme code that uses various programming interfaces (APIs). We, Guix developers, never change APIs gratuitously, but we do *not* commit to freezing APIs either. When you maintain package definitions outside Guix, we consider that *the compatibility burden is on you*.
- Corolario: si está usando un canal externo y el canal se rompe, por favor *informe del problema a las autoras del canal*, no al proyecto Guix.

¡Ha quedado advertida! Habiendo dicho esto, creemos que los canales externos son una forma práctica de ejercitar su libertad para aumentar la colección de paquetes de Guix y compartir su mejoras, que son pilares básicos del software libre (`https://www.gnu.org/philosophy/free-sw.html`). Por favor, envíenos un correo a `guix-devel@gnu.org` si quiere hablar sobre esto.

## 6.8 Módulos de paquetes en un subdirectorio

Como autora de un canal, es posible que desee mantener los módulos de su canal en un subdirectorio. Si sus módulos se encuentran en el subdirectorio `guix`, debe añadir un archivo `.guix-channel` de metadatos que contenga:

```
(channel
  (version 0)
  (directory "guix"))
```

The modules must be **underneath** the specified directory, as the `directory` changes Guile's `load-path`. For example, if `.guix-channel` has `(directory "base")`, then a module defined as `(define-module (gnu packages fun))` must be located at `base/gnu/packages/fun.scm`.

Doing this allows for only parts of a repository to be used as a channel, as Guix expects valid Guile modules when pulling. For instance, `guix deploy` machine configuration files are not valid Guile modules, and treating them as such would make `guix pull` fail.

## 6.9 Declaración de dependencias de canales

Las autoras de canales pueden decidir aumentar una colección de paquetes proporcionada por otros canales. Pueden declarar su canal como dependiente de otros canales en el archivo de metadatos `.guix-channel`, que debe encontrarse en la raíz del repositorio del canal.

Este archivo de metadatos debe contener una expresión-S simple como esta:

```
(channel
 (version 0)
 (dependencies
  (channel
   (name some-collection)
   (url "https://example.org/first-collection.git")

   ;; The 'introduction' bit below is optional: you would
   ;; provide it for dependencies that can be authenticated.
   (introduction
    (channel-introduction
      (version 0)
      (commit "a8883b58dc82e167c96506cf05095f37c2c2c6cd")
      (signer "CABB A931 C0FF EEC6 900D  0CFB 090B 1199 3D9A EBB5"))))
  (channel
   (name some-other-collection)
   (url "https://example.org/second-collection.git")
   (branch "testing"))))
```

En el ejemplo previo, este canal se declara como dependiente de otros dos canales, que se obtendrán de manera automática. Los módulos proporcionados por el canal se compilarán en un entorno donde los módulos de todos estos canales declarados estén disponibles.

De cara a la confianza proporcionada y el esfuerzo que supondrá su mantenimiento, debería evitar depender de canales que no controle, y debería intentar minimizar el número de dependencias.

## 6.10 Especificación de autorizaciones del canal

Como hemos visto previamente, Guix se asegura de que el código fuente que obtiene de los canales proviene de desarrolladoras autorizadas. Como autora del canal, es necesario que especifique la lista de desarrolladoras autorizadas en el archivo `.guix-authorizations` del repositorio Git del canal. Las reglas para la verificación son simples: cada revisión debe firmarse con una de las claves enumeradas en el archivo `.guix-authorizations` de

la revisión o revisiones anteriores[1] El archivo `.guix-authorizations` tiene está estructura básica:

```
;; Archivo '.guix-authorizations' de ejemplo.

(authorizations
 (version 0)               ;versión de formato actual

 (("AD17 A21E F8AE D8F1 CC02  DBD9 F8AE D8F1 765C 61E3"
   (name "alicia"))
  ("CABB A931 C0FF EEC6 900D  0CFB 090B 1199 3D9A EBB5"
   (name "carlos"))
  ("2A39 3FFF 68F4 EF7A 3D29  12AF 68F4 EF7A 22FB B2D5"
   (name "rober"))))
```

Cada huella va seguida de pares clave/valor opcionales, como en el ejemplo siguiente. Actualmente se ignoran dichos pares.

Estas reglas de verificación dan lugar a un problema "del huevo y la gallina": ¿cómo se verifica la primera revisión? En relación con esto: ¿cómo se gestionan los canales cuyo repositorio tiene en su historia revisiones sin firmar y carece del archivo `.guix-authorizations`? ¿Y cómo creamos un nuevo canal separado en base a un canal ya existente?

Channel introductions answer these questions by describing the first commit of a channel that should be authenticated. The first time a channel is fetched with `guix pull` or `guix time-machine`, the command looks up the introductory commit and verifies that it is signed by the specified OpenPGP key. From then on, it authenticates commits according to the rule above. Authentication fails if the target commit is neither a descendant nor an ancestor of the introductory commit.

De manera adicional, su canal debe proporcionar todas las claves públicas que hayan sido mencionadas en `.guix-authorizations`, almacenadas como archivos `.key`, los cuales pueden ser binarios o tener "armadura ASCII". De manera predeterminada, esos archivos `.key` se buscan en la rama llamada `keyring` pero puede especificar una rama diferente en `.guix-channel` de este modo:

```
(channel
  (version 0)
  (keyring-reference "mi-rama-de-claves"))
```

En resumen, como autora de un canal, hay tres cosas que debe hacer para permitir que las usuarias verifiquen su código:

1. Exportar las claves OpenPGP de quienes contribuyan en el presente y quienes hayan contribuido en el pasado con `gpg --export` y almacenarlas en archivos `.key`, de manera predeterminada en una rama llamada `keyring` (recomendamos que sea una *rama huérfana*).

[1] Las revisiones en Git forman un *grafo acíclico dirigido* (DAG). Cada revisión puede tener cero o más antecesores; las revisiones "normales" tienen un antecesor, y las revisiones de mezcla tienen dos antecesores. Lea el artículo en inglés *Git for Computer Scientists* (`https://eagain.net/articles/git-for-computer-scientists/`) para una buena introducción.

2. Introducir un archivo inicial `.guix-authorizations` en el repositorio del canal. Hágalo con una revisión firmada (see Section 22.12 [Acceso al repositorio], page 769, para más información sobre cómo firmar revisiones).
3. Anuncie la presentación del canal, por ejemplo, en la página web de su canal. La presentación del canal, como hemos visto antes, es el par revisión/clave—es decir, la revisión que introdujo el archivo `.guix-authorizations`, y la huella de la clave de OpenPGP usada para firmarlo.

Before pushing to your public Git repository, you can run `guix git authenticate` to verify that you did sign all the commits you are about to push with an authorized key:

```
guix git authenticate revisión firma
```

donde *revisión* y *firma* son la presentación de su canal. See Section 7.5 [Invocación de guix git authenticate], page 99, para obtener más detalles.

Publicar un canal firmado requiere disciplina: cualquier error, como una revisión sin firmar o una revisión firmada por una clave no autorizada, evitará que las usuarias obtengan nuevas versiones de su canal—bueno, ¡ese es principalmente el objetivo de la verificación! Preste especial atención a la mezcla de ramas: las revisiones de mezcla se consideran auténticas únicamente en caso de que la clave que firma esté presente en el archivo `.guix-authorizations` de *ambas* ramas.

## 6.11 URL primaria

Las autoras pueden declarar la URL primaria del repositorio Git de su canal en el archivo `.guix-channel` de esta manera:

```
(channel
  (version 0)
  (url "https://example.org/guix.git"))
```

This allows `guix pull` to determine whether it is pulling code from a mirror of the channel; when that is the case, it warns the user that the mirror might be stale and displays the primary URL. That way, users cannot be tricked into fetching code from a stale mirror that does not receive security updates.

Esta caraterística únicamente tiene sentido en repositorios verificables, como el canal oficial `guix`, en el que `guix pull` se asegura de verificar la autenticidad del código que obtiene.

## 6.12 Escribir de noticias del canal

Las autoras los canales pueden querer ocasionalmente comunicar información a sus usuarias acerca de cambios importantes en el canal. Podrían mandar un correo a todo el mundo, pero esto no es tan conveniente.

En vez de eso, los canales proporcionan un *archivo de noticias*; cuando las usuarias de un canal ejecutan `guix pull`, dicho archivo de noticias se lee automáticamente y `guix pull --news` puede mostrar los anuncios que correspondan a las nuevas revisiones que se han obtenido, si existen.

Para hacerlo, las autoras del canal deben declarar primero el nombre del archivo de noticias en su archivo `.guix-channel`:

```
(channel
```

```
  (version 0)
  (news-file "etc/noticias.txt"))
```

El archivo de noticias en sí, `etc/noticias.txt` en este ejemplo, debe ser similar a este:

```
(channel-news
  (version 0)
  (entry (tag "the-bug-fix")
         (title (en "Fixed terrible bug")
                (fr "Oh la la")
                (es "Podemos volver a dormir en calma"))
         (body (en "@emph{Good news}!  It's fixed!")
               (eo "Certe ĝi pli bone funkcias nun!")
               (es "¡Al fin se ha corregido el error!")))
  (entry (commit "bdcabe815cd28144a2d2b4bc3c5057b051fa9906")
         (title (en "Added a great package")
                (ca "Què vol dir guix?")
                (es "Nuevo paquete añadido"))
         (body (en "Don't miss the @code{hello} package!")
               (es "Atención a la versátil herramienta @code{hello}"))))
```

Aunque el archivo de noticias use sintaxis de Scheme evite nombrarlo con `.scm` como extensión o se usará cuando se construya el canal, lo que emitirá un error debido a que no es un módulo válido. También puede mover el módulo del canal a un subdirectorio y almacenar el archivo de noticias en otro directorio.

Este archivo consiste en una lista de *entradas de noticias*. Cada entrada[2] se asocia a una revisión o una etiqueta: describe los cambios llevados a cabo en ella, y posiblemente también en revisiones anteriores. Las usuarias ven las entradas únicamente la primera vez que obtienen la revisión a la que la entrada hace referencia.

El campo del título (`title`) debe ser un resumen de una línea mientras que el cuerpo de la noticia (`body`) puede ser arbitrariamente largo, y ambos pueden contener marcas de Texinfo (see Section "Overview" in *GNU Texinfo*). Tanto el título como el cuerpo son una lista de tuplas de etiqueta de lengua y mensaje, lo que permite a `guix pull` mostrar las noticias en la lengua que corresponde a la localización de la usuaria.

Si desea traducir las noticias siguiendo un flujo de trabajo basado en gettext, puede extraer las cadenas traducibles con `xgettext` (see Section "xgettext Invocation" in *GNU Gettext Utilities*). Por ejemplo, asumiendo que escribe las entradas de noticias primero en inglés, la siguiente orden crea un archivo PO que contiene las cadenas a traducir:

```
xgettext -o news.po -l scheme -ken etc/news.txt
```

En resumen, sí, puede usar su canal como un blog. Pero tenga en cuenta que esto puede que *no sea exactamente* lo que sus usuarias podrían esperar.

[2] NdT: "entry" en inglés

# 7 Desarrollo

Si es una desarrolladora de software, Guix le proporciona herramientas que debería encontrar útiles—independientemente del lenguaje en el que desarrolle actualmente. Esto es sobre lo que trata este capítulo.

The `guix shell` command provides a convenient way to set up one-off software environments, be it for development purposes or to run a command without installing it in your profile. The `guix pack` command allows you to create *application bundles* that can be easily distributed to users who do not run Guix.

## 7.1 Invoking `guix shell`

The purpose of `guix shell` is to make it easy to create one-off software environments, without changing one's profile. It is typically used to create development environments; it is also a convenient way to run applications without "polluting" your profile.

> **Nota:** The `guix shell` command was recently introduced to supersede `guix environment` (see Section 7.2 [Invocación de guix environment], page 86). If you are familiar with `guix environment`, you will notice that it is similar but also—we hope!—more convenient.

La sintaxis general es:

```
guix shell [options] [package...]
```

The following example creates an environment containing Python and NumPy, building or downloading any missing package, and runs the `python3` command in that environment:

```
guix shell python python-numpy -- python3
```

Note that it is necessary to include the main `python` package in this command even if it is already installed into your environment. This is so that the shell environment knows to set `PYTHONPATH` and other related variables. The shell environment cannot check the previously installed environment, because then it would be non-deterministic. This is true for most libraries: their corresponding language package should be included in the shell invocation.

> **Nota:**
>
> `guix shell` can be also be used as a script interpreter, also known as *shebang*. Here is an example self-contained Python script making use of this feature:
>
> ```
> #!/usr/bin/env -S guix shell python python-numpy -- python3
> import numpy
> print("This is numpy", numpy.version.version)
> ```
>
> You may pass any `guix shell` option, but there's one caveat: the Linux kernel has a limit of 127 bytes on shebang length.

Development environments can be created as in the example below, which spawns an interactive shell containing all the dependencies and environment variables needed to work on Inkscape:

```
guix shell --development inkscape
```

Exiting the shell places the user back in the original environment before `guix shell` was invoked. The next garbage collection (see Section 5.6 [Invocación de guix gc], page 53)

may clean up packages that were installed in the environment and that are no longer used outside of it.

As an added convenience, `guix shell` will try to do what you mean when it is invoked interactively without any other arguments as in:

```
guix shell
```

If it finds a `manifest.scm` in the current working directory or any of its parents, it uses this manifest as though it was given via `--manifest`. Likewise, if it finds a `guix.scm` in the same directories, it uses it to build a development profile as though both `--development` and `--file` were present. In either case, the file will only be loaded if the directory it resides in is listed in `~/.config/guix/shell-authorized-directories`. This provides an easy way to define, share, and enter development environments.

By default, the shell session or command runs in an *augmented* environment, where the new packages are added to search path environment variables such as `PATH`. You can, instead, choose to create an *isolated* environment containing nothing but the packages you asked for. Passing the `--pure` option clears environment variable definitions found in the parent environment[1]; passing `--container` goes one step further by spawning a *container* isolated from the rest of the system:

```
guix shell --container emacs gcc-toolchain
```

The command above spawns an interactive shell in a container where nothing but `emacs`, `gcc-toolchain`, and their dependencies is available. The container lacks network access and shares no files other than the current working directory with the surrounding environment. This is useful to prevent access to system-wide resources such as `/usr/bin` on foreign distros.

This `--container` option can also prove useful if you wish to run a security-sensitive application, such as a web browser, in an isolated environment. For example, the command below launches Ungoogled-Chromium in an isolated environment, which:

- shares network access with the host
- inherits host's environment variables `DISPLAY` and `XAUTHORITY`
- has access to host's authentication records from the `XAUTHORITY` file
- has no information about host's current directory

```
guix shell --container --network --no-cwd ungoogled-chromium \
  --preserve='^XAUTHORITY$' --expose="${XAUTHORITY}" \
  --preserve='^DISPLAY$' -- chromium
```

`guix shell` defines the `GUIX_ENVIRONMENT` variable in the shell it spawns; its value is the file name of the profile of this environment. This allows users to, say, define a specific prompt for development environments in their `.bashrc` (see Section "Bash Startup Files" in *The GNU Bash Reference Manual*):

```
if [ -n "$GUIX_ENVIRONMENT" ]
then
    export PS1="\u@\h \w [dev]\$ "
fi
```

[1] Be sure to use the `--check` option the first time you use `guix shell` interactively to make sure the shell does not undo the effect of `--pure`.

... o para explorar el perfil:

```
$ ls "$GUIX_ENVIRONMENT/bin"
```

Las opciones disponibles se resumen a continuación.

`--check` Set up the environment and check whether the shell would clobber environment variables. It's a good idea to use this option the first time you run `guix shell` for an interactive session to make sure your setup is correct.

For example, if the shell modifies the `PATH` environment variable, report it since you would get a different environment than what you asked for.

Such problems usually indicate that the shell startup files are unexpectedly modifying those environment variables. For example, if you are using Bash, make sure that environment variables are set or modified in `~/.bash_profile` and *not* in `~/.bashrc`—the former is sourced only by log-in shells. See Section "Bash Startup Files" in *The GNU Bash Reference Manual*, for details on Bash start-up files.

`--development`
`-D` Cause `guix shell` to include in the environment the dependencies of the following package rather than the package itself. This can be combined with other packages. For instance, the command below starts an interactive shell containing the build-time dependencies of GNU Guile, plus Autoconf, Automake, and Libtool:

```
guix shell -D guile autoconf automake libtool
```

`--expression=`*expr*
`-e` *expr* Crea un entorno para el paquete o lista de paquetes a los que evalúa *expr*.

Por ejemplo, ejecutando:

```
guix shell -D -e '(@ (gnu packages maths) petsc-openmpi)'
```

inicia un shell con el entorno para esta variante específica del paquete PETSc.

Ejecutar:

```
guix shell -e '(@ (gnu) %base-packages)'
```

inicia un shell con todos los paquetes básicos del sistema disponibles.

Las órdenes previas usan únicamente la salida predeterminada de los paquetes dados. Para seleccionar otras salidas, tuplas de dos elementos pueden ser especificadas:

```
guix shell -e '(list (@ (gnu packages bash) bash) "include")'
```

See [package-development-manifest], page 122, for information on how to write a manifest for the development environment of a package.

`--file=`*archivo*
`-f` *archivo*
Create an environment containing the package or list of packages that the code within *file* evaluates to.

Como un ejemplo, *archivo* puede contener una definición como esta (see Section 8.2 [Definición de paquetes], page 102):

```
(use-modules (guix)
```

```
                    (gnu packages gdb)
                    (gnu packages autotools)
                    (gnu packages texinfo))

;; Augment the package definition of GDB with the build tools
;; needed when developing GDB (and which are not needed when
;; simply installing it.)
(package
  (inherit gdb)
  (native-inputs (modify-inputs (package-native-inputs gdb)
                   (prepend autoconf-2.69 automake texinfo))))
```

With the file above, you can enter a development environment for GDB by running:

```
guix shell -D -f gdb-devel.scm
```

`--manifest=`*archivo*
`-m` *archivo*

Crea un entorno para los paquetes contenidos en el objeto manifest devuelto por el código Scheme en *file*. Esta opción se puede repetir varias veces, en cuyo caso los manifiestos se concatenan.

Esto es similar a la opción del mismo nombre en `guix package` (see [profile-manifest], page 40) y usa los mismos archivos de manifiesto.

See Section 8.4 [Writing Manifests], page 119, for information on how to write a manifest. See `--export-manifest` below on how to obtain a first manifest.

`--export-manifest`

Write to standard output a manifest suitable for `--manifest` corresponding to given command-line options.

This is a way to "convert" command-line arguments into a manifest. For example, imagine you are tired of typing long lines and would like to get a manifest equivalent to this command line:

```
guix shell -D guile git emacs emacs-geiser emacs-geiser-guile
```

Just add `--export-manifest` to the command line above:

```
guix shell --export-manifest \
  -D guile git emacs emacs-geiser emacs-geiser-guile
```

... and you get a manifest along these lines:

```
(concatenate-manifests
  (list (specifications->manifest
          (list "git"
                "emacs"
                "emacs-geiser"
                "emacs-geiser-guile"))
        (package->development-manifest
          (specification->package "guile"))))
```

You can store it into a file, say `manifest.scm`, and from there pass it to `guix shell` or indeed pretty much any `guix` command:

```
guix shell -m manifest.scm
```

Voilà, you've converted a long command line into a manifest! That conversion process honors package transformation options (see Section 9.1.2 [Opciones de transformación de paquetes], page 184) so it should be lossless.

`--profile=`*perfil*
`-p` *perfil* Create an environment containing the packages installed in *profile*. Use `guix package` (see Section 5.2 [Invocación de guix package], page 36) to create and manage profiles.

`--pure` Olvida las variables de entorno existentes cuando se construye un nuevo entorno, excepto aquellas especificadas con `--preserve` (véase más adelante). Esto tiene el efecto de crear un entorno en el que las rutas de búsqueda únicamente contienen las entradas del paquete.

`--preserve=`*regexp*
`-E` *regexp* Cuando se usa junto a `--pure`, preserva las variables de entorno que corresponden con *regexp*—en otras palabras, las pone en una lista de variables de entorno que deben preservarse. Esta opción puede repetirse varias veces.

```
guix shell --pure --preserve=^SLURM openmpi ... \
  -- mpirun ...
```

Este ejemplo ejecuta `mpirun` en un contexto donde las únicas variables de entorno definidas son `PATH`, variables de entorno cuyo nombre empiece con '`SLURM`', así como las variables "preciosas" habituales (`HOME`, `USER`, etc.).

`--search-paths`
Muestra las definiciones de variables de entorno que componen el entorno.

`--system=`*sistema*
`-s` *sistema*
Intenta construir para *sistema*—por ejemplo, `i686-linux`.

`--container`
`-C` Ejecuta la *orden* en un contenedor aislado. El directorio actual fuera del contenedor es asociado al interior del contenedor. Adicionalmente, a menos que se fuerce con `--user`, un directorio de prueba de la usuaria se crea de forma que coincida con el directorio actual de la usuaria, y `/etc/passwd` se configura adecuadamente.

El proceso lanzado se ejecuta como el usuario actual fuera del contenedor. Dentro del contenedor, tiene el mismo UID y GID que el usuario actual, a menos que se proporcione `--user` (véase más adelante).

`--network`
`-N` Para contenedores, comparte el espacio de nombres de red con el sistema anfitrión. Los contenedores creados sin esta opción únicamente tienen acceso a la red local.

`--link-profile`
`-P` For containers, link the environment profile to `~/.guix-profile` within the container and set `GUIX_ENVIRONMENT` to that. This is equivalent to making

`~/.guix-profile` a symlink to the actual profile within the container. Linking will fail and abort the environment if the directory already exists, which will certainly be the case if `guix shell` was invoked in the user's home directory.

Determinados paquetes se configuran para buscar en `~/.guix-profile` archivos de configuración y datos;[2] `--link-profile` permite a estos programas operar de la manera esperada dentro del entorno.

`--user=`*usuaria*
`-u` *usuaria*

Para contenedores, usa el nombre de usuaria *usuaria* en vez de la actual. La entrada generada en `/etc/passwd` dentro del contenedor contendrá el nombre *usuaria*; su directorio será `/home/`*usuaria* y ningún dato GECOS de la usuaria se copiará. Más aún, el UID y GID dentro del contenedor son 1000. *usuaria* no debe existir en el sistema.

Adicionalmente, cualquier ruta compartida o expuesta (véanse `--share` y `--expose` respectivamente) cuyo destino esté dentro de la carpeta actual de la usuaria será reasociada en relación a `/home/`*usuaria*; esto incluye la relación automática del directorio de trabajo actual.

```
# will expose paths as /home/foo/wd, /home/foo/test, and /home/foo/target
cd $HOME/wd
guix shell --container --user=foo \
     --expose=$HOME/test \
     --expose=/tmp/target=$HOME/target
```

Mientras esto limita el escape de la identidad de la usuaria a través de las rutas de sus directorios y cada uno de los campos de usuaria, esto es únicamente un componente útil de una solución de privacidad/anonimato más amplia—no una solución completa.

`--no-cwd` El comportamiento predeterminado con contenedores es compartir el directorio de trabajo actual con el contenedor aislado e inmediatamente cambiar a dicho directorio dentro del contenedor. Si no se desea este comportamiento, `--no-cwd` indica que el directorio actual *no* se compartirá automáticamente y, en vez de cambiar a dicho directorio, se cambiará al directorio de la usuaria dentro del contenedor. Véase también `--user`.

`--expose=`*fuente*`[=`*destino*`]`
`--share=`*fuente*`[=`*destino*`]`

En contenedores, la `--expose` expone el sistema de archivos *fuente* del sistema anfitrión como un sistema de archivos de solo-lectura *destino* dentro del contenedor. `--share` de la misma manera expone el sistema de archivos con posibilidad de escritura. Si no se especifica *destino*, *fuente* se usa como el punto de montaje en el contenedor.

El ejemplo a continuación lanza una sesión interactiva de Guile en un contenedor donde el directorio principal de la usuaria es accesible en modo solo-lectura a través del directorio `/intercambio`:

[2] Por ejemplo, el paquete `fontconfig` inspecciona `~/.guix-profile/share/fonts` en busca de nuevas tipografías.

```
guix shell --container --expose=$HOME=/exchange guile -- guile
```

`--symlink=`*spec*
`-S` *spec* For containers, create the symbolic links specified by *spec*, as documented in [pack-symlink-option], page 98.

`--emulate-fhs`
`-F` When used with `--container`, emulate a Filesystem Hierarchy Standard (FHS) (`https://refspecs.linuxfoundation.org/fhs.shtml`) configuration within the container, providing `/bin`, `/lib`, and other directories and files specified by the FHS.

As Guix deviates from the FHS specification, this option sets up the container to more closely mimic that of other GNU/Linux distributions. This is useful for reproducing other development environments, testing, and using programs which expect the FHS specification to be followed. With this option, the container will include a version of glibc that will read `/etc/ld.so.cache` within the container for the shared library cache (contrary to glibc in regular Guix usage) and set up the expected FHS directories: `/bin`, `/etc`, `/lib`, and `/usr` from the container's profile.

`--nesting`
`-W` When used with `--container`, provide Guix *inside* the container and arrange so that it can interact with the build daemon that runs outside the container. This is useful if you want, within your isolated container, to create other containers, as in this sample session:

```
$ guix shell -CW coreutils
[env]$ guix shell -C guile -- guile -c '(display "hello!\n")'
hello!
[env]$ exit
```

The session above starts a container with `coreutils` programs available in `PATH`. From there, we spawn `guix shell` to create a *nested* container that provides nothing but Guile.

Another example is evaluating a `guix.scm` file that is untrusted, as shown here:

```
guix shell -CW -- guix build -f guix.scm
```

The `guix build` command as executed above can only access the current directory.

Under the hood, the `-W` option does several things:

- map the daemon's socket (by default `/var/guix/daemon-socket/socket`) inside the container;
- map the whole store (by default `/gnu/store`) inside the container such that store items made available by nested `guix` invocations are visible;
- add the currently-used `guix` command to the profile in the container, such that `guix describe` returns the same state inside and outside the container;
- share the cache (by default `~/.cache/guix`) with the host, to speed up operations such as `guix time-machine` and `guix shell`.

`--rebuild-cache`

In most cases, `guix shell` caches the environment so that subsequent uses are instantaneous. Least-recently used cache entries are periodically removed. The cache is also invalidated, when using `--file` or `--manifest`, anytime the corresponding file is modified.

The `--rebuild-cache` forces the cached environment to be refreshed. This is useful when using `--file` or `--manifest` and the `guix.scm` or `manifest.scm` file has external dependencies, or if its behavior depends, say, on environment variables.

`--root=`*archivo*
`-r` *archivo*

Hace que *archivo* sea un enlace simbólico al perfil para este entorno, y lo registra como una raíz del recolector de basura.

Esto es útil si desea proteger su entorno de la recolección de basura, hacerlo "persistente".

When this option is omitted, `guix shell` caches profiles so that subsequent uses of the same environment are instantaneous—this is comparable to using `--root` except that `guix shell` takes care of periodically removing the least-recently used garbage collector roots.

In some cases, `guix shell` does not cache profiles—e.g., if transformation options such as `--with-latest` are used. In those cases, the environment is protected from garbage collection only for the duration of the `guix shell` session. This means that next time you recreate the same environment, you could have to rebuild or re-download packages.

See Section 5.6 [Invocación de guix gc], page 53, for more on GC roots.

`guix shell` also supports all of the common build options that `guix build` supports (see Section 9.1.1 [Opciones comunes de construcción], page 181) as well as package transformation options (see Section 9.1.2 [Opciones de transformación de paquetes], page 184).

## 7.2 Invocación de `guix environment`

The purpose of `guix environment` is to assist in creating development environments.

**Deprecation warning:** The `guix environment` command is deprecated in favor of `guix shell`, which performs similar functions but is more convenient to use. See Section 7.1 [Invoking guix shell], page 79.

Being deprecated, `guix environment` is slated for eventual removal, but the Guix project is committed to keeping it until May 1st, 2023. Please get in touch with us at `guix-devel@gnu.org` if you would like to discuss it.

La sintaxis general es:

```
guix environment opciones paquete...
```

El ejemplo siguiente lanza un nuevo shell preparado para el desarrollo de GNU Guile:

```
guix environment guile
```

Si las dependencias necesarias no están construidas todavía, `guix environment` las construye automáticamente. El entorno del nuevo shell es una versión aumentada del entorno

en el que `guix environment` se ejecutó. Contiene las rutas de búsqueda necesarias para la construcción del paquete proporcionado añadidas a las variables ya existentes. Para crear un entorno "puro", donde las variables de entorno previas no existen, use la opción `--pure`[3].

Exiting from a Guix environment is the same as exiting from the shell, and will place the user back in the old environment before `guix environment` was invoked. The next garbage collection (see Section 5.6 [Invocación de guix gc], page 53) will clean up packages that were installed from within the environment and are no longer used outside of it.

`guix environment` define la variable `GUIX_ENVIRONMENT` en el shell que lanza; su valor es el nombre de archivo del perfil para este entorno. Esto permite a las usuarias, digamos, definir un prompt para entornos de desarrollo en su `.bashrc` (see Section "Bash Startup Files" in *The GNU Bash Reference Manual*):

```
if [ -n "$GUIX_ENVIRONMENT" ]
then
    export PS1="\u@\h \w [dev]\$ "
fi
```

... o para explorar el perfil:

```
$ ls "$GUIX_ENVIRONMENT/bin"
```

Adicionalmente, más de un paquete puede ser especificado, en cuyo caso se usa la unión de las entradas de los paquetes proporcionados. Por ejemplo, la siguiente orden lanza un shell donde todas las dependencias tanto de Guile como de Emacs están disponibles:

```
guix environment guile emacs
```

A veces no se desea una sesión interactiva de shell. Una orden arbitraria se puede invocar usando el valor `--` para separar la orden del resto de los parámetros:

```
guix environment guile -- make -j4
```

In other situations, it is more convenient to specify the list of packages needed in the environment. For example, the following command runs `python` from an environment containing Python 3 and NumPy:

```
guix environment --ad-hoc python-numpy python -- python3
```

Es más, se pueden desear las dependencias de un paquete y también algunos paquetes adicionales que no son dependencias ni en tiempo de construcción ni en el de ejecución, pero son útiles no obstante para el desarrollo. Por esta razón, la opción `--ad-hoc` es posicional. Los paquetes que aparecen antes de `--ad-hoc` se interpretan como paquetes cuyas dependencias se añadirán al entorno. Los paquetes que aparecen después se interpretan como paquetes que se añadirán directamente al entorno. Por ejemplo, la siguiente orden crea un entorno de desarrollo Guix que incluye adicionalmente Git y strace:

```
guix environment --pure guix --ad-hoc git strace
```

En ocasiones es deseable aislar el entorno tanto como sea posible, para obtener la máxima pureza y reproducibilidad. En particular, cuando se usa Guix en una distribución anfitriona

[3] Las usuarias habitualmente aumentan de forma incorrecta las variables de entorno como `PATH` en su archivo `~/.bashrc`. Como consecuencia, cuando `guix environment` se ejecuta, Bash puede leer `~/.bashrc`, por tanto introduciendo "impurezas" en esas variables de entorno. Es un error definir dichas variables de entorno en `~/.bashrc`; en vez de ello deben definirse en `.bash_profile`, el cual es únicamente cargado por el shell de ingreso al sistema. See Section "Bash Startup Files" in *The GNU Bash Reference Manual*, para detalles sobre los archivos de inicio de Bash.

que no es el sistema Guix, es deseable prevenir acceso a `/usr/bin` y otros recursos del sistema desde el entorno de desarrollo. Por ejemplo, la siguiente orden lanza un REPL Guile en un "contenedor" donde únicamente el almacén y el directorio actual están montados:

```
guix environment --ad-hoc --container guile -- guile
```

**Nota:** La opción `--container` requiere Linux-libre 3.19 o posterior.

Otro caso de uso típico para los contenedores es la ejecución de aplicaciones sensibles como navegadores web. Para ejecutar Eolie, debemos exponer y compartir algunos archivos y directorios; incluimos `nss-certs` y exponemos `/etc/ssl/certs/` para la identificación HTTPS; por último preservamos la variable de entorno `DISPLAY` ya que las aplicaciones gráficas en el contenedor no se mostrarían sin ella.

```
guix environment --preserve='^DISPLAY$' --container --network \
  --expose=/etc/machine-id \
  --expose=/etc/ssl/certs/ \
  --share=$HOME/.local/share/eolie/=$HOME/.local/share/eolie/ \
  --ad-hoc eolie nss-certs dbus --  eolie
```

Las opciones disponibles se resumen a continuación.

`--check` Set up the environment and check whether the shell would clobber environment variables. See Section 7.1 [Invoking guix shell], page 79, for more info.

`--root=`*archivo*
`-r` *archivo*

Hace que *archivo* sea un enlace simbólico al perfil para este entorno, y lo registra como una raíz del recolector de basura.

Esto es útil si desea proteger su entorno de la recolección de basura, hacerlo "persistente".

Cuando se omite esta opción, el entorno se protege de la recolección de basura únicamente por la duración de la sesión `guix environment`. Esto significa que la siguiente vez que vuelva a crear el mismo entorno, puede tener que reconstruir o volver a descargar paquetes. See Section 5.6 [Invocación de guix gc], page 53, para más información sobre las raíces del recolector de basura.

`--expression=`*expr*
`-e` *expr* Crea un entorno para el paquete o lista de paquetes a los que evalúa *expr*.

Por ejemplo, ejecutando:

```
guix environment -e '(@ (gnu packages maths) petsc-openmpi)'
```

inicia un shell con el entorno para esta variante específica del paquete PETSc.

Ejecutar:

```
guix environment --ad-hoc -e '(@ (gnu) %base-packages)'
```

inicia un shell con todos los paquetes básicos del sistema disponibles.

Las órdenes previas usan únicamente la salida predeterminada de los paquetes dados. Para seleccionar otras salidas, tuplas de dos elementos pueden ser especificadas:

```
guix environment --ad-hoc -e '(list (@ (gnu packages bash) bash) "include")'
```

`--load=`*archivo*
`-l` *archivo*

Crea un entorno para el paquete o la lista de paquetes a la que el código en *archivo* evalúa.

Como un ejemplo, *archivo* puede contener una definición como esta (see Section 8.2 [Definición de paquetes], page 102):

```
(use-modules (guix)
             (gnu packages gdb)
             (gnu packages autotools)
             (gnu packages texinfo))

;; Augment the package definition of GDB with the build tools
;; needed when developing GDB (and which are not needed when
;; simply installing it.)
(package
  (inherit gdb)
  (native-inputs (modify-inputs (package-native-inputs gdb)
                   (prepend autoconf-2.69 automake texinfo))))
```

`--manifest=`*archivo*
`-m` *archivo*

Crea un entorno para los paquetes contenidos en el objeto manifest devuelto por el código Scheme en *file*. Esta opción se puede repetir varias veces, en cuyo caso los manifiestos se concatenan.

Esto es similar a la opción del mismo nombre en `guix package` (see [profile-manifest], page 40) y usa los mismos archivos de manifiesto.

See [shell-export-manifest], page 82, for information on how to "convert" command-line options into a manifest.

`--ad-hoc` Incluye todos los paquetes especificados en el entorno resultante, como si un paquete *ad hoc* hubiese sido definido con ellos como entradas. Esta opción es útil para la creación rápida un entorno sin tener que escribir una expresión de paquete que contenga las entradas deseadas.

Por ejemplo, la orden:

```
guix environment --ad-hoc guile guile-sdl -- guile
```

ejecuta `guile` en un entorno donde están disponibles Guile y Guile-SDL.

Fíjese que este ejemplo solicita implícitamente la salida predeterminada de `guile` y `guile-sdl`, pero es posible solicitar una salida específica—por ejemplo, `glib:bin` solicita la salida `bin` de `glib` (see Section 5.4 [Paquetes con múltiples salidas], page 51).

Esta opción puede componerse con el comportamiento predeterminado de `guix environment`. Los paquetes que aparecen antes de `--ad-hoc` se interpretan como paquetes cuyas dependencias se añadirán al entorno, el comportamiento predefinido. Los paquetes que aparecen después se interpretan como paquetes a añadir directamente al entorno.

`--profile=`*perfil*
`-p` *perfil* Create an environment containing the packages installed in *profile*. Use `guix package` (see Section 5.2 [Invocación de guix package], page 36) to create and manage profiles.

`--pure` Olvida las variables de entorno existentes cuando se construye un nuevo entorno, excepto aquellas especificadas con `--preserve` (véase más adelante). Esto tiene el efecto de crear un entorno en el que las rutas de búsqueda únicamente contienen las entradas del paquete.

`--preserve=`*regexp*
`-E` *regexp* Cuando se usa junto a `--pure`, preserva las variables de entorno que corresponden con *regexp*—en otras palabras, las pone en una lista de variables de entorno que deben preservarse. Esta opción puede repetirse varias veces.

```
guix environment --pure --preserve=^SLURM --ad-hoc openmpi ... \
  -- mpirun ...
```

Este ejemplo ejecuta `mpirun` en un contexto donde las únicas variables de entorno definidas son `PATH`, variables de entorno cuyo nombre empiece con '`SLURM`', así como las variables "preciosas" habituales (`HOME`, `USER`, etc.).

`--search-paths`
Muestra las definiciones de variables de entorno que componen el entorno.

`--system=`*sistema*
`-s` *sistema*
Intenta construir para *sistema*—por ejemplo, `i686-linux`.

`--container`
`-C` Ejecuta la *orden* en un contenedor aislado. El directorio actual fuera del contenedor es asociado al interior del contenedor. Adicionalmente, a menos que se fuerce con `--user`, un directorio de prueba de la usuaria se crea de forma que coincida con el directorio actual de la usuaria, y `/etc/passwd` se configura adecuadamente.

El proceso lanzado se ejecuta como el usuario actual fuera del contenedor. Dentro del contenedor, tiene el mismo UID y GID que el usuario actual, a menos que se proporcione `--user` (véase más adelante).

`--network`
`-N` Para contenedores, comparte el espacio de nombres de red con el sistema anfitrión. Los contenedores creados sin esta opción únicamente tienen acceso a la red local.

`--link-profile`
`-P` Para contenedores, enlaza el perfil del entorno a `~/.guix-profile` dentro del contenedor y asigna ese valor a `GUIX_ENVIRONMENT`. Es equivalente a que `~/.guix-profile` sea un enlace al perfil real dentro del contenedor. El enlace fallará e interrumpirá el entorno si el directorio ya existe, lo cual será probablemente el caso si `guix environment` se invocó en el directorio de la usuaria.

Determinados paquetes se configuran para buscar en `~/.guix-profile` archivos de configuración y datos;[4] `--link-profile` permite a estos programas operar de la manera esperada dentro del entorno.

`--user=`*usuaria*
`-u` *usuaria*

Para contenedores, usa el nombre de usuaria *usuaria* en vez de la actual. La entrada generada en `/etc/passwd` dentro del contenedor contendrá el nombre *usuaria*; su directorio será `/home/`*usuaria* y ningún dato GECOS de la usuaria se copiará. Más aún, el UID y GID dentro del contenedor son 1000. *usuaria* no debe existir en el sistema.

Adicionalmente, cualquier ruta compartida o expuesta (véanse `--share` y `--expose` respectivamente) cuyo destino esté dentro de la carpeta actual de la usuaria será reasociada en relación a `/home/`*usuaria*; esto incluye la relación automática del directorio de trabajo actual.

```
# expondrá las rutas /home/foo/ddt, /home/foo/prueba y /home/foo/objetivo
cd $HOME/ddt
guix environment --container --user=foo \
     --expose=$HOME/prueba \
     --expose=/tmp/objetivo=$HOME/objetivo
```

Mientras esto limita el escape de la identidad de la usuaria a través de las rutas de sus directorios y cada uno de los campos de usuaria, esto es únicamente un componente útil de una solución de privacidad/anonimato más amplia—no una solución completa.

`--no-cwd` El comportamiento predeterminado con contenedores es compartir el directorio de trabajo actual con el contenedor aislado e inmediatamente cambiar a dicho directorio dentro del contenedor. Si no se desea este comportamiento, `--no-cwd` indica que el directorio actual *no* se compartirá automáticamente y, en vez de cambiar a dicho directorio, se cambiará al directorio de la usuaria dentro del contenedor. Véase también `--user`.

`--expose=`*fuente*`[=`*destino*`]`
`--share=`*fuente*`[=`*destino*`]`

En contenedores, la `--expose` expone el sistema de archivos *fuente* del sistema anfitrión como un sistema de archivos de solo-lectura *destino* dentro del contenedor. `--share` de la misma manera expone el sistema de archivos con posibilidad de escritura. Si no se especifica *destino*, *fuente* se usa como el punto de montaje en el contenedor.

El ejemplo a continuación lanza una sesión interactiva de Guile en un contenedor donde el directorio principal de la usuaria es accesible en modo solo-lectura a través del directorio `/intercambio`:

```
guix environment --container --expose=$HOME=/intercambio --ad-hoc guile -- g
```

[4] Por ejemplo, el paquete `fontconfig` inspecciona `~/.guix-profile/share/fonts` en busca de nuevas tipografías.

`--emulate-fhs`
`-F`
: For containers, emulate a Filesystem Hierarchy Standard (FHS) configuration within the container, see the official specification (`https://refspecs.linuxfoundation.org/fhs.shtml`). As Guix deviates from the FHS specification, this option sets up the container to more closely mimic that of other GNU/Linux distributions. This is useful for reproducing other development environments, testing, and using programs which expect the FHS specification to be followed. With this option, the container will include a version of `glibc` which will read `/etc/ld.so.cache` within the container for the shared library cache (contrary to `glibc` in regular Guix usage) and set up the expected FHS directories: `/bin`, `/etc`, `/lib`, and `/usr` from the container's profile.

Además, `guix environment` acepta todas las opciones comunes de construcción que permite `guix build` (see Section 9.1.1 [Opciones comunes de construcción], page 181) así como las opciones de transformación de paquetes (see Section 9.1.2 [Opciones de transformación de paquetes], page 184).

## 7.3 Invocación de `guix pack`

De manera ocasional querrá dar software a gente que (¡todavía!) no tiene la suerte de usar Guix. Usted les diría que ejecuten `guix package -i` *algo*, pero eso no es posible en este caso. Aquí es donde viene `guix pack`.

> **Nota:** Si está buscando formas de intercambiar binarios entre máquinas que ya ejecutan Guix, see Section 9.13 [Invocación de guix copy], page 233, Section 9.11 [Invocación de guix publish], page 226, y Section 5.11 [Invocación de guix archive], page 66.

La orden `guix pack` crea un *paquete* reducido o *empaquetado de software*: crea un archivador tar u otro tipo que contiene los binarios del software en el que está interesada y todas sus dependencias. El archivo resultante puede ser usado en una máquina que no tiene Guix, y la gente puede ejecutar exactamente los mismos binarios que usted tiene con Guix. El paquete en sí es creado de forma reproducible bit-a-bit, para que cualquiera pueda verificar que realmente contiene los resultados de construcción que pretende distribuir.

Por ejemplo, para crear un empaquetado que contenga Guile, Emacs, Geiser y todas sus dependencias, puede ejecutar:

```
$ guix pack guile emacs emacs-geiser
...
/gnu/store/...-pack.tar.gz
```

El resultado aquí es un archivador tar que contiene un directorio de `/gnu/store` con todos los paquetes relevantes. El archivador resultante contiene un *perfil* con los tres paquetes de interés; el perfil es el mismo que se hubiera creado por `guix package -i`. Este es el mecanismo usado para crear el propio archivador de binarios separado de Guix (see Section 2.1 [Instalación binaria], page 4).

Las usuarias de este empaquetad tendrán que ejecutar `/gnu/store/...-profile/bin/guile` para ejecutar guile, lo que puede resultar inconveniente. Para evitarlo, puede crear, digamos, un enlace simbólico `/opt/gnu/bin` al perfil:

```
guix pack -S /opt/gnu/bin=bin guile emacs emacs-geiser
```

De este modo, las usuarias pueden escribir alegremente `/opt/gnu/bin/guile` y disfrutar.

¿Qué pasa se la receptora de su paquete no tiene privilegios de root en su máquina y por lo tanto no puede desempaquetarlo en la raíz del sistema de archivos? En ese caso, lo que usted desea es usar la opción `--relocatable` (véase a continuación). Esta opción produce *binarios reposicionables*, significando que pueden ser colocados en cualquier lugar de la jerarquía del sistema de archivos: en el ejemplo anterior, las usuarias pueden desempaquetar el archivador en su directorio de usuaria y ejecutar directamente `./opt/gnu/bin/guile`.

De manera alternativa, puede producir un empaquetado en el formato de imagen Docker usando la siguiente orden:

```
guix pack -f docker -S /bin=bin guile guile-readline
```

El resultado es un archivador "tar" que puede ser proporcionado a la orden `docker load`, seguida de `docker run`:

```
docker load < archivo
docker run -ti guile-guile-readline /bin/guile
```

where *file* is the image returned by `guix pack`, and `guile-guile-readline` is its "image tag". See the Docker documentation (`https://docs.docker.com/engine/reference/commandline/load/`) for more information.

Otra opción más es producir una imagen SquashFS con la siguiente orden:

```
guix pack -f squashfs bash guile emacs emacs-geiser
```

El resultado es una imagen de sistema de archivos SquashFS que puede ser o bien montada, o bien usada directamente como una imagen contenedora de sistemas de archivos con el entorno de ejecución de contenedores Singularity (`https://www.sylabs.io/docs/`), usando órdenes como `singularity shell` o `singularity exec`.

Varias opciones de la línea de órdenes le permiten personalizar su empaquetado:

`--format=`*formato*
`-f` *formato*

Produce un empaquetado en el *formato* específico.

Los formatos disponibles son:

`tarball` Es el formato predeterminado. Produce un archivador que contiene todos los binarios y enlaces simbólicos especificados.

`docker` This produces a tarball that follows the Docker Image Specification (`https://github.com/docker/docker/blob/master/image/spec/v1.2.md`). By default, the "repository name" as it appears in the output of the `docker images` command is computed from package names passed on the command line or in the manifest file. Alternatively, the "repository name" can also be configured via the `--image-tag` option. Refer to `--help-docker-format` for more information on such advanced options.

`squashfs` Produce una imagen SquashFS que contiene todos los binarios y enlaces simbólicos especificados, así como puntos de montaje vacíos para sistemas de archivos virtuales como procfs.

> **Nota:** Singularity *necesita* que proporcione `/bin/sh` en la imagen. Por esta razón, `guix pack -f squashfs`

siempre implica `-S /bin=bin`. Por tanto, su invocación de `guix pack` debe siempre comenzar de manera similar a esta:

```
guix pack -f squashfs bash ...
```

Si se olvida del paquete `bash` (o similar), `singularity run` y `singularity exec` fallarán con el mensaje "no existe el archivo o directorio", lo que no sirve de ayuda.

`deb` This produces a Debian archive (a package with the '`.deb`' file extension) containing all the specified binaries and symbolic links, that can be installed on top of any dpkg-based GNU(/Linux) distribution. Advanced options can be revealed via the `--help-deb-format` option. They allow embedding control files for more fine-grained control, such as activating specific triggers or providing a maintainer configure script to run arbitrary setup code upon installation.

```
guix pack -f deb -C xz -S /usr/bin/hello=bin/hello hello
```

**Nota:** Because archives produced with `guix pack` contain a collection of store items and because each `dpkg` package must not have conflicting files, in practice that means you likely won't be able to install more than one such archive on a given system. You can nonetheless pack as many Guix packages as you want in one such archive.

**Aviso:** `dpkg` will assume ownership of any files contained in the pack that it does *not* know about. It is unwise to install Guix-produced '`.deb`' files on a system where `/gnu/store` is shared by other software, such as a Guix installation or other, non-deb packs.

`rpm` This produces an RPM archive (a package with the '`.rpm`' file extension) containing all the specified binaries and symbolic links, that can be installed on top of any RPM-based GNU/Linux distribution. The RPM format embeds checksums for every file it contains, which the `rpm` command uses to validate the integrity of the archive.

Advanced RPM-related options are revealed via the `--help-rpm-format` option. These options allow embedding maintainer scripts that can run before or after the installation of the RPM archive, for example.

The RPM format supports relocatable packages via the `--prefix` option of the `rpm` command, which can be handy to install an RPM package to a specific prefix.

```
guix pack -f rpm -R -C xz -S /usr/bin/hello=bin/hello hello
sudo rpm --install --prefix=/opt /gnu/store/...-hello.rpm
```

**Nota:** Contrary to Debian packages, conflicting but *identical* files in RPM packages can be installed simultaneously, which means multiple `guix pack`-produced RPM packages can usually be installed side by side without any problem.

**Aviso:** `rpm` assumes ownership of any files contained in the pack, which means it will remove `/gnu/store` upon uninstalling a Guix-generated RPM package, unless the RPM package was installed with the `--prefix` option of the `rpm` command. It is unwise to install Guix-produced '`.rpm`' packages on a system where `/gnu/store` is shared by other software, such as a Guix installation or other, non-rpm packs.

`--relocatable`
`-R` Produce *binarios reposicionables*—es decir, binarios que se pueden encontrar en cualquier lugar de la jerarquía del sistema de archivos, y ejecutarse desde allí.

Cuando se proporciona una vez la opción, los binarios resultantes necesitan la implementación de *espacios de nombres de usuaria* del núcleo Linux; cuando se proporciona *dos veces*[5], los binarios reposicionables usan otras técnicas si los espacios de nombres de usuaria no están disponibles, y funcionan esencialmente en cualquier sitio—véase más adelante las implicaciones.

Por ejemplo, si crea un empaquetado que contiene Bash con:

```
guix pack -RR -S /mybin=bin bash
```

... puede copiar ese empaquetado a una máquina que no tiene Guix, y desde su directorio, como una usuaria normal, ejecutar:

```
tar xf pack.tar.gz
./mibin/sh
```

En ese shell, si escribe `ls /gnu/store`, notará que `/gnu/store` muestra y contiene todas las dependencias de `bash`, ¡incluso cuando la máquina no tiene el directorio `/gnu/store`! Esto es probablemente el modo más simple de desplegar software construido en Guix en una máquina no-Guix.

**Nota:** No obstante hay un punto a tener en cuenta: esta técnica descansa en la característica de *espacios de nombres de usuaria* del núcleo Linux, la cual permite a usuarias no privilegiadas montar o cambiar la raíz. Versiones antiguas de Linux no los implementan, y algunas distribuciones GNU/Linux los desactivan.

Para producir binarios reposicionables que funcionen incluso en ausencia de espacios de nombre de usuaria, proporcione `--relocatable` o `-R` *dos veces*. En ese caso, los binarios intentarán el uso de espacios de nombres de usuaria y usarán otro *motor de*

[5] Este es un truco para memorizarlo: `-RR`, que añade PRoot, puede pensarse como "Realmente Reposicionable". Curioso, ¿no es cierto?

*ejecución* si los espacios de nombres no están disponibles. Existe implementación para siguientes motores de ejecución:

`default` Intenta usar espacios de nombres de usuaria y usa PRoot en caso de no estar disponibles (véase a continuación).

`performance`
Intenta usar espacios de nombres de usuaria y usa Fakechroot en caso de no estar disponibles (véase a continuación).

`userns` Usa espacios de nombres de usuaria o aborta el programa si no están disponibles.

`proot` Ejecución a través de PRoot. El programa PRoot (`https://proot-me.github.io/`) proporciona el soporte necesario para la virtualización del sistema de archivos. Lo consigue mediante el uso de la llamada al sistema `ptrace` en el programa en ejecución. Esta aproximación tiene la ventaja de funcionar sin soporte especial en el núcleo, pero incurre en una sobrecarga en el tiempo de ejecución cada vez que se realiza una llamada al sistema.

`fakechroot`
Ejecución a través de Fakechroot. Fakechroot (`https://github.com/dex4er/fakechroot/`) virtualiza los accesos al sistema de archivos interceptando las llamadas a las funciones de la biblioteca de C como `open`, `stat`, `exec`, etcétera. Al contrario que PRoot, el proceso se somete únicamente a una pequeña sobrecarga. No obstante, no siempre funciona: algunos accesos realizados dentro de la biblioteca de C no se interceptan, ni tampoco los accesos al sistema de archivos a través de llamadas al sistema directas, lo que puede provocar un comportamiento impredecible.

Cuando ejecute un programa recubierto puede solicitar explícitamente uno de los motores de ejecución enumerados previamente proporcionando el valor adecuado a la variable de entorno `GUIX_EXECUTION_ENGINE`.

`--entry-point=`*orden*
Usa *orden* como el *punto de entrada* del empaquetado resultante, si el formato de empaquetado lo permite—actualmente `docker` y `squashfs` (Singularity) lo permiten. *orden* debe ser una ruta relativa al perfil contenido en el empaquetado.

El punto de entrada especifica la orden que herramientas como `docker run` o `singularity run` arrancan de manera automática de forma predeterminada. Por ejemplo, puede ejecutar:

```
guix pack -f docker --entry-point=bin/guile guile
```

El empaquetado resultante puede cargarse fácilmente y `docker run` sin parámetros adicionales lanzará `bin/guile`:

```
docker load -i pack.tar.gz
docker run image-id
```

`--entry-point-argument=`*command*
`-A` *command*

Use *command* as an argument to *entry point* of the resulting pack. This option is only valid in conjunction with `--entry-point` and can appear multiple times on the command line.

```
guix pack -f docker --entry-point=bin/guile --entry-point-argument="--help"
```

`--max-layers=n`

Specifies the maximum number of Docker image layers allowed when building an image.

```
guix pack -f docker --max-layers=100 guile
```

This option allows you to limit the number of layers in a Docker image. Docker images are comprised of multiple layers, and each layer adds to the overall size and complexity of the image. By setting a maximum number of layers, you can control the following effects:

- Disk Usage: Increasing the number of layers can help optimize the disk space required to store multiple images built with a similar package graph.
- Pulling: When transferring images between different nodes or systems, having more layers can reduce the time required to pull the image.

`--expression=`*expr*
`-e` *expr* Considera el paquete al que evalúa *expr*

Su propósito es idéntico a la opción del mismo nombre en `guix build` (see Section 9.1.3 [Opciones de construcción adicionales], page 190).

`--manifest=`*archivo*
`-m` *archivo*

Usa los paquetes contenidos en el objeto manifest devuelto por el código Scheme en *archivo*. Esta opción puede repetirse varias veces, en cuyo caso los manifiestos se concatenan.

Esto tiene un propósito similar al de la opción del mismo nombre en `guix package` (see [profile-manifest], page 40) y usa los mismos archivos de manifiesto. Esto le permite definir una colección de paquetes una vez y usarla tanto para crear perfiles como para crear archivos en máquinas que no tienen instalado Guix. Fíjese que puede especificar *o bien* un archivo de manifiesto *o bien* una lista de paquetes, pero no ambas.

See Section 8.4 [Writing Manifests], page 119, for information on how to write a manifest. See [shell-export-manifest], page 82, for information on how to "convert" command-line options into a manifest.

`--system=`*sistema*
`-s` *sistema*
: Intenta construir paquetes para *sistema*—por ejemplo, `x86_64-linux`—en vez del tipo de sistema de la máquina de construcción.

`--target=`*tripleta*
: Compilación cruzada para la *tripleta*, que debe ser una tripleta GNU válida, cómo `"aarch64-linux-gnu"` (see Section "Specifying target triplets" in *Autoconf*).

`--compression=`*herramienta*
`-C` *herramienta*
: Comprime el archivador resultante usando *herramienta*—un valor que puede ser `gzip`, `zstd`, `bzip2`, `xz`, `lzip` o `none` para no usar compresión.

`--symlink=`*spec*
`-S` *spec*
: Añade los enlaces simbólicos especificados por *spec* al empaquetado. Esta opción puede aparecer varias veces.

  La forma de *spec* es *`fuente=destino`*, donde *fuente* es el enlace simbólico que será creado y *destino* es el destino del enlace simbólico.

  Por ejemplo, `-S /opt/gnu/bin=bin` crea un enlace simbólico `/opt/gnu/bin` apuntando al subdirectorio `bin` del perfil.

`--save-provenance`
: Almacena la información de procedencia para paquetes proporcionados en la línea de órdenes. La información de procedencia incluye la URL y revisión de los canales en uso (see Chapter 6 [Canales], page 69).

  La información de procedencia se almacena en el archivo `/gnu/store/...-profile/manifest` dentro del empaquetado, junto a los metadatos habituales del paquete—el nombre y la versión de cada paquete, sus entradas propagadas, etcétera. Es información útil para la parte receptora del empaquetado, quien de ese modo conoce como se obtuvo (supuestamente) dicho empaquetado.

  Esta opción no se usa de manera predeterminada debido a que, como las marcas de tiempo, la información de procedencia no aportan nada al proceso de construcción. En otras palabras, hay una infinidad de URL de canales e identificadores de revisiones que pueden llevar al mismo empaquetado. Almacenar estos metadatos "silenciosos" en la salida puede potencialmente romper la propiedad de reproducibilidad bit a bit entre fuentes y binarios.

`--root=`*archivo*
`-r` *archivo*
: Hace que *archivo* sea un enlace simbólico al empaquetado resultante, y lo registra como una raíz del recolector de basura.

`--localstatedir`
`--profile-name=`*nombre*
: Incluye el "directorio de estado local", `/var/guix`, en el empaquetado resultante, y notablemente el perfil `/var/guix/profiles/per-user/root/`*`nombre`*—por defecto *nombre* es `guix-profile`, que corresponde con `~root/.guix-profile`.

`/var/guix` contiene la base de datos del almacén (see Section 8.9 [El almacén], page 157) así como las raíces del recolector de basura (see Section 5.6 [Invocación de guix gc], page 53). Proporcionarlo junto al empaquetado significa que el almacén está "completo" y Guix puede trabajar con él; no proporcionarlo significa que el almacén está "muerto": no se pueden añadir o borrar nuevos elementos después de la extracción del empaquetado.

Un caso de uso para esto es el archivador tar autocontenido de binarios de Guix (see Section 2.1 [Instalación binaria], page 4).

`--derivation`
`-d` Imprime el nombre de la derivación que construye el empaquetado.

`--bootstrap`
Usa los binarios del lanzamiento para construir el empaquetado. Esta opción es útil únicamente a las desarrolladoras de Guix.

Además, `guix pack` acepta todas las opciones comunes de construcción (see Section 9.1.1 [Opciones comunes de construcción], page 181) y todas las opciones de transformación de paquetes (see Section 9.1.2 [Opciones de transformación de paquetes], page 184).

## 7.4 La cadena de herramientas de GCC

Si necesita una cadena de herramientas de desarrollo completa para compilar y enlazar código fuente C o C++, use el paquete `gcc-toolchain`. Este paquete proporciona una cadena de herramientas GCC para desarrollo C/C++, incluyendo el propio GCC, la biblioteca de C GNU (cabeceras y binarios, más símbolos de depuración de la salida `debug`), Binutils y un recubrimiento del enlazador.

El propósito del recubrimiento es inspeccionar las opciones `-L` y `-l` proporcionadas al enlazador, y los correspondientes parámetros `-rpath`, y llamar al enlazador real con este nuevo conjunto de parámetros. Puede instruir al recubrimiento para rechazar el enlace contra bibliotecas que no se encuentren en el almacén proporcionando el valor `no` a la variable de entorno `GUIX_LD_WRAPPER_ALLOW_IMPURITIES`.

El paquete `gfortran-toolchain` proporciona una cadena de herramientas de desarrollo completa de GCC para desarrollo en Fortran. Para otros lenguajes por favor use '`guix search gcc toolchain`' (see [Invoking guix package], page 42).

## 7.5 Invocación de `guix git authenticate`

La orden `guix git authenticate` verifica una revisión de Git siguiendo las mismas reglas que con los canales (see [channel-authentication], page 72). Es decir, empezando en una revisión dada, se asegura que todas las revisiones posteriores están firmadas por una clave OpenPGP cuya huella aparece en el archivo `.guix-authorizations` de su revisión o revisiones antecesoras.

Encontrará útil esta orden si mantiene un canal. Pero de hecho, este sistema de verificación es útil en un contexto más amplio, por lo que quizá quiera usarlo para repositorios de Git que no estén relacionados con Guix.

La sintaxis general es:

```
guix git authenticate revisión firma [opciones...]
```

By default, this command authenticates the Git checkout in the current directory; it outputs nothing and exits with exit code zero on success and non-zero on failure. *commit* above denotes the first commit where authentication takes place, and *signer* is the OpenPGP fingerprint of public key used to sign *commit*. Together, they form a *channel introduction* (see [channel-authentication], page 72). On your first successful run, the introduction is recorded in the `.git/config` file of your checkout, allowing you to omit them from subsequent invocations:

```
guix git authenticate [options...]
```

Should you have branches that require different introductions, you can specify them directly in `.git/config`. For example, if the branch called `personal-fork` has a different introduction than other branches, you can extend `.git/config` along these lines:

```
[guix "authentication-personal-fork"]
introduction-commit = cabba936fd807b096b48283debdcddccfea3900d
introduction-signer = C0FF EECA BBA9 E6A8 0D1D  E643 A2A0 6DF2 A33A 54FA
keyring = keyring
```

The first run also attempts to install pre-push and post-merge hooks, such that `guix git authenticate` is invoked as soon as you run `git push`, `git pull`, and related commands; it does not overwrite preexisting hooks though.

The command-line options described below allow you to fine-tune the process.

`--repository=`*directorio*
`-r` *directorio*
: Usa el repositorio Git en *directorio* en vez del directorio actual.

`--keyring=`*referencia*
`-k` *referencia*
: Carga el anillo de claves desde *referencia*, la rama de referencia como por ejemplo `origin/keyring` o `mi-anillo-de-claves`. La rama debe contener las claves públicas de OpenPGP en archivos `.key`, binarios o con "armadura ASCII'. De manera predeterminada el anillo de claves se carga de la rama con nombre `keyring`.

`--end=`*commit*
: Authenticate revisions up to *commit*.

`--stats`
: Muestra las estadísticas de firmas de revisiones tras finalizar.

`--cache-key=`*clave*
: Las revisiones verificadas previamente se almacenan en un archivo bajo `~/.cache/guix/authentication`. Esta opción fuerza el almacenamiento en el archivo *clave* de dicho directorio.

`--historical-authorizations=`*archivo*
: De manera predeterminada, cualquier revisión cuyo antecesor o antecesores carezcan del archivo `.guix-authorizations` no se considera auténtica. En contraste, esta opción considera las autorizaciones en *archivo* para cualquier revisión que carezca de `.guix-authorizations`. El formato de *archivo* es el mismo que el de `.guix-authorizations` (see [channel-authorizations], page 75).

# 8 Interfaz programática

GNU Guix proporciona viarias interfaces programáticas Scheme (APIs) para definir, construir y consultar paquetes. La primera interfaz permite a las usuarias escribir definiciones de paquetes a alto nivel. Estas definiciones referencian conceptos familiares de empaquetamiento, como el nombre y la versión de un paquete, su sistema de construcción y sus dependencias. Estas definiciones se pueden convertir en acciones concretas de construcción.

Las acciones de construcción son realizadas por el daemon Guix, en delegación de las usuarias. En una configuración estándar, el daemon tiene acceso de escritura al almacén—el directorio `/gnu/store`—mientras que las usuarias no. En la configuración recomendada el daemon también realiza las construcciones en chroots, bajo usuarias específicas de construcción, para minimizar la interferencia con el resto del sistema.

Las APIs de nivel más bajo están disponibles para interactuar con el daemon y el almacén. Para instruir al daemon para realizar una acción de construcción, las usuarias realmente proporcionan una *derivación*. Una derivación es una representación de bajo nivel de las acciones de construcción a tomar, y el entorno en el que deberían suceder—las derivaciones son a las definiciones de paquetes lo que es el ensamblador a los programas en C. El término "derivación" viene del hecho de que los resultados de la construcción *derivan* de ellas.

This chapter describes all these APIs in turn, starting from high-level package definitions. See Section 22.7 [Source Tree Structure], page 744, for a more general overview of the source code.

## 8.1 Módulos de paquetes

Desde un punto de vista programático, las definiciones de paquetes de la distribución GNU se proporcionan por módulos Guile en el espacio de nombres `(gnu packages ...)`[1] (see Section "Modules" in *GNU Guile Reference Manual*). Por ejemplo, el módulo `(gnu packages emacs)` exporta una variable con nombre `emacs`, que está asociada a un objeto `<package>` (see Section 8.2 [Definición de paquetes], page 102).

El espacio de nombres de módulos `(gnu packages ...)` se recorre automáticamente en busca de paquetes en las herramientas de línea de ordenes. Por ejemplo, cuando se ejecuta `guix install emacs`, todos los módulos `(gnu packages ...)` son procesados hasta encontrar uno que exporte un objeto de paquete cuyo nombre sea `emacs`. Esta búsqueda de paquetes se implementa en el módulo `(gnu packages)`.

Las usuarias pueden almacenar definiciones de paquetes en módulos con nombres diferentes—por ejemplo, `(mis-paquetes emacs)`[2]. Existen dos maneras de hacer visibles estas definiciones de paquetes a las interfaces de usuaria:

---

[1] Fíjese que los paquetes bajo el espacio de nombres de módulo `(gnu packages ...)` no son necesariamente "paquetes GNU". Este esquema de nombrado de módulos sigue la convención habitual de Guile para el nombrado de módulos: `gnu` significa que estos módulos se distribuyen como parte del sistema GNU, y `packages` identifica módulos que definen paquetes.

[2] Fíjese que el nombre de archivo y el nombre de módulo deben coincidir. Por ejemplo, el módulo `(mis-paquetes emacs)` debe almacenarse en el archivo `mis-paquetes/emacs.scm` en relación con la ruta de carga especificada con `--load-path` o `GUIX_PACKAGE_PATH`. See Section "Modules and the File System" in *GNU Guile Reference Manual*, para obtener detalles.

1. Mediante la adición del directorio que contiene sus módulos de paquetes a la ruta de búsqueda con la opción `-L` de `guix package` y otras órdenes (see Section 9.1.1 [Opciones comunes de construcción], page 181), o usando la variable de entorno `GUIX_PACKAGE_PATH` descrita a continuación.
2. Mediante la definición de un *canal* y la configuración de `guix pull` de manera que se actualice desde él. Un canal es esencialmente un repositorio Git que contiene módulos de paquetes. See Chapter 6 [Canales], page 69, para más información sobre cómo definir y usar canales.

`GUIX_PACKAGE_PATH` funciona de forma similar a otras variables de rutas de búsqueda:

`GUIX_PACKAGE_PATH` [Variable de entorno]
Es una lista separada por dos puntos de directorios en los que se buscarán módulos de paquetes adicionales. Los directorios enumerados en esta variable tienen preferencia sobre los propios módulos de la distribución.

La distribución es *auto-contenida* y completamente *basada en el lanzamiento inicial*: cada paquete se construye basado únicamente en otros paquetes de la distribución. La raíz de este grafo de dependencias es un pequeño conjunto de *binarios del lanzamiento inicial*, proporcionados por el módulo `(gnu packages bootstrap)`. Para más información sobre el lanzamiento inicial, see Chapter 20 [Lanzamiento inicial], page 725.

## 8.2 Definición de paquetes

La interfaz de alto nivel de las definiciones de paquetes está implementada en los módulos `(guix packages)` y `(guix build-system)`. Como un ejemplo, la definición de paquete, o *receta*, para el paquete GNU Hello es como sigue:

```
(define-module (gnu packages hello)
  #:use-module (guix packages)
  #:use-module (guix download)
  #:use-module (guix build-system gnu)
  #:use-module (guix licenses)
  #:use-module (gnu packages gawk))

(define-public hello
  (package
    (name "hello")
    (version "2.10")
    (source (origin
              (method url-fetch)
              (uri (string-append "mirror://gnu/hello/hello-" version
                                  ".tar.gz"))
              (sha256
               (base32
                "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i"))))
    (build-system gnu-build-system)
    (arguments '(#:configure-flags '("--enable-silent-rules")))
    (inputs (list gawk))
```

```
(synopsis "Hello, GNU world: An example GNU package")
(description "Guess what GNU Hello prints!")
(home-page "https://www.gnu.org/software/hello/")
(license gpl3+)))
```

Sin ser una experta en Scheme—pero conociendo un poco de inglés—, la lectora puede haber supuesto el significado de varios campos aquí. Esta expresión asocia la variable `hello` al objeto `<package>`, que esencialmente es un registro (see Section "SRFI-9" in *GNU Guile Reference Manual*). Este objeto de paquete puede ser inspeccionado usando los procedimientos encontrados en el módulo `(guix packages)`; por ejemplo, `(package-name hello)` devuelve—¡sorpresa!—`"hello"`.

Con suerte, puede que sea capaz de importar parte o toda la definición del paquete de su interés de otro repositorio, usando la orden `guix import` (see Section 9.5 [Invocación de guix import], page 198).

En el ejemplo previo, `hello` se define en un módulo para ella, `(gnu packages hello)`. Técnicamente, esto no es estrictamente necesario, pero es conveniente hacerlo: todos los paquetes definidos en módulos bajo `(gnu packages ...)` se reconocen automáticamente en las herramientas de línea de órdenes (see Section 8.1 [Módulos de paquetes], page 101).

Hay unos pocos puntos que merece la pena destacar de la definición de paquete previa:

- El campo `source` del paquete es un objeto `<origin>` (see Section 8.2.2 [Referencia de origin], page 110, para la referencia completa). Aquí se usa el método `url-fetch` de `(guix download)`, lo que significa que la fuente es un archivo a descargar por FTP o HTTP.

  El prefijo `mirror://gnu` instruye a `url-fetch` para usar uno de los espejos GNU definidos en `(guix download)`.

  El campo `sha256` especifica el hash SHA256 esperado del archivo descargado. Es obligatorio, y permite a Guix comprobar la integridad del archivo. La forma `(base32 ...)` introduce la representación base32 del hash. Puede obtener esta información con `guix download` (see Section 9.3 [Invocación de guix download], page 196) y `guix hash` (see Section 9.4 [Invocación de guix hash], page 197).

  Cuando sea necesario, la forma `origin` también puede tener un campo `patches` con la lista de parches a ser aplicados, y un campo `snippet` con una expresión Scheme para modificar el código fuente.

- El campo `build-system` especifica el procedimiento de construcción del paquete (see Section 8.5 [Sistemas de construcción], page 123). Aquí, `gnu-build-system` representa el familiar sistema de construcción GNU, donde los paquetes pueden configurarse, construirse e instalarse con la secuencia de ordenes habitual `./configure && make && make check && make install`.

  Cuando comience a empaquetar software no trivial puede que necesite herramientas para manipular estas fases de construcción, manipular archivos, etcétera. See Section 8.7 [Utilidades de construcción], page 147, para obtener más información sobre este tema.

- El campo `arguments` especifica las opciones para el sistema de construcción (see Section 8.5 [Sistemas de construcción], page 123). Aquí son interpretadas por `gnu-build-system` como una petición de ejecutar `configure` con la opción `--enable-silent-rules`.

What about these quote (`'`) characters? They are Scheme syntax to introduce a literal list; `'` is synonymous with `quote`. Sometimes you'll also see `` ` `` (a backquote, synonymous with `quasiquote`) and `,` (a comma, synonymous with `unquote`). See Section "Expression Syntax" in *GNU Guile Reference Manual*, for details. Here the value of the `arguments` field is a list of arguments passed to the build system down the road, as with `apply` (see Section "Fly Evaluation" in *GNU Guile Reference Manual*).

La secuencia almohadilla-dos puntos (`#:`) define una *palabra clave* Scheme (see Section "Keywords" in *GNU Guile Reference Manual*), y `#:configure-flags` es una palabra clave usada para pasar un parámetro nominal al sistema de construcción (see Section "Coding With Keywords" in *GNU Guile Reference Manual*).

- The `inputs` field specifies inputs to the build process—i.e., build-time or run-time dependencies of the package. Here, we add an input, a reference to the `gawk` variable; `gawk` is itself bound to a `<package>` object.

  Fíjese que no hace falta que GCC, Coreutils, Bash y otras herramientas esenciales se especifiquen como entradas aquí. En vez de eso, `gnu-build-system` se hace cargo de asegurar que están presentes (see Section 8.5 [Sistemas de construcción], page 123).

  No obstante, cualquier otra dependencia debe ser especificada en el campo `inputs`. Las dependencias no especificadas aquí simplemente no estarán disponibles para el proceso de construcción, provocando posiblemente un fallo de construcción.

See Section 8.2.1 [Referencia de package], page 105, para una descripción completa de los campos posibles.

**Más allá:**

Intimidated by the Scheme language or curious about it? The Cookbook has a short section to get started that recaps some of the things shown above and explains the fundamentals. See Section "A Scheme Crash Course" in *GNU Guix Cookbook*, for more information.

Una vez la definición de paquete esté en su lugar, el paquete puede ser construido realmente usando la herramienta de línea de órdenes `guix build` (see Section 9.1 [Invocación de guix build], page 181), pudiendo resolver cualquier fallo de construcción que encuentre (see Section 9.1.4 [Depuración de fallos de construcción], page 194). Puede volver a la definición del paquete fácilmente usando la orden `guix edit` (see Section 9.2 [Invocación de guix edit], page 196). See Section 22.8 [Pautas de empaquetamiento], page 748, para más información sobre cómo probar definiciones de paquetes, y Section 9.8 [Invocación de guix lint], page 216, para información sobre cómo comprobar la consistencia del estilo de una definición.

Por último, see Chapter 6 [Canales], page 69, para información sobre cómo extender la distribución añadiendo sus propias definiciones de paquetes en un "canal".

Finalmente, la actualización de la definición con una nueva versión oficial puede ser automatizada parcialmente por la orden `guix refresh` (see Section 9.6 [Invocación de guix refresh], page 207).

Tras el telón, una derivación correspondiente al objeto `<package>` se calcula primero mediante el procedimiento `package-derivation`. Esta derivación se almacena en un archivo `.drv` bajo `/gnu/store`. Las acciones de construcción que prescribe pueden entonces llevarse a cabo usando el procedimiento `build-derivations` (see Section 8.9 [El almacén], page 157).

`package-derivation` *almacén paquete* [*sistema*] [Procedimiento]
Devuelve el objeto `<derivation>` del *paquete* pra el *sistema* (see Section 8.10 [Derivaciones], page 159).

*paquete* debe ser un objeto `<package>` válido, y *sistema* debe ser una cadena que denote el tipo de sistema objetivo—por ejemplo, `"x86_64-linux"` para un sistema GNU x86_64 basado en Linux. *almacén* debe ser una conexión al daemon, que opera en el almacén (see Section 8.9 [El almacén], page 157).

De manera similar, es posible calcular una derivación que construye de forma cruzada un paquete para otro sistema:

`package-derivation` *almacén paquete* [*sistema*] [Procedimiento]
Devuelve el objeto `<derivation>` de *paquete* compilado de forma cruzada desde *sistema* a *plataforma*.

*plataforma* debe ser una tripleta GNU válida que identifique al hardware y el sistema operativo deseado, como por ejemplo `"aarch64-linux-gnu"` (see Section "Specifying Target Triplets" in *Autoconf*).

Una vez tenga sus definiciones de paquetes puede definir facilmente *variantes* de dichos paquetes. See Section 8.3 [Definición de variantes de paquetes], page 114, para obtener más información sobre ello.

### 8.2.1 Referencia de `package`

Esta sección resume todas las opciones disponibles en declaraciones `package` (see Section 8.2 [Definición de paquetes], page 102).

`package` [Tipo de datos]
Este es el tipo de datos que representa la receta de un paquete.

`name` El nombre del paquete, como una cadena.

`version` The version of the package, as a string. See Section 22.8.3 [Versiones numéricas], page 750, for guidelines.

`source` Un objeto que determina cómo se debería obtener el código fuente del paquete. La mayor parte del tiempo, es un objeto `origin`, que denota un archivo obtenido de Internet (see Section 8.2.2 [Referencia de origin], page 110). También puede ser cualquier otro objeto "tipo-archivo" como `local-file`, que denota un archivo del sistema local de archivos (see Section 8.12 [Expresiones-G], page 167).

`build-system`
El sistema de construcción que debe ser usado para construir el paquete (see Section 8.5 [Sistemas de construcción], page 123).

`arguments` (predeterminados: `'()`)
The arguments that should be passed to the build system (see Section 8.5 [Sistemas de construcción], page 123). This is a list, typically containing sequential keyword-value pairs, as in this example:

```
(package
```

```
(name "example")
;; several fields omitted
(arguments
  (list #:tests? #f                     ;skip tests
        #:make-flags #~'("VERBOSE=1")   ;pass flags to 'make'
        #:configure-flags #~'("--enable-frobbing"))))
```

The exact set of supported keywords depends on the build system (see Section 8.5 [Sistemas de construcción], page 123), but you will find that almost all of them honor `#:configure-flags`, `#:make-flags`, `#:tests?`, and `#:phases`. The `#:phases` keyword in particular lets you modify the set of build phases for your package (see Section 8.6 [Fases de construcción], page 143).

The REPL has dedicated commands to interactively inspect values of some of these arguments, as a convenient debugging aid (see Section 8.14 [Using Guix Interactively], page 178).

**Compatibility Note:** Until version 1.3.0, the `arguments` field would typically use `quote` (`'`) or `quasiquote` (`` ` ``) and no G-expressions, like so:

```
(package
  ;; several fields omitted
  (arguments   ;old-style quoted arguments
   '(#:tests? #f
     #:configure-flags '("--enable-frobbing"))))
```

To convert from that style to the one shown above, you can run `guix style -S arguments` *package* (see Section 9.7 [Invoking guix style], page 214).

`inputs` (predeterminadas: `'()`)
`native-inputs` (predeterminadas: `'()`)
`propagated-inputs` (predeterminadas: `'()`)

These fields list dependencies of the package. Each element of these lists is either a package, origin, or other "file-like object" (see Section 8.12 [Expresiones-G], page 167); to specify the output of that file-like object that should be used, pass a two-element list where the second element is the output (see Section 5.4 [Paquetes con múltiples salidas], page 51, for more on package outputs). For example, the list below specifies three inputs:

```
(list libffi libunistring
      `(,glib "bin"))      ;the "bin" output of GLib
```

In the example above, the `"out"` output of `libffi` and `libunistring` is used.

**Compatibility Note:** Until version 1.3.0, input lists were a list of tuples, where each tuple has a label for the input (a string) as its first element, a package, origin, or derivation as its second element, and optionally the name of the output thereof that should be used, which defaults to `"out"`. For

example, the list below is equivalent to the one above, but using the *old input style*:

```
;; Old input style (deprecated).
`(("libffi" ,libffi)
  ("libunistring" ,libunistring)
  ("glib:bin" ,glib "bin"))  ;the "bin" output of GLib
```

This style is now deprecated; it is still supported but support will be removed in a future version. It should not be used for new package definitions. See Section 9.7 [Invoking guix style], page 214, on how to migrate to the new style.

La distinción entre `native-inputs` y `inputs` es necesaria cuando se considera la compilación cruzada. Cuando se compila desde una arquitectura distinta, las dependencias enumeradas en `inputs` son construidas para la arquitectura *objetivo*; de modo contrario, las dependencias enumeradas en `native-inputs` se construyen para la arquitectura de la máquina de *construcción*.

`native-inputs` se usa típicamente para enumerar herramientas necesarias en tiempo de construcción, pero no en tiempo de ejecución, como Autoconf, Automake, pkg-config, Gettext o Bison. `guix lint` puede informar de probables errores en este área (see Section 9.8 [Invocación de guix lint], page 216).

Por último, `propagated-inputs` es similar a `inputs`, pero los paquetes especificados se instalarán automáticamente a los perfiles (see Section 5.1 [Características], page 35) junto al paquete al que pertenecen (see [package-cmd-propagated-inputs], page 38, para información sobre cómo `guix package` gestiona las entradas propagadas).

Por ejemplo esto es necesario cuando se empaqueta una biblioteca C/C++ que necesita cabeceras de otra biblioteca para compilar, o cuando un archivo pkg-config se refiere a otro a través de su campo `Requires`.

Otro ejemplo donde `propagated-inputs` es útil es en lenguajes que carecen de la facilidad de almacenar la ruta de búsqueda de tiempo de ejecución de la misma manera que el campo `RUNPATH` de los archivos ELF; esto incluye Guile, Python, Perl y más. Cuando se empaquetan bibliotecas escritas en estos lenguajes, enumerar en `propagated-inputs` en vez de en `inputs` las dependencias de tiempo de ejecución permite asegurarse de encontrar el código de las bibliotecas de las que dependan en tiempo de ejecución.

`outputs` (predeterminada: `'("out")`)
: La lista de nombres de salidas del paquete. See Section 5.4 [Paquetes con múltiples salidas], page 51, para usos típicos de salidas adicionales.

`native-search-paths` (predeterminadas: `'()`)
`search-paths` (predeterminadas: `'()`)
: A list of `search-path-specification` objects describing search-path environment variables honored by the package. See Section 8.8 [Search Paths], page 154, for more on search path specifications.

As for inputs, the distinction between `native-search-paths` and `search-paths` only matters when cross-compiling. In a cross-compilation context, `native-search-paths` applies exclusively to native inputs whereas `search-paths` applies only to host inputs.

Packages such as cross-compilers care about target inputs—for instance, our (modified) GCC cross-compiler has `CROSS_C_INCLUDE_PATH` in `search-paths`, which allows it to pick `.h` files for the target system and *not* those of native inputs. For the majority of packages though, only `native-search-paths` makes sense.

`replacement` (predeterminado: `1.0`)
: Esto debe ser o bien `#f` o bien un objeto package que será usado como *reemplazo* para ete paquete. See Chapter 19 [Actualizaciones de seguridad], page 723, para más detalles.

`synopsis` Una descripción en una línea del paquete.

`description`
: A more elaborate description of the package, as a string in Texinfo syntax.

`license` La licencia del paquete; un valor de `(guix licenses)`, o una lista de dichos valores.

`home-page`
: La URL de la página principal del paquete, como una cadena.

`supported-systems` (predeterminados: `%supported-systems`)
: La lista de sistemas en los que se mantiene el paquete, como cadenas de la forma `arquitectura-núcleo`, por ejemplo `"x86_64-linux"`.

`location` (predeterminada: la localización de los fuentes de la forma `package`)
: La localización de las fuentes del paquete. Es útil forzar su valor cuando se hereda de otro paquete, en cuyo caso este campo no se corrige automáticamente.

`this-package` [Macro]

Cuando se usa en el *ámbito léxico* de la definición de un paquete, este identificador resuelve al paquete que se está definiendo.

El ejemplo previo muestra cómo añadir un paquete como su propia entrada nativa cuando se compila de forma cruzada:

```
(package
  (name "guile")
  ;; ...

  ;; When cross-compiled, Guile, for example, depends on
  ;; a native version of itself.  Add it here.
  (native-inputs (if (%current-target-system)
                     (list this-package)
                     '())))
```

Es un error hacer referencia a `this-package` fuera de la definición de un paquete.

The following helper procedures are provided to help deal with package inputs.

`lookup-package-input` *package name* [Procedure]
`lookup-package-native-input` *package name* [Procedure]
`lookup-package-propagated-input` *package name* [Procedure]
`lookup-package-direct-input` *package name* [Procedure]

Look up *name* among *package*'s inputs (or native, propagated, or direct inputs). Return it if found, `#f` otherwise.

*name* is the name of a package depended on. Here's how you might use it:

```
(use-modules (guix packages) (gnu packages base))

(lookup-package-direct-input coreutils "gmp")
⇒ #<package gmp@6.2.1 ...>
```

In this example we obtain the `gmp` package that is among the direct inputs of `coreutils`.

Sometimes you will want to obtain the list of inputs needed to *develop* a package—all the inputs that are visible when the package is compiled. This is what the `package-development-inputs` procedure returns.

`package-development-inputs` *package* [*system*] [*#:target #f*] [Procedure]

Return the list of inputs required by *package* for development purposes on *system*. When *target* is true, return the inputs needed to cross-compile *package* from *system* to *target*, where *target* is a triplet such as `"aarch64-linux-gnu"`.

Note that the result includes both explicit inputs and implicit inputs—inputs automatically added by the build system (see Section 8.5 [Sistemas de construcción], page 123). Let us take the `hello` package to illustrate that:

```
(use-modules (gnu packages base) (guix packages))

hello
⇒ #<package hello@2.10 gnu/packages/base.scm:79 7f585d4f6790>

(package-direct-inputs hello)
⇒ ()

(package-development-inputs hello)
⇒ (("source" ...) ("tar" #<package tar@1.32 ...>) ...)
```

In this example, `package-direct-inputs` returns the empty list, because `hello` has zero explicit dependencies. Conversely, `package-development-inputs` includes inputs implicitly added by `gnu-build-system` that are required to build `hello`: tar, gzip, GCC, libc, Bash, and more. To visualize it, `guix graph hello` would show you explicit inputs, whereas `guix graph -t bag hello` would include implicit inputs (see Section 9.10 [Invocación de guix graph], page 221).

Debido a que los paquetes son objetos Scheme normales que capturan un grafo de dependencias completo y se asocian a procedimientos de construcción, habitualmente es útil escribir procedimientos que toman un paquete y devuelven una versión modificada de acuerdo a ciertos parámetros. A continuación se muestran algunos ejemplos:

`package-with-c-toolchain` *paquete cadena* [Procedimiento]

Devuelve una variación de *paquete* que usa *cadena* en vez de la cadena de herramientas de construcción de C/C++ de GNU predeterminada. *cadena* debe ser una lista de entradas (tuplas etiqueta/paquete) que proporcionen una funcionalidad equivalente a la del paquete `gcc-toolchain`.

El siguiente ejemplo devuelve una variación del paquete `hello` que se ha construido con GCC 10.x y el resto de la cadena de herramientas de construcción de GNU (Binutils y la biblioteca de C de GNU) en vez de la cadena de construcción predeterminada:

```
(let ((cadena (specification->package "gcc-toolchain@10")))
  ;; El nombre de la entrada debe ser "toolchain".
  (package-with-c-toolchain hello `(("toolchain" ,cadena))))
```

La cadena de herramientas de construcción es parte de las *entradas impícitas* de los paquetes—habitualmente no se enumera como parte de los distintos campos de entrada ("inputs") sino que el sistema de construcción es quién la incorpora. Por lo tanto este procedimiento funciona cambiando el sistema de construcción del *paquete* de modo que se incorpora *cadena* en vez de los valores predeterminados. Section 8.5 [Sistemas de construcción], page 123, para obtener más información sobre sistemas de construcción.

### 8.2.2 Referencia de `origin`

Esta sección documenta los *orígenes*. Una declaración de origen (`origin`) especifica datos que se deben "producir"—descargandose, habitualmente—y el hash de su contenido se conoce de antemano. Los origenes se usan habitualmente para reprensentar el código fuente de los paquetes (see Section 8.2 [Definición de paquetes], page 102). Por esta razón la forma sintáctica `origin` le permite declarar tanto parches para aplicar al código fuente original como fragmentos de código que para su modificación.

`origin` [Tipo de datos]

Este es el tipo de datos que representa un origen de código fuente.

`uri` Un objeto que contiene el URI de las fuentes. El tipo de objeto depende del valor de `method` (véase a continuación). Por ejemplo, cuando se usa el método *url-fetch* de `(guix download)`, los valores adecuados para `uri` son: una cadena que contiene una URL, o una lista de cadenas.

`method` A monadic procedure that handles the given URI. The procedure must accept at least three arguments: the value of the `uri` field and the hash algorithm and hash value specified by the `hash` field. It must return a store item or a derivation in the store monad (see Section 8.11 [La mónada del almacén], page 162); most methods return a fixed-output derivation (see Section 8.10 [Derivaciones], page 159).

Los métodos habitualmente usados incluyen `url-fetch`, que obtiene datos a partir de una URL, y `git-fetch`, que obtiene datos de un repositorio Git (véase a continuación).

`sha256` Un vector de bytes que contiene el hash SHA-256 de las fuentes. Es equivalente a proporcionar un objeto SHA256 `content-hash` en el campo `hash` descrito a continuación.

`hash` El objeto `content-hash` de las fuentes—véase a continuación cómo usar `content-hash`.

Puede obtener esta información usando `guix download` (see Section 9.3 [Invocación de guix download], page 196) o `guix hash` (see Section 9.4 [Invocación de guix hash], page 197).

`file-name` (predeterminado: `#f`)
El nombre de archivo bajo el que el código fuente se almacenará. Cuando este es `#f`, un valor predeterminado sensato se usará en la mayor parte de casos. En caso de que las fuentes se obtengan de una URL, el nombre de archivo de la URL se usará. Para copias de trabajo de sistemas de control de versiones, se recomienda proporcionar el nombre de archivo explícitamente ya que el predeterminado no es muy descriptivo.

`patches` (predeterminados: `'()`)
Una lista de nombres de archivos, orígenes u objetos tipo-archivo (see Section 8.12 [Expresiones-G], page 167) apuntando a parches que deben ser aplicados a las fuentes.

La lista de parches debe ser incondicional. En particular, no puede depender del valor de `%current-system` o `%current-target-system`.

`snippet` (predeterminado: `#f`)
Una expresión-G (see Section 8.12 [Expresiones-G], page 167) o expresión-S que se ejecutará en el directorio de fuentes. Esta es una forma conveniente de modificar el software, a veces más que un parche.

`patch-flags` (predeterminadas: `'("-p1")`)
Una lista de opciones de línea de órdenes que deberían ser pasadas a la orden `patch`.

`patch-inputs` (predeterminada: `#f`)
Paquetes o derivaciones de entrada al proceso de aplicación de los parches. Cuando es `#f`, se proporciona el conjunto habitual de entradas necesarias para la aplicación de parches, como GNU Patch.

`modules` (predeterminados: `'()`)
Una lista de módulos Guile que debe ser cargada durante el proceso de aplicación de parches y mientras se ejecuta el código del campo `snippet`.

`patch-guile` (predeterminado: `#f`)
El paquete Guile que debe ser usado durante la aplicación de parches. Cuando es `#f` se usa un valor predeterminado.

`content-hash` *valor* [*algoritmo*] [Tipo de datos]
Construye un objeto del hash del contenido para el *algoritmo* proporcionado, y con *valor* como su valor. Cuando se omite *algoritmo* se asume que es `sha256`.

*valor* puede ser una cadena literal, en cuyo caso se decodifica en base32, o un vector de bytes.

Las siguientes opciones equivalentes entre sí:

```
(content-hash "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj")
```

```
(content-hash "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj"
              sha256)
(content-hash (base32
               "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj"))
(content-hash (base64 "kkb+RPaP7uyMZmu4eXPVkM4BN8yhRd8BTHLslb6f/Rc=")
              sha256)
```

Técnicamente, `content-hash` se implementa actualmente con un macro. Realiza comprobaciones de seguridad en tiempo de expansión, cuando es posible, como asegurarse de que *valor* tiene el tamaño adecuado para *algoritmo*.

Como se ha visto previamente, la forma exacta en la que un origen hace referencia a los datos se determina por su campo `method`. El módulo `(guix download)` proporciona el método más común, `url-fetch`, descrito a continuación.

`url-fetch` *url algo-hash hash* [*nombre*] [*#:executable? #f*] [Procedimiento]
Devuelve una derivación de salida fija que obtiene datos desde *url* (una cadena o una lista de cadenas indicando URL alternativas), la cual se espera que tenga el hash *hash* del tipo *algo-hash* (un símbolo). De manera predeterminada el nombre de archivo es el identificador final de la URL; de manera opcional se puede usar *nombre* para especificar un nombre de archivo diferente. Cuando *executable?* es verdadero se proporciona el permiso de ejecución al archivo descargado.

Cuando una de las URL comienza con `mirror://`, la parte de la máquina se interpreta como el nombre de un esquema de espejos, obtenido de `%mirror-file`.

De manera alternativa, cuando URL comienza con `file://`, devuelve el nombre de archivo del almacén correspondiente.

De igual modo el módulo `(guix git-download)` define el método de origen `git-fetch`, que descarga datos de un repositorio de control de versiones Git, y el tipo de datos `git-reference`, que describe el repositorio y la revisión que se debe obtener.

`git-fetch` *url algo-hash hash* [Procedimiento]
Devuelve una derivación de salida fija que obtiene *ref*, un objeto `<git-reference>`. El hash recursivo de la salida se espera que tenga el valor *hash* del tipo *algo-hash* (un símbolo). Se usa *nombre* para el nombre de archivo, o un nombre genérico si su valor es `#f`.

`git-fetch/lfs` *ref hash-algo hash* [Procedure]
This is a variant of the `git-fetch` procedure that supports the Git LFS (Large File Storage) extension. This may be useful to pull some binary test data to run the test suite of a package, for example.

`git-reference` [Tipo de datos]
Este es el tipo de datos que representa una referencia de Git que debe obtener el módulo `git-fetch`.

`url` La URL del repositorio Git que debe clonarse.

`commit` This string denotes either the commit to fetch (a hexadecimal string), or the tag to fetch. You can also use a "short" commit ID or a `git describe` style identifier such as `v1.0.1-10-g58d7909c97`.

`recursive?` (predeterminado: `#f`)
: Este valor booleano indica si se obtienen los sub-modulos de Git de manera recursiva.

El siguiente ejemplo representa la etiqueta `v2.10` del repositorio de GNU Hello:

```
(git-reference
  (url "https://git.savannah.gnu.org/git/hello.git")
  (commit "v2.10"))
```

Es equivalente a la siguiente referencia, la cual nombra de manera explícita la revisión:

```
(git-reference
  (url "https://git.savannah.gnu.org/git/hello.git")
  (commit "dc7dc56a00e48fe6f231a58f6537139fe2908fb9"))
```

For Mercurial repositories, the module `(guix hg-download)` defines the `hg-fetch` origin method and `hg-reference` data type for support of the Mercurial version control system.

`hg-fetch` *ref hash-algo hash* [*name*] [Procedure]
: Return a fixed-output derivation that fetches *ref*, a `<hg-reference>` object. The output is expected to have recursive hash *hash* of type *hash-algo* (a symbol). Use *name* as the file name, or a generic name if `#f`.

`hg-reference` [Data Type]
: This data type represents a Mercurial reference for `hg-fetch` to retrieve.

`url` The URL of the Mercurial repository to clone.

`changeset`
: This string denotes changeset to fetch.

For Subversion repositories, the module `(guix svn-download)` defines the `svn-fetch` origin method and `svn-reference` data type for support of the Subversion version control system.

`svn-fetch` *ref hash-algo hash* [*name*] [Procedure]
: Return a fixed-output derivation that fetches *ref*, a `<svn-reference>` object. The output is expected to have recursive hash *hash* of type *hash-algo* (a symbol). Use *name* as the file name, or a generic name if `#f`.

`svn-reference` [Data Type]
: This data type represents a Subversion reference for `svn-fetch` to retrieve.

`url` The URL of the Subversion repository to clone.

`revision` This string denotes revision to fetch specified as a number.

`recursive?` (predeterminado: `#f`)
: This Boolean indicates whether to recursively fetch Subversion “externals”.

`user-name` (default: `#f`)
: The name of an account that has read-access to the repository, if the repository isn't public.

`password` (predeterminada: `#f`)
: Password to access the Subversion repository, if required.

For Bazaar repositories, the module `(guix bzr-download)` defines the `bzr-fetch` origin method and `bzr-reference` data type for support of the Bazaar version control system.

`bzr-fetch` *ref hash-algo hash* [*name*] [Procedure]
: Return a fixed-output derivation that fetches *ref*, a `<bzr-reference>` object. The output is expected to have recursive hash *hash* of type *hash-algo* (a symbol). Use *name* as the file name, or a generic name if `#f`.

`bzr-reference` [Data Type]
: This data type represents a Bazaar reference for `bzr-fetch` to retrieve.

`url` The URL of the Bazaar repository to clone.

`revision` This string denotes revision to fetch specified as a number.

For CVS repositories, the module `(guix cvs-download)` defines the `cvs-fetch` origin method and `cvs-reference` data type for support of the Concurrent Versions System (CVS).

`cvs-fetch` *ref hash-algo hash* [*name*] [Procedure]
: Return a fixed-output derivation that fetches *ref*, a `<cvs-reference>` object. The output is expected to have recursive hash *hash* of type *hash-algo* (a symbol). Use *name* as the file name, or a generic name if `#f`.

`cvs-reference` [Data Type]
: This data type represents a CVS reference for `cvs-fetch` to retrieve.

`root-directory`
: The CVS root directory.

`module` Module to fetch.

`revision` Revision to fetch.

The example below denotes a version of gnu-standards to fetch:

```
(cvs-reference
  (root-directory ":pserver:anonymous@cvs.savannah.gnu.org:/sources/gnustandards"
  (module "gnustandards")
  (revision "2020-11-25"))
```

## 8.3 Definición de variantes de paquetes

One of the nice things with Guix is that, given a package definition, you can easily *derive* variants of that package—for a different upstream version, with different dependencies, different compilation options, and so on. Some of these custom packages can be defined straight from the command line (see Section 9.1.2 [Opciones de transformación de paquetes], page 184). This section describes how to define package variants in code. This can be useful in "manifests" (see Section 8.4 [Writing Manifests], page 119) and in your own package collection (see Section 6.7 [Creación de un canal], page 73), among others!

Como se ha mostrado previamente, los paquetes son objetos de primera clase del lenguage Scheme. El módulo `(guix packages)` proporciona la forma sintáctica `package` para definir nuevos objetos de paquetes (see Section 8.2.1 [Referencia de package], page 105). La forma más fácil de definir una variante de un paquete es usar la palabra clave `inherit` junto a `package`. Esto le permite heredar de una definición de paquete y modificar únicamente los campos que desee.

Por ejemplo, a partir de la variable `hello`, que contiene la definición de la versión actual de GNU Hello, podría definir de esta forma una variante para la versión 2.2 (publicada 2006, ¡con solera!):

```
(use-modules (gnu packages base))    ;para 'hello'

(define hello-2.2
  (package
    (inherit hello)
    (version "2.2")
    (source (origin
              (method url-fetch)
              (uri (string-append "mirror://gnu/hello/hello-" version
                                  ".tar.gz"))
              (sha256
               (base32
                "0lappv4slgb5spyqbh6yl5r013zv72yqg2pcl30mginf3wdqd8k9"))))))
```

The example above corresponds to what the `--with-version` or `--with-source` package transformations option do. Essentially `hello-2.2` preserves all the fields of `hello`, except `version` and `source`, which it overrides. Note that the original `hello` variable is still there, in the `(gnu packages base)` module, unchanged. When you define a custom package like this, you are really *adding* a new package definition; the original one remains available.

De igual manera puede definir variantes con un conjunto de dependencias distinto al del paquete original. Por ejemplo, el paquete `gdb` predeterminado depende de `guile` pero, puesto que es una dependencia opcional, podría definir una variante que elimina dicha dependencia de este modo:

```
(use-modules (gnu packages gdb))   ;for 'gdb'

(define gdb-sans-guile
  (package
    (inherit gdb)
    (inputs (modify-inputs (package-inputs gdb)
              (delete "guile")))))
```

The `modify-inputs` form above removes the `"guile"` package from the `inputs` field of `gdb`. The `modify-inputs` macro is a helper that can prove useful anytime you want to remove, add, or replace package inputs.

`modify-inputs` *inputs clauses* [Macro]

Modify the given package inputs, as returned by `package-inputs` & co., according to the given clauses. Each clause must have one of the following forms:

`(delete` *name*`...)`
: Delete from the inputs packages with the given *name*s (strings).

`(prepend` *package*`...)`
: Add *package*s to the front of the input list.

`(append` *package*`...)`
: Add *package*s to the end of the input list.

`(replace` *name replacement*`)`
: Replace the package called *name* with *replacement*.

The example below removes the GMP and ACL inputs of Coreutils and adds libcap to the front of the input list:

```
(modify-inputs (package-inputs coreutils)
  (delete "gmp" "acl")
  (prepend libcap))
```

The example below replaces the `guile` package from the inputs of `guile-redis` with `guile-2.2`:

```
(modify-inputs (package-inputs guile-redis)
  (replace "guile" guile-2.2))
```

The last type of clause is `append`, to add inputs at the back of the list.

En ciertos casos encontrará útil escribir funciones («procedimientos» en el vocabulario de Scheme) que devuelven un paquete en base a ciertos parámetros. Por ejemplo, considere la biblioteca `luasocket` para el lenguaje de programación Lua. Se desea crear paquetes de `luasocket` para las versiones mayores de Lua. Una forma de hacerlo es definir un procedimiento que recibe un paquete Lua y devuelve un paquete `luasocket` que depende de él:

```
(define (make-lua-socket name lua)
  ;; Return a luasocket package built with LUA.
  (package
    (name name)
    (version "3.0")
    ;; several fields omitted
    (inputs (list lua))
    (synopsis "Socket library for Lua")))

(define-public lua5.1-socket
  (make-lua-socket "lua5.1-socket" lua-5.1))

(define-public lua5.2-socket
  (make-lua-socket "lua5.2-socket" lua-5.2))
```

En este ejemplo se han definido los paquetes `lua5.1-socket` y `lua5.2-socket` llamando a `crea-lua-socket` con distintos parámetros. See Section “Procedures” in *GNU Guile Reference Manual* para más información sobre procedimientos. El hecho de disponer de definiciones públicas de nivel superior de estos dos paquetes permite que se les haga referencia desde la línea de órdenes (see Section 8.1 [Módulos de paquetes], page 101).

Estas son variantes muy simples. Para facilitar esta tarea, el módulo `(guix transformations)` proporciona una interfaz de alto nivel que se corresponde directamente con las opciones de transformación de paquetes más sofisticadas (see Section 9.1.2 [Opciones de transformación de paquetes], page 184):

`options->transformation` *opciones* [Procedimiento]

Devuelve un procedimiento que, cuando se le proporciona un objeto que construir (paquete, derivación, etc.), aplica las transformaciones especificadas en *opciones* y devuelve los objetos resultantes. *opciones* debe ser una lista de pares símbolo/cadena como los siguientes:

```
((with-branch . "guile-gcrypt=master")
 (without-tests . "libgcrypt"))
```

Cada símbolo nombra una transformación y la cadena correspondiente es el parámetro de dicha transformación.

Por ejemplo, un manifiesto equivalente a esta orden:

```
guix build guix \
  --with-branch=guile-gcrypt=master \
  --with-debug-info=zlib
```

... sería algo parecido a esto:

```
(use-modules (guix transformations))

(define transforma
  ;; El procedimiento de transformación del paquete.
  (options->transformation
   '((with-branch . "guile-gcrypt=master")
     (with-debug-info . "zlib"))))

(packages->manifest
 (list (transforma (specification->package "guix"))))
```

El procedimiento `options->transformation` es conveniente, pero quizá no es tan flexible como pudiese desear. ¿Cómo se ha implementado? Es posible que ya se haya percatado de que la mayoría de las opciones de transformación de paquetes van más allá de los cambios superficiales mostrados en los primeros ejemplos de esta sección: implican *reescritura de entradas*, lo que significa que el grafo de dependencias de un paquete se reescribe sustituyendo entradas específicas por otras.

La reescritura del grafo de dependencias, con el propósito de reemplazar paquetes del grafo, es implementada por el procedimiento `package-input-rewriting` en `(guix packages)`.

`package-input-rewriting` *reemplazos* [*nombre-reescrito*] [*#:deep? #t*] [Procedimiento]

Devuelve un procedimiento que, cuando se le pasa un paquete, reemplaza sus dependencias directas e indirectas, incluyendo sus entradas implícitas cuando *deep?* es verdadero, de acuerdo a *reemplazos*. *reemplazos* es una lista de pares de paquetes; el primer elemento de cada par es el paquete a reemplazar, el segundo es el reemplazo.

Opcionalmente, *nombre-reescrito* es un procedimiento de un parámetro que toma el nombre del paquete y devuelve su nuevo nombre tras la reescritura.

Considere este ejemplo:

```
(define libressl-en-vez-de-openssl
  ;; Esto es un procedimiento para reemplazar OPENSSL
  ;; por LIBRESSL, recursivamente.
  (package-input-rewriting `((,openssl . ,libressl))))

(define git-con-libressl
  (libressl-en-vez-de-openssl git))
```

Aquí primero definimos un procedimiento de reescritura que substituye *openssl* por *libressl*. Una vez hecho esto, lo usamos para definir una *variante* del paquete *git* que usa *libressl* en vez de *openssl*. Esto es exactamente lo que hace la opción de línea de órdenes `--with-input` (see Section 9.1.2 [Opciones de transformación de paquetes], page 184).

La siguiente variante de `package-input-rewriting` puede encontrar paquetes a reemplazar por su nombre en vez de por su identidad.

`package-input-rewriting/spec` *reemplazos* [*#:deep? #t*] [Procedimiento]
Devuelve un procedimiento que, proporcionado un paquete, realiza los *reemplazos* proporcionados sobre todo el grafo del paquete, incluyendo las entradas implícitas a menos que *deep?* sea falso.

*replacements* is a list of spec/procedures pair; each spec is a package specification such as `"gcc"` or `"guile@2"`, and each procedure takes a matching package and returns a replacement for that package. Matching packages that have the `hidden?` property set are not replaced.

El ejemplo previo podría ser reescrito de esta forma:

```
(define libressl-en-vez-de-openssl
  ;; Reemplaza todos los paquetes llamados "openssl" con LibreSSL.
  (package-input-rewriting/spec `(("openssl" . ,(const libressl)))))
```

La diferencia principal en este caso es que, esta vez, los paquetes se buscan por su especificación y no por su identidad. En otras palabras, cualquier paquete en el grafo que se llame `openssl` será reemplazado.

Un procedimiento más genérico para reescribir el grafo de dependencias de un paquete es `package-mapping`: acepta cambios arbitrarios sobre nodos del grafo.

`package-mapping` *proc* [*cortar?*] [*#:deep? #f*] [Procedimiento]
Devuelve un procedimiento que, dado un paquete, aplica *proc* a todos los paquetes de los que depende y devuelve el paquete resultante. El procedimiento para la recursión cuando *cortar?* devuelve verdadero para un paquete dado. Cuando *deep?* tiene valor verdadero, *proc* se aplica también a las entradas implícitas.

**Tips:** Understanding what a variant really looks like can be difficult as one starts combining the tools shown above. There are several ways to inspect a package before attempting to build it that can prove handy:

- You can inspect the package interactively at the REPL, for instance to view its inputs, the code of its build phases, or its configure flags (see Section 8.14 [Using Guix Interactively], page 178).
- When rewriting dependencies, `guix graph` can often help visualize the changes that are made (see Section 9.10 [Invocación de guix graph], page 221).

## 8.4 Writing Manifests

`guix` commands let you specify package lists on the command line. This is convenient, but as the command line becomes longer and less trivial, it quickly becomes more convenient to have that package list in what we call a *manifest*. A manifest is some sort of a "bill of materials" that defines a package set. You would typically come up with a code snippet that builds the manifest, store it in a file, say `manifest.scm`, and then pass that file to the `-m` (or `--manifest`) option that many `guix` commands support. For example, here's what a manifest for a simple package set might look like:

```
;; Manifest for three packages.
(specifications->manifest '("gcc-toolchain" "make" "git"))
```

Once you have that manifest, you can pass it, for example, to `guix package` to install just those three packages to your profile (see [profile-manifest], page 40):

```
guix package -m manifest.scm
```

... or you can pass it to `guix shell` (see [shell-manifest], page 82) to spawn an ephemeral environment:

```
guix shell -m manifest.scm
```

... or you can pass it to `guix pack` in pretty much the same way (see [pack-manifest], page 97). You can store the manifest under version control, share it with others so they can easily get set up, etc.

But how do you write your first manifest? To get started, maybe you'll want to write a manifest that mirrors what you already have in a profile. Rather than start from a blank page, `guix package` can generate a manifest for you (see [export-manifest], page 45):

```
# Write to 'manifest.scm' a manifest corresponding to the
# default profile, ~/.guix-profile.
guix package --export-manifest > manifest.scm
```

Or maybe you'll want to "translate" command-line arguments into a manifest. In that case, `guix shell` can help (see [shell-export-manifest], page 82):

```
# Write a manifest for the packages specified on the command line.
guix shell --export-manifest gcc-toolchain make git > manifest.scm
```

In both cases, the `--export-manifest` option tries hard to generate a faithful manifest; in particular, it takes package transformation options into account (see Section 9.1.2 [Opciones de transformación de paquetes], page 184).

> **Nota:** Manifests are *symbolic*: they refer to packages of the channels *currently in use* (see Chapter 6 [Canales], page 69). In the example above, `gcc-toolchain` might refer to version 11 today, but it might refer to version 13 two years from now.

If you want to "pin" your software environment to specific package versions and variants, you need an additional piece of information: the list of channel revisions in use, as returned by `guix describe`. See Section 6.3 [Replicación de Guix], page 70, for more information.

Once you've obtained your first manifest, perhaps you'll want to customize it. Since your manifest is code, you now have access to all the Guix programming interfaces!

Let's assume you want a manifest to deploy a custom variant of GDB, the GNU Debugger, that does not depend on Guile, together with another package. Building on the example seen in the previous section (see Section 8.3 [Definición de variantes de paquetes], page 114), you can write a manifest along these lines:

```
(use-modules (guix packages)
             (gnu packages gdb)               ;for 'gdb'
             (gnu packages version-control))  ;for 'git'

;; Define a variant of GDB without a dependency on Guile.
(define gdb-sans-guile
  (package
    (inherit gdb)
    (inputs (modify-inputs (package-inputs gdb)
              (delete "guile")))))

;; Return a manifest containing that one package plus Git.
(packages->manifest (list gdb-sans-guile git))
```

Note that in this example, the manifest directly refers to the `gdb` and `git` variables, which are bound to a `package` object (see Section 8.2.1 [Referencia de package], page 105), instead of calling `specifications->manifest` to look up packages by name as we did before. The `use-modules` form at the top lets us access the core package interface (see Section 8.2 [Definición de paquetes], page 102) and the modules that define `gdb` and `git` (see Section 8.1 [Módulos de paquetes], page 101). Seamlessly, we're weaving all this together—the possibilities are endless, unleash your creativity!

The data type for manifests as well as supporting procedures are defined in the `(guix profiles)` module, which is automatically available to code passed to `-m`. The reference follows.

`manifest` [Data Type]

Data type representing a manifest.

It currently has one field:

`entries` This must be a list of `manifest-entry` records—see below.

`manifest-entry` [Data Type]

Data type representing a manifest entry. A manifest entry contains essential metadata: a name and version string, the object (usually a package) for that entry, the desired output (see Section 5.4 [Paquetes con múltiples salidas], page 51), and a number of optional pieces of information detailed below.

Most of the time, you won't build a manifest entry directly; instead, you will pass a package to `package->manifest-entry`, described below. In some unusual cases

though, you might want to create manifest entries for things that are *not* packages, as in this example:

```
;; Manually build a single manifest entry for a non-package object.
(let ((hello (program-file "hello" #~(display "Hi!"))))
  (manifest-entry
    (name "foo")
    (version "42")
    (item
     (computed-file "hello-directory"
                    #~(let ((bin (string-append #$output "/bin")))
                        (mkdir #$output) (mkdir bin)
                        (symlink #$hello
                                 (string-append bin "/hello")))))))
```

The available fields are the following:

`name`
`version` Name and version string for this entry.

`item` A package or other file-like object (see Section 8.12 [Expresiones-G], page 167).

`output` (default: `"out"`)
Output of `item` to use, in case `item` has multiple outputs (see Section 5.4 [Paquetes con múltiples salidas], page 51).

`dependencies` (predeterminadas: `'()`)
List of manifest entries this entry depends on. When building a profile, dependencies are added to the profile.

Typically, the propagated inputs of a package (see Section 8.2.1 [Referencia de package], page 105) end up having a corresponding manifest entry in among the dependencies of the package's own manifest entry.

`search-paths` (predeterminadas: `'()`)
The list of search path specifications honored by this entry (see Section 8.8 [Search Paths], page 154).

`properties` (default: `'()`)
List of symbol/value pairs. When building a profile, those properties get serialized.

This can be used to piggyback additional metadata—e.g., the transformations applied to a package (see Section 9.1.2 [Opciones de transformación de paquetes], page 184).

`parent` (default: `(delay #f)`)
A promise pointing to the "parent" manifest entry.

This is used as a hint to provide context when reporting an error related to a manifest entry coming from a `dependencies` field.

`concatenate-manifests` *lst* [Procedure]
Concatenate the manifests listed in *lst* and return the resulting manifest.

`package->manifest-entry` *package* [*output*] [*#:properties*] [Procedure]
Return a manifest entry for the *output* of package *package*, where *output* defaults to `"out"`, and with the given *properties*. By default *properties* is the empty list or, if one or more package transformations were applied to *package*, it is an association list representing those transformations, suitable as an argument to `options->transformation` (see Section 8.3 [Definición de variantes de paquetes], page 114).

The code snippet below builds a manifest with an entry for the default output and the `send-email` output of the `git` package:

```
(use-modules (gnu packages version-control))

(manifest (list (package->manifest-entry git)
                (package->manifest-entry git "send-email")))
```

`packages->manifest` *packages* [Procedure]
Return a list of manifest entries, one for each item listed in *packages*. Elements of *packages* can be either package objects or package/string tuples denoting a specific output of a package.

Using this procedure, the manifest above may be rewritten more concisely:

```
(use-modules (gnu packages version-control))

(packages->manifest (list git `(,git "send-email")))
```

`package->development-manifest` *package* [*system*] [*#:target*] [Procedure]
Return a manifest for the *development inputs* of *package* for *system*, optionally when cross-compiling to *target*. Development inputs include both explicit and implicit inputs of *package*.

Like the `-D` option of `guix shell` (see [shell-development-option], page 81), the resulting manifest describes the environment in which one can develop *package*. For example, suppose you're willing to set up a development environment for Inkscape, with the addition of Git for version control; you can describe that "bill of materials" with the following manifest:

```
(use-modules (gnu packages inkscape)          ;for 'inkscape'
             (gnu packages version-control))  ;for 'git'

(concatenate-manifests
 (list (package->development-manifest inkscape)
       (packages->manifest (list git))))
```

In this example, the development manifest that `package->development-manifest` returns includes the compiler (GCC), the many supporting libraries (Boost, GLib, GTK, etc.), and a couple of additional development tools—these are the dependencies `guix show inkscape` lists.

Last, the `(gnu packages)` module provides higher-level facilities to build manifests. In particular, it lets you look up packages by name—see below.

`specifications->manifest` *specs* [Procedure]

Given *specs*, a list of specifications such as `"emacs@25.2"` or `"guile:debug"`, return a manifest. Specs have the format that command-line tools such as `guix install` and `guix package` understand (see Section 5.2 [Invocación de guix package], page 36).

As an example, it lets you rewrite the Git manifest that we saw earlier like this:

```
(specifications->manifest '("git" "git:send-email"))
```

Notice that we do not need to worry about `use-modules`, importing the right set of modules, and referring to the right variables. Instead, we directly refer to packages in the same way as on the command line, which can often be more convenient.

## 8.5 Sistemas de construcción

Cada definición de paquete especifica un *sistema de construcción* y parámetros para dicho sistema de construcción (see Section 8.2 [Definición de paquetes], page 102). Este campo `build-system` representa el procedimiento de construcción del paquete, así como las dependencias implícitas de dicho procedimiento de construcción.

Los sistemas de construcción son objetos `<build-system>`. La interfaz para crear y manipularlos se proporciona en el módulo `(guix build-system)`, y otros módulos exportan sistemas de construcción reales.

En su implementación, los sistemas de construcción primero compilan los objetos package a objetos *bag*. Una bolsa (traducción de *bag*) es como un paquete, pero con menos ornamentos—en otras palabras, una bolsa es una representación a un nivel más bajo de un paquete, que contiene todas las entradas de dicho paquete, incluyendo algunas implícitamente añadidas por el sistema de construcción. Esta representación intermedia se compila entonces a una derivación (see Section 8.10 [Derivaciones], page 159). EL procedimiento `package-with-c-toolchain` es un ejemplo de una forma de cambiar las entradas implícitas que el sistema de construcción del paquete incluye (see Section 8.2.1 [Referencia de package], page 105).

Los sistemas de construcción aceptan una lista opcional de *parámetros*. En las definiciones de paquete, estos son pasados *vía* el campo `arguments` (see Section 8.2 [Definición de paquetes], page 102). Normalmente son parámetros con palabras clave (see Section "Optional Arguments" in *GNU Guile Reference Manual*). El valor de estos parámetros normalmente se evalúa en la *capa de construcción*—es decir, por un proceso Guile lanzado por el daemon (see Section 8.10 [Derivaciones], page 159).

El sistema de construcción principal es *gnu-build-system*, el cual implementa el procedimiento estándar de construcción para GNU y muchos otros paquetes. Se proporciona por el módulo `(guix build-system gnu)`.

`gnu-build-system` [Variable]

*gnu-build-system* representa el sistema de construcción GNU y sus variantes (see Section "Configuration" in *GNU Coding Standards*).

In a nutshell, packages using it are configured, built, and installed with the usual `./configure && make && make check && make install` command sequence. In practice, a few additional steps are often needed. All these steps are split up in separate *phases*. See Section 8.6 [Fases de construcción], page 143, for more info on build phases and ways to customize them.

Además, este sistema de construcción asegura que el entorno "estándar" para paquetes GNU está disponible. Esto incluye herramientas como GCC, libc, Coreutils, Bash, Make, Diffutils, grep y sed (vea el módulo `(guix build system gnu)` para una lista completa). A estas las llamamos las *entradas implícitas* de un paquete, porque las definiciones de paquete no las mencionan.

This build system supports a number of keyword arguments, which can be passed *via* the `arguments` field of a package. Here are some of the main parameters:

`#:phases` This argument specifies build-side code that evaluates to an alist of build phases. See Section 8.6 [Fases de construcción], page 143, for more information.

`#:configure-flags`
: This is a list of flags (strings) passed to the `configure` script. See Section 8.2 [Definición de paquetes], page 102, for an example.

`#:make-flags`
: This list of strings contains flags passed as arguments to `make` invocations in the `build`, `check`, and `install` phases.

`#:out-of-source?`
: This Boolean, `#f` by default, indicates whether to run builds in a build directory separate from the source tree.

  When it is true, the `configure` phase creates a separate build directory, changes to that directory, and runs the `configure` script from there. This is useful for packages that require it, such as `glibc`.

`#:tests?` This Boolean, `#t` by default, indicates whether the `check` phase should run the package's test suite.

`#:test-target`
: This string, `"check"` by default, gives the name of the makefile target used by the `check` phase.

`#:parallel-build?`
`#:parallel-tests?`
: These Boolean values specify whether to build, respectively run the test suite, in parallel, with the `-j` flag of `make`. When they are true, `make` is passed `-jn`, where *n* is the number specified as the `--cores` option of `guix-daemon` or that of the `guix` client command (see Section 9.1.1 [Opciones comunes de construcción], page 181).

`#:validate-runpath?`
: This Boolean, `#t` by default, determines whether to "validate" the `RUNPATH` of ELF binaries (`.so` shared libraries as well as executables) previously installed by the `install` phase. See [phase-validate-runpath], page 144, for details.

`#:substitutable?`
: This Boolean, `#t` by default, tells whether the package outputs should be substitutable—i.e., whether users should be able to obtain substitutes for them instead of building locally (see Section 5.3 [Sustituciones], page 46).

`#:allowed-references`
`#:disallowed-references`

When true, these arguments must be a list of dependencies that must not appear among the references of the build results. If, upon build completion, some of these references are retained, the build process fails.

This is useful to ensure that a package does not erroneously keep a reference to some of it build-time inputs, in cases where doing so would, for example, unnecessarily increase its size (see Section 9.9 [Invocación de guix size], page 219).

Most other build systems support these keyword arguments.

Hay definidos otros objetos `<build-system>` para implementar otras convenciones y herramientas usadas por paquetes de software libre. Heredan la mayor parte de *gnu-build-system*, y se diferencian principalmente en el conjunto de entradas implícitamente añadidas al proceso de construcción, y en la lista de fases ejecutadas. Algunos de estos sistemas de construcción se enumeran a continuación.

`agda-build-system` [Variable]

This variable is exported by `(guix build-system agda)`. It implements a build procedure for Agda libraries.

It adds `agda` to the set of inputs. A different Agda can be specified with the `#:agda` key.

The `#:plan` key is a list of cons cells `(`*`regexp`*` . `*`parameters`*`)`, where *regexp* is a regexp that should match the `.agda` files to build, and *parameters* is an optional list of parameters that will be passed to `agda` when type-checking it.

When the library uses Haskell to generate a file containing all imports, the convenience `#:gnu-and-haskell?` can be set to `#t` to add `ghc` and the standard inputs of `gnu-build-system` to the input list. You will still need to manually add a phase or tweak the `'build` phase, as in the definition of `agda-stdlib`.

`ant-build-system` [Variable]

`(guix build-system ant)` exporta esta variable. Implementa el procedimiento de construcción de paquetes Java que pueden construirse con la herramienta de construcción Ant (`https://ant.apache.org/`).

Añade tanto `ant` como el *kit de desarrollo Java* (JDK), que proporciona el paquete `icedtea`, al conjunto de entradas. Se pueden especificar paquetes diferentes con los parámetros `#:ant` y `#:jdk`, respectivamente.

Cuando el paquete original no proporciona un archivo Ant apropiado, el parámetro `#:jar-name` puede usarse para generar un archivo de construcción Ant `build.xml` mínimo con tareas para construir el archivo jar especificado. En este caso, el parámetro `#:source-dir` se puede usar para especificar el subdirectorio de fuentes, con "src" como valor predeterminado.

The `#:main-class` parameter can be used with the minimal ant buildfile to specify the main class of the resulting jar. This makes the jar file executable. The `#:test-include` parameter can be used to specify the list of junit tests to run. It defaults to `(list "**/*Test.java")`. The `#:test-exclude` can be used to disable some tests.

It defaults to `(list "**/Abstract*.java")`, because abstract classes cannot be run as tests.

El parámetro `#:build-target` se puede usar para especificar la tarea Ant que debe ser ejecutada durante la fase `build`. Por defecto se ejecuta la tarea "jar".

`android-ndk-build-system` [Variable]

Esta variable es exportada por `(guix build-system android-ndk)`. Implementa un procedimiento de construcción para paquetes Android NDK (kit de desarrollo nativo) usando un proceso de construcción específico de Guix.

El sistema de construcción asume que los paquetes instalan sus archivos de interfaz pública (cabeceras) en el subdirectorio `include` de la salida `out` y sus bibliotecas en el subdirectorio `lib` de la salida `out"`.

También se asume que la unión de todas las dependencias de un paquete no tiene archivos en conflicto.

En este momento no funciona la compilación cruzada - por lo que las bibliotecas y los archivos de cabecera se asumen que son locales.

`asdf-build-system/source` [Variable]
`asdf-build-system/sbcl` [Variable]
`asdf-build-system/ecl` [Variable]

These variables, exported by `(guix build-system asdf)`, implement build procedures for Common Lisp packages using "ASDF" (`https://common-lisp.net/project/asdf/`). ASDF is a system definition facility for Common Lisp programs and libraries.

The `asdf-build-system/source` system installs the packages in source form, and can be loaded using any common lisp implementation, via ASDF. The others, such as `asdf-build-system/sbcl`, install binary systems in the format which a particular implementation understands. These build systems can also be used to produce executable programs, or lisp images which contain a set of packages pre-loaded.

El sistema de construcción usa convenciones de nombres. Para paquetes binarios, el paquete debería estar prefijado con la implementación lisp, como `sbcl-` para `asdf-build-system/sbcl`.

Adicionalmente, el paquete de fuentes correspondiente debe etiquetarse usando la misma convención que los paquetes Python (see Section 22.8.8 [Módulos Python], page 754), usando el prefijo `cl-`.

In order to create executable programs and images, the build-side procedures `build-program` and `build-image` can be used. They should be called in a build phase after the `create-asdf-configuration` phase, so that the system which was just built can be used within the resulting image. `build-program` requires a list of Common Lisp expressions to be passed as the `#:entry-program` argument.

By default, all the `.asd` files present in the sources are read to find system definitions. The `#:asd-files` parameter can be used to specify the list of `.asd` files to read. Furthermore, if the package defines a system for its tests in a separate file, it will be loaded before the tests are run if it is specified by the `#:test-asd-file` parameter. If it is not set, the files `<system>-tests.asd`, `<system>-test.asd`, `tests.asd`, and `test.asd` will be tried if they exist.

If for some reason the package must be named in a different way than the naming conventions suggest, or if several systems must be compiled, the `#:asd-systems` parameter can be used to specify the list of system names.

`cargo-build-system` [Variable]

Esta variable se exporta en `(guix build-system cargo)`. Permite la construcción de paquetes usando Cargo, la herramienta de construcción del lenguaje de programación Rust (`https://www.rust-lang.org`).

Automáticamente añade `rustc` y `cargo` al conjunto de entradas. Se puede especificar el uso de un paquete Rust distinto con el parámetro `#:rust`.

Regular cargo dependencies should be added to the package definition similarly to other packages; those needed only at build time to native-inputs, others to inputs. If you need to add source-only crates then you should add them to via the `#:cargo-inputs` parameter as a list of name and spec pairs, where the spec can be a package or a source definition. Note that the spec must evaluate to a path to a gzipped tarball which includes a `Cargo.toml` file at its root, or it will be ignored. Similarly, cargo dev-dependencies should be added to the package definition via the `#:cargo-development-inputs` parameter.

In its `configure` phase, this build system will make any source inputs specified in the `#:cargo-inputs` and `#:cargo-development-inputs` parameters available to cargo. It will also remove an included `Cargo.lock` file to be recreated by `cargo` during the `build` phase. The `package` phase will run `cargo package` to create a source crate for future use. The `install` phase installs the binaries defined by the crate. Unless `install-source? #f` is defined it will also install a source crate repository of itself and unpacked sources, to ease in future hacking on rust packages.

`chicken-build-system` [Variable]

This variable is exported by `(guix build-system chicken)`. It builds CHICKEN Scheme (`https://call-cc.org/`) modules, also called "eggs" or "extensions". CHICKEN generates C source code, which then gets compiled by a C compiler, in this case GCC.

This build system adds `chicken` to the package inputs, as well as the packages of `gnu-build-system`.

The build system can't (yet) deduce the egg's name automatically, so just like with `go-build-system` and its `#:import-path`, you should define `#:egg-name` in the package's `arguments` field.

For example, if you are packaging the `srfi-1` egg:

```
(arguments '(#:egg-name "srfi-1"))
```

Egg dependencies must be defined in `propagated-inputs`, not `inputs` because CHICKEN doesn't embed absolute references in compiled eggs. Test dependencies should go to `native-inputs`, as usual.

`copy-build-system` [Variable]

Esta variable se exporta en `(guix build-system copy)`. Permite la construcción de paquetes simples que no necesitan mucha compilación y en su mayor parte dependen únicamente de la copia de archivos en distintas rutas.

Añade gran parte de los paquetes de `gnu-build-system` al conjunto de entradas. Por esta razón, `copy-build-system` no necesita toda la verborrea que habitualmente requiere `trivial-build-system`.

Para simplificar más aún el proceso de instalación de archivos, se expone un parámetro `#:install-plan` para permitir a quien genera el paquete especificar dónde van los distintos archivos. El plan de instalación (`#:install-plan`) es una lista de (*fuente destino* [*filtro*]). Los *filtros* son opcionales.

- Cuando *fuente* corresponde con un archivo o un directorio sin la barra final, se instala en *destino*.
  - Si *destino* contiene una barra al final, *fuente* se instala con su nombre de archivo en la ruta *destino*.
  - En otro caso se instala *fuente* como *destino*.
- Cuando *fuente* es un directorio terminado en una barra, o cuando se usa algún *filtro*, la barra al final de *destino* tiene el significado que se describió anteriormente.
  - Sin *filtros*, instala todo el *contenido* de *fuente* en *destino*.
  - Cuando *filtro* contiene `#:include`, `#:include-regexp`, `#:exclude`, `#:exclude-regexp`, únicamente se seleccionan los archivos instalados dependiendo de los filtros. Cada filtro se especifica como una lista de cadenas.
    - Con `#:include`, se instalan todos los archivos de la ruta cuyo sufijo corresponda con al menos uno de los elementos de la lista proporcionada.
    - Con `#:include-regexp` se instalan todos los archivos cuyas rutas relativas correspondan con al menos una de las expresiones regulares en la lista proporcionada.
    - Los filtros `#:exclude` y `#:exclude-regexp` son complementarios a sus equivalentes inclusivos. Sin opciones `#:include`, se instalan todos los archivos excepto aquellos que correspondan con los filtros de exclusión. Si se proporcionan tanto inclusiones como exclusiones, las exclusiones tienen efecto sobre los resultados de las inclusiones.

  En todos los casos, las rutas relativas a *fuente* se preservan dentro de *destino*.

Ejemplos:

- `("foo/bar" "share/my-app/")`: Instala `bar` en `share/my-app/bar`.
- `("foo/bar" "share/my-app/baz")`: Instala `bar` en `share/my-app/baz`.
- `("foo/" "share/my-app")`: Instala el contenido de `foo` dentro de `share/my-app`, por ejemplo, instala `foo/sub/file` en `share/my-app/sub/file`.
- `("foo/" "share/my-app" #:include ("sub/file"))`: Instala únicamente `foo/sub/file` en `share/my-app/sub/file`.
- `("foo/sub" "share/my-app" #:include ("file"))`: Instala `foo/sub/file` en `share/my-app/file`.

`vim-build-system` [Variable]

This variable is exported by `(guix build-system vim)`. It is an extension of the `copy-build-system`, installing Vim and Neovim plugins into locations where these two text editors know to find their plugins, using their packpaths.

Packages which are prefixed with `vim-` will be installed in Vim's packpath, while those prefixed with `neovim-` will be installed in Neovim's packpath. If there is a `doc` directory with the plugin then helptags will be generated automatically.

There are a couple of keywords added with the `vim-build-system`:

- With `plugin-name` it is possible to set the name of the plugin. While by default this is set to the name and version of the package, it is often more helpful to set this to name which the upstream author calls their plugin. This is the name used for `:packadd` from inside Vim.
- With `install-plan` it is possible to augment the built-in install-plan of the `vim-build-system`. This is particularly helpful if you have files which should be installed in other locations. For more information about using the `install-plan`, see the `copy-build-system` (see Section 8.5 [Sistemas de construcción], page 123).
- With `#:vim` it is possible to add this package to Vim's packpath, in addition to if it is added automatically because of the `vim-` prefix in the package's name.
- With `#:neovim` it is possible to add this package to Neovim's packpath, in addition to if it is added automatically because of the `neovim-` prefix in the package's name.
- With `#:mode` it is possible to adjust the path which the plugin is installed into. By default the plugin is installed into `start` and other options are available, including `opt`. Adding a plugin into `opt` will mean you will need to run, for example, `:packadd foo` to load the `foo` plugin from inside of Vim.

`clojure-build-system` [Variable]

Esta variable se exporta en `(guix build-system clojure)`. Implementa un procedimiento de construcción simple para paquetes Clojure (`https://clojure.org/`) usando directamente `compile` en Clojure. La compilación cruzada no está disponible todavía.

Añade `clojure`, `icedtea` y `zip` al conjunto de entradas. Se pueden especificar paquetes diferentes con los parámetros `#:clojure`, `#:jdk` y `#:zip`, respectivamente.

Una lista de directorios de fuentes, directorios de pruebas y nombres de jar pueden especificarse con los parámetros `#:source-dirs`, `#:test-dirs` y `#:jar-names`, respectivamente. El directorio de compilación y la clase principal pueden especificarse con los parámetros `#:compile-dir` y `#:main-class`, respectivamente. Otros parámetros se documentan más adelante.

Este sistema de construcción es una extensión de *ant-build-system*, pero con las siguientes fases cambiadas:

`build` Esta fase llama `compile` en Clojure para compilar los archivos de fuentes y ejecuta `jar` para crear archivadores jar tanto de archivos de fuentes y compilados de acuerdo con las listas de inclusión y exclusión especificadas en `#:aot-include` y `#:aot-exclude`, respectivamente. La lista de

exclusión tiene prioridad sobre la de inclusión. Estas listas consisten en símbolos que representan bibliotecas Clojure o la palabra clave especial `#:all` que representa todas las bibliotecas encontradas en los directorios de fuentes. El parámetro `#:omit-source?` determina si las fuentes deben incluirse en los archivadores jar.

`check` Esta fase ejecuta las pruebas de acuerdo a las listas de inclusión y exclusión especificadas en `#:test-include` y `#:test-exclude`, respectivamente. Sus significados son análogos a los de `#:aot-include` y `#:aot-exclude`, excepto que la palabra clave especial `#:all` designa ahora a todas las bibliotecas Clojure encontradas en los directorios de pruebas. El parámetro `#:tests?` determina si se deben ejecutar las pruebas.

`install` Esta fase instala todos los archivadores jar construidos previamente.

Además de las previas, este sistema de construcción contiene una fase adicional:

`install-doc`
Esta fase instala todos los archivos de nivel superior con un nombre que corresponda con *%doc-regex*. Una expresión regular diferente se puede especificar con el parámetro `#:doc-regex`. Todos los archivos dentro (recursivamente) de los directorios de documentación especificados en `#:doc-dirs` se instalan también.

`cmake-build-system` [Variable]
Esta variable se exporta en `(guix build-system cmake)`. Implementa el procedimiento de construcción para paquetes que usen la herramienta de construcción CMake (`https://www.cmake.org`).

Automáticamente añade el paquete `cmake` al conjunto de entradas. El paquete usado se puede especificar con el parámetro `#:cmake`.

El parámetro `#:configure-flags` se toma como una lista de opciones a pasar a `cmake`. El parámetro `#:build-type` especifica en términos abstractos las opciones pasadas al compilador; su valor predeterminado es `"RelWithDebInfo"` (quiere decir "modo de entrega con información de depuración"), lo que aproximadamente significa que el código se compila con `-O2 -g`, lo cual es el caso predeterminado en paquetes basados en Autoconf.

`composer-build-system` [Variable]
This variable is exported by `(guix build-system composer)`. It implements the build procedure for packages using Composer (`https://getcomposer.org/`), the PHP package manager.

It automatically adds the `php` package to the set of inputs. Which package is used can be specified with the `#:php` parameter.

The `#:test-target` parameter is used to control which script is run for the tests. By default, the `test` script is run if it exists. If the script does not exist, the build system will run `phpunit` from the source directory, assuming there is a `phpunit.xml` file.

`dune-build-system` [Variable]

Esta variable se exporta en `(guix build-system dune)`. Permite la construcción de paquetes mediante el uso de Dune (`https://dune.build/`), una herramienta de construcción para el lenguaje de programación OCaml. Se implementa como una extensión de `ocaml-build-system` que se describe a continuación. Como tal, se pueden proporcionar los parámetros `#:ocaml` y `#:findlib` a este sistema de construcción.

Automáticamente añade el paquete `dune` al conjunto de entradas. El paquete usado se puede especificar con el parámetro `#:dune`.

No existe una fase `configure` debido a que los paquetes dune no necesitan ser configurados típicamente. El parámetro `#:build-flags` se toma como una lista de opciones proporcionadas a la orden `dune` durante la construcción.

El parámetro `#:jbuild?` puede proporcionarse para usar la orden `jbuild` en vez de la más reciente `dune` durante la construcción de un paquete. Su valor predeterminado es `#f`.

El parámetro `#:package` puede proporcionarse para especificar un nombre de paquete, lo que resulta útil cuando un paquete contiene múltiples paquetes y únicamente quiere construir uno de ellos. Es equivalente a proporcionar el parámetro `-p` a `dune`.

`elm-build-system` [Variable]

This variable is exported by `(guix build-system elm)`. It implements a build procedure for Elm (`https://elm-lang.org`) packages similar to '`elm install`'.

The build system adds an Elm compiler package to the set of inputs. The default compiler package (currently `elm-sans-reactor`) can be overridden using the `#:elm` argument. Additionally, Elm packages needed by the build system itself are added as implicit inputs if they are not already present: to suppress this behavior, use the `#:implicit-elm-package-inputs?` argument, which is primarily useful for bootstrapping.

The `"dependencies"` and `"test-dependencies"` in an Elm package's `elm.json` file correspond to `propagated-inputs` and `inputs`, respectively.

Elm requires a particular structure for package names: see Section 22.8.12 [Paquetes Elm], page 756, for more details, including utilities provided by `(guix build-system elm)`.

There are currently a few noteworthy limitations to `elm-build-system`:

- The build system is focused on *packages* in the Elm sense of the word: Elm *projects* which declare `{ "type": "package" }` in their `elm.json` files. Using `elm-build-system` to build Elm *applications* (which declare `{ "type": "application" }`) is possible, but requires ad-hoc modifications to the build phases. For examples, see the definitions of the `elm-todomvc` example application and the `elm` package itself (because the front-end for the '`elm reactor`' command is an Elm application).
- Elm supports multiple versions of a package coexisting simultaneously under `ELM_HOME`, but this does not yet work well with `elm-build-system`. This limitation primarily affects Elm applications, because they specify exact versions for their dependencies, whereas Elm packages specify supported version ranges.

As a workaround, the example applications mentioned above use the `patch-application-dependencies` procedure provided by `(guix build elm-build-system)` to rewrite their `elm.json` files to refer to the package versions actually present in the build environment. Alternatively, Guix package transformations (see Section 8.3 [Definición de variantes de paquetes], page 114) could be used to rewrite an application's entire dependency graph.

- We are not yet able to run tests for Elm projects because neither `elm-test-rs` (`https://github.com/mpizenberg/elm-test-rs`) nor the Node.js-based `elm-test` (`https://github.com/rtfeldman/node-test-runner`) runner has been packaged for Guix yet.

`go-build-system` [Variable]

Esta variable se exporta en `(guix build-system go)`. Implementa el procedimiento de construcción para paquetes Go usando los mecanismos de construcción de Go (`https://golang.org/cmd/go/#hdr-Compile_packages_and_dependencies`) estándares.

Se espera que la usuaria proporcione un valor para el parámetro `#:import-path` y, en algunos caso, `#:unpack-path`. La ruta de importación (`https://golang.org/doc/code.html#ImportPaths`) corresponde a la ruta del sistema de archivos esperada por los guiones de construcción del paquete y los paquetes a los que hace referencia, y proporciona una forma de hacer referencia a un paquete Go unívocamente. Está basado típicamente en una combinación de la URI remota del paquete de archivos de fuente y la estructura jerárquica del sistema de archivos. En algunos casos, necesitará desempaquetar el código fuente del paquete en una estructura de directorios diferente a la indicada en la ruta de importación, y `#:unpack-path` debe usarse en dichos casos.

Los paquetes que proporcionan bibliotecas Go deben instalar su código fuente en la salida de la construcción. El parámetro `#:install-source?`, cuyo valor por defecto es `#t`, controla si se instalará o no el código fuente. Puede proporcionarse `#f` en paquetes que proporcionan únicamente archivos ejecutables.

Packages can be cross-built, and if a specific architecture or operating system is desired then the keywords `#:goarch` and `#:goos` can be used to force the package to be built for that architecture and operating system. The combinations known to Go can be found in their documentation (`https://golang.org/doc/install/source#environment`).

The key `#:go` can be used to specify the Go compiler package with which to build the package.

`glib-or-gtk-build-system` [Variable]

Esta variable se exporta en `(guix build-system glib-or-gtk)`. Está pensada para usarse con paquetes que usan GLib o GTK+.

Este sistema de construcción añade las dos fases siguientes a las definidas en *gnu-build-system*:

`glib-or-gtk-wrap`

La fase `glib-or-gtk-wrap` se asegura de que los programas en `bin/` son capaces de encontrar los "esquemas" GLib y los módulos GTK+ (`https://developer.gnome.org/gtk3/stable/gtk-running.html`).

Esto se consigue recubriendo los programas en guiones de lanzamiento que proporcionan valores apropiados para las variables de entorno `XDG_DATA_DIRS` y `GTK_PATH`.

Es posible excluir salidas específicas del paquete del proceso de recubrimiento enumerando sus nombres en el parámetro `#:glib-org-gtk-wrap-excluded-outputs`. Esto es útil cuando se sabe que una salida no contiene binarios GLib o GTK+, y cuando empaquetar gratuitamente añadiría una dependencia de dicha salida en GLib y GTK+.

`glib-or-gtk-compile-schemas`

La fase `glib-or-gtk-compile-schemas` se asegura que todos los esquemas GSettings (`https://developer.gnome.org/gio/stable/glib-compile-schemas.html`) o GLib se compilan. La compilación la realiza el programa `glib-compile-schemas`. Lo proporciona el paquete `glib:bin` que se importa automáticamente por el sistema de construcción. El paquete `glib` que proporciona `glib-compile-schemas` puede especificarse con el parámetro `#:glib`.

Ambas fases se ejecutan tras la fase `install`.

`guile-build-system` [Variable]

Este sistema de construcción es para paquetes Guile que consisten exclusivamente en código Scheme y son tan simples que no tienen ni siquiera un archivo Makefile, menos un guión `configure`. Compila código Scheme usando `guild compile` (see Section "Compilation" in *GNU Guile Reference Manual*) e instala los archivos `.scm` y `.go` en el lugar correcto. También instala documentación.

Este sistema de construcción permite la compilación cruzada usando la opción `--target` de `guild compile`.

Los paquetes construidos con `guile-build-system` deben proporcionar un paquete Guile en su campo `native-inputs`.

`julia-build-system` [Variable]

This variable is exported by `(guix build-system julia)`. It implements the build procedure used by julia (`https://julialang.org/`) packages, which essentially is similar to running '`julia -e 'using Pkg; Pkg.add(package)'`' in an environment where `JULIA_LOAD_PATH` contains the paths to all Julia package inputs. Tests are run by calling `/test/runtests.jl`.

The Julia package name and uuid is read from the file `Project.toml`. These values can be overridden by passing the argument `#:julia-package-name` (which must be correctly capitalized) or `#:julia-package-uuid`.

Julia packages usually manage their binary dependencies via `JLLWrappers.jl`, a Julia package that creates a module (named after the wrapped library followed by `_jll.jl`.

To add the binary path `_jll.jl` packages, you need to patch the files under `src/wrappers/`, replacing the call to the macro `JLLWrappers.@generate_wrapper_header`, adding as a second argument containing the store path the binary.

As an example, in the MbedTLS Julia package, we add a build phase (see Section 8.6 [Fases de construcción], page 143) to insert the absolute file name of the wrapped MbedTLS package:

```
(add-after 'unpack 'override-binary-path
  (lambda* (#:key inputs #:allow-other-keys)
    (for-each (lambda (wrapper)
                (substitute* wrapper
                  (("generate_wrapper_header.*")
                   (string-append
                    "generate_wrapper_header(\"MbedTLS\", \""
                    (assoc-ref inputs "mbedtls") "\")\n"))))
              ;; There's a Julia file for each platform, override them all.
              (find-files "src/wrappers/" "\\.jl$"))))
```

Some older packages that aren't using `Project.toml` yet, will require this file to be created, too. It is internally done if the arguments `#:julia-package-name` and `#:julia-package-uuid` are provided.

`maven-build-system` [Variable]

Esta variable se exporta en `(guix build-system maven)`. Implementa un procedimiento de construcción para paquetes basados en Maven (`https://maven.apache.org`). Maven es una herramienta para Java de gestión de dependencias y ciclo de vida. Las usuarias de Maven especifican las dependencias y módulos en un archivo `pom.xml` que Maven lee. Cuando Maven no dispone de una de dichas dependencias o módulos en su repositorio, las descarga y las usa para construir el paquete.

El sistema de compilación de maven se asegura de que maven no intentará descargar ninguna dependencia ejecutándo maven en modo sin conexión. Maven fallará si falta alguna dependencia. Antes de ejecutar Maven, el archivo `pom.xml` (y los subproyectos) se modifican para especificar la versión de las dependencias y módulos que corresponden a las versiones disponibles en el entorno de construcción de guix. Las dependencias y los módulos se deben instalar en un repositorio de maven *ad hoc* en `lib/m2`, y se enlazan un repositorio real antes de que se ejecute maven. Se le indica a Maven que use ese repositorio para la construcción e instale los artefactos generados allí. Los archivos cambiados se copian del directorio `lib/m2` a la salida del paquete.

Puede especificar un archivo `pom.xml` con el parámetro `#:pom-file`, o dejar al sistema de construcción usar el archivo predeterminado `pom.xml` en las fuentes.

En caso de que necesite especificar la versión de una dependencia manualmente puede usar el parámetro `#:local-packages`. Toma como valor una lista asociativa donde la clave es el valor del campo "groupId" del paquete y su valor es una lista asociativa donde la clave es el campo "artifactId" del paquete y su valor la versión que quiere forzar en vez de la que se encuentra en `pom.xml`.

Algunos paquetes usan dependencias o módulos que no son útiles en tiempo de ejecución ni en tiempo de construcción en Guix. Puede modificar el archivo `pom.xml` para eliminarlos usando el parámetro `#:exclude`. Su valor es una lista asociativa donde la clave es el valor del campo "groupId" del módulo o dependencia que quiere eliminar, y su valor es una lista de valores del campo "artifactId" que se eliminarán.

Puede usar valores distintos para los paquetes `jdk` y `maven` con el parámetro correspondiente, `#:jdk` y `#:maven`.

El parámetro `#:maven-plugins` es una lista de módulos de maven usados durante la construcción, con el mismo formato que el campo `inputs` de la declaración del paquete. Su valor predeterminado es `(default-maven-plugins)` que también se exporta.

`minetest-mod-build-system` [Variable]

This variable is exported by `(guix build-system minetest)`. It implements a build procedure for Minetest (`https://www.minetest.net`) mods, which consists of copying Lua code, images and other resources to the location Minetest searches for mods. The build system also minimises PNG images and verifies that Minetest can load the mod without errors.

`minify-build-system` [Variable]

Esta variable se exporta en `(guix build-system minify)`. Implementa un procedimiento de minificación para paquetes JavaScript simples.

Añade `uglify-js` al conjunto de entradas y lo utiliza para comprimir todos los archivos JavaScript en el directorio `src`. Un paquete de minificación diferente puede especificarse con el parámetro `#:uglify-js`, pero se espera que el paquete escriba el código minificado en la salida estándar.

Cuando los archivos JavaScript de entrada no se encuentran en el directorio `src`, el parámetro `#:javascript-files` puede usarse para especificar una lista de nombres de archivo que proporcionar al minificador.

`mozilla-build-system` [Variable]

This variable is exported by `(guix build-system mozilla)`. It sets the `--target` and `--host` configuration flags to what software developed by Mozilla expects – due to historical reasons, Mozilla software expects `--host` to be the system that is cross-compiled from and `--target` to be the system that is cross-compiled to, contrary to the standard Autotools conventions.

`ocaml-build-system` [Variable]

Esta variable se exporta en `(guix build-system ocaml)`. Implementa un procedimiento de construcción para paquetes OCaml (`https://ocaml.org`), que consiste en seleccionar el conjunto correcto de órdenes a ejecutar para cada paquete. Los paquetes OCaml pueden esperar la ejecución de muchas ordenes diferentes. Este sistema de construcción probará algunas de ellas.

Cuando el paquete tiene un archivo `setup.ml` presente en el nivel superior, se ejecuta `ocaml setup.ml -configure`, `ocaml setup.ml -build` y `ocaml setup.ml -install`. El sistema de construcción asumirá que este archivo se generó con `http://oasis.forge.ocamlcore.org/ OASIS` y se encargará de establecer el prefijo y la activación de las pruebas si no se desactivaron. Puede pasar opciones de configuración y construcción con `#:configure-flags` y `#:build-flags`, respectivamente. El parámetro `#:test-flags` puede usarse para cambiar el conjunto de opciones usadas para activar las pruebas. El parámetro `#:use-make?` puede usarse para ignorar este sistema en las fases de construcción e instalación.

Cuando el paquete tiene un archivo `configure`, se asume que es un guión de configuración hecho a mano que necesita un formato de parámetros diferente a los del sistema `gnu-build-system`. Puede añadir más opciones con el parámetro `#:configure-flags`.

Cuando el paquete tiene un archivo `Makefile` (o `#:use-make?` es `#t`), será usado y se pueden proporcionar más opciones para las fases de construcción y e instalación con el parámetro `#:make-flags`.

Por último, algunos paquetes no tienen estos archivos y usan unas localizaciones de algún modo estándares para su sistema de construcción. En este caso, el sistema de construcción ejecutará `ocaml pkg/pkg.ml` o `ocaml pkg/build.ml` y se hará cargo de proporcionar la ruta del módulo findlib necesario. Se pueden pasar opciones adicionales con el parámetro `#:build-flags`. De la instalación se hace cargo `opam-installer`. En este caso, el paquete `opam` debe añadirse al campo `native-inputs` de la definición del paquete.

Fíjese que la mayoría de los paquetes OCaml asumen su instalación en el mismo directorio que OCaml, lo que no es el comportamiento deseado en guix. En particular, tratarán de instalar archivos `.so` en su directorio de módulos, lo que normalmente es aceptable puesto que está bajo el directorio del compilador de OCaml. No obstante, en guix estas bibliotecas no se pueden instalar ahí, por lo que se usa `CAML_LD_LIBRARY_PATH`. Esta variable apunta a `lib/ocaml/site-lib/stubslibs` y allí es donde se deben instalar las bibliotecas `.so`.

`python-build-system` [Variable]

Esta variable se exporta en `(guix build-system python)`. Implementa el procedimiento más o menos estándar de construcción usado por paquetes Python, que consiste en la ejecución de `python setup.py build` y `python setup.py install --prefix=/gnu/store/...`.

For packages that install stand-alone Python programs under `bin/`, it takes care of wrapping these programs so that their `GUIX_PYTHONPATH` environment variable points to all the Python libraries they depend on.

Se puede especificar el paquete Python usado para llevar a cabo la construcción con el parámetro `#:python`. Esta es habitualmente una forma de forzar la construcción de un paquete para una versión específica del intérprete Python, lo que puede ser necesario si el paquete es compatible únicamente con una versión del intérprete.

De manera predeterminada guix llama a `setup.py` bajo el control de `setuptools` de manera similar a lo realizado por `pip`. Algunos paquetes no son compatibles con setuptools (y pip), por lo que puede desactivar esta configuración estableciendo el parámetro `#:use-setuptools` a `#f`.

If a `"python"` output is available, the package is installed into it instead of the default `"out"` output. This is useful for packages that include a Python package as only a part of the software, and thus want to combine the phases of `python-build-system` with another build system. Python bindings are a common usecase.

`pyproject-build-system` [Variable]

This is a variable exported by `guix build-system pyproject`. It is based on *python-build-system*, and adds support for `pyproject.toml` and PEP 517 (`https://peps.`

python.org/pep-0517/). It also supports a variety of build backends and test frameworks.

The API is slightly different from *python-build-system*:

- `#:use-setuptools?` and `#:test-target` is removed.
- `#:build-backend` is added. It defaults to `#false` and will try to guess the appropriate backend based on `pyproject.toml`.
- `#:test-backend` is added. It defaults to `#false` and will guess an appropriate test backend based on what is available in package inputs.
- `#:test-flags` is added. The default is `'()`. These flags are passed as arguments to the test command. Note that flags for verbose output is always enabled on supported backends.

It is considered "experimental" in that the implementation details are not set in stone yet, however users are encouraged to try it for new Python projects (even those using `setup.py`). The API is subject to change, but any breaking changes in the Guix channel will be dealt with.

Eventually this build system will be deprecated and merged back into *python-build-system*, probably some time in 2024.

`perl-build-system` [Variable]

Esta variable se exporta en `(guix build-system perl)`. Implementa el procedimiento de construcción estándar para paquetes Perl, lo que o bien consiste en la ejecución de `perl Build.PL --prefix=/gnu/store/...`, seguido de `Build` y `Build install`; o en la ejecución de `perl Makefile.PL PREFIX=/gnu/store/...`, seguida de `make` y `make install`, dependiendo de si `Build.PL` o `Makefile.PL` están presentes en la distribución del paquete. El primero tiene preferencia si existen tanto `Build.PL` como `Makefile.PL` en la distribución del paquete. Esta preferencia puede ser invertida mediante la especificación del valor `#t` en el parámetro `#:make-maker?`.

La invocación inicial de `perl Makefile.PL` o `perl Build.PL` pasa a su vez las opciones especificadas por los parámetros `#:make-maker-flags` o `#:module-build-flags`, respectivamente.

El paquete Perl usado puede especificarse con `#:perl`.

`renpy-build-system` [Variable]

This variable is exported by `(guix build-system renpy)`. It implements the more or less standard build procedure used by Ren'py games, which consists of loading `#:game` once, thereby creating bytecode for it.

It further creates a wrapper script in `bin/` and a desktop entry in `share/applications`, both of which can be used to launch the game.

Which Ren'py package is used can be specified with `#:renpy`. Games can also be installed in outputs other than "out" by using `#:output`.

`qt-build-system` [Variable]

Esta variable se exporta en `(guix build-system qt)`. Está pensado para usarse con aplicaciones que usen Qt o KDE.

Este sistema de construcción añade las dos fases siguientes a las definidas en *cmake-build-system*:

`check-setup`
: La fase `check-setup` prepara el entorno para la ejecución de las comprobaciones usadas habitualmente por los programas de pruebas de Qt. Por ahora únicamente proporciona valor a algunas variables de entorno: `QT_QPA_PLATFORM=offscreen`, `DBUS_FATAL_WARNINGS=0` y `CTEST_OUTPUT_ON_FAILURE=1`.

  Esta fase se añade previamente a la fase `check`. Es una fase separada para facilitar el ajuste si fuese necesario.

`qt-wrap`
: La fase `qt-wrap` busca las rutas de módulos de Qt5, las rutas de QML y algunas rutas XDG en las entradas y la salida. En caso de que alguna ruta se encuentra, todos los programas en los directorios `bin/`, `sbin/`, `libexec/` y `lib/libexec/` de la salida se envuelven en guiones que definen las variables de entorno necesarias.

  Es posible excluir salidas específicas del paquete del proceso de recubrimiento enumerando sus nombres en el parámetro `#:qt-wrap-excluded-outputs`. Esto es útil cuando se sabe que una salida no contiene binarios que usen Qt, y cuando empaquetar gratuitamente añadiría una dependencia de dicha salida en Qt.

  Ambas fases se añaden tras la fase `install`.

`r-build-system` [Variable]
: Esta variable se exporta en `(guix build-system r)`. Implementa el procedimiento de construcción usados por paquetes R (`https://r-project.org`), lo que esencialmente es poco más que la ejecución de '`R CMD INSTALL --library=/gnu/store/...`' en un entorno donde `R_LIBS_SITE` contiene las rutas de todos los paquetes R de entrada. Las pruebas se ejecutan tras la instalación usando la función R `tools::testInstalledPackage`.

`rakudo-build-system` [Variable]
: This variable is exported by `(guix build-system rakudo)`. It implements the build procedure used by Rakudo (`https://rakudo.org/`) for Perl6 (`https://perl6.org/`) packages. It installs the package to `/gnu/store/.../NAME-VERSION/share/perl6` and installs the binaries, library files and the resources, as well as wrap the files under the `bin/` directory. Tests can be skipped by passing `#f` to the `tests?` parameter.

  El paquete rakudo en uso puede especificarse con `rakudo`. El paquete perl6-tap-harness en uso durante las pruebas puede especificarse con `#:prove6` o eliminarse proporcionando `#f` al parámetro `with-prove6?`. El paquete perl6-zef en uso durante las pruebas e instalación puede especificarse con `#:zef` o eliminarse proporcionando `#f` al parámetro `with-zef?`.

`rebar-build-system` [Variable]
: This variable is exported by `(guix build-system rebar)`. It implements a build procedure around rebar3 (`https://rebar3.org`), a build system for programs written in the Erlang language.

It adds both `rebar3` and the `erlang` to the set of inputs. Different packages can be specified with the `#:rebar` and `#:erlang` parameters, respectively.

This build system is based on `gnu-build-system`, but with the following phases changed:

`unpack` This phase, after unpacking the source like the `gnu-build-system` does, checks for a file `contents.tar.gz` at the top-level of the source. If this file exists, it will be unpacked, too. This eases handling of package hosted at `https://hex.pm/`, the Erlang and Elixir package repository.

`bootstrap`
`configure`
There are no `bootstrap` and `configure` phase because erlang packages typically don't need to be configured.

`build` This phase runs `rebar3 compile` with the flags listed in `#:rebar-flags`.

`check` Unless `#:tests? #f` is passed, this phase runs `rebar3 eunit`, or some other target specified with `#:test-target`, with the flags listed in `#:rebar-flags`,

`install` This installs the files created in the *default* profile, or some other profile specified with `#:install-profile`.

`texlive-build-system` [Variable]

Esta variable se exporta en `(guix build-system texlive)`. Se usa para construir paquetes TeX en modo de procesamiento de lotes con el motor especificado. El sistema de construcción proporciona valor a la variable `TEXINPUTS` para encontrar todos los archivos de fuentes TeX en las entradas.

By default it tries to run `luatex` on all `.ins` files, and if it fails to find any, on all `.dtx` files. A different engine and format can be specified with, respectively, the `#:tex-engine` and `#:tex-format` arguments. Different build targets can be specified with the `#:build-targets` argument, which expects a list of file names.

It also generates font metrics (i.e., `.tfm` files) out of Metafont files whenever possible. Likewise, it can also create TeX formats (i.e., `.fmt` files) listed in the `#:create-formats` argument, and generate a symbolic link from `bin/` directory to any script located in `texmf-dist/scripts/`, provided its file name is listed in `#:link-scripts` argument.

The build system adds `texlive-bin` from `(gnu packages tex)` to the native inputs. It can be overridden with the `#:texlive-bin` argument.

The package `texlive-latex-bin`, from the same module, contains most of the tools for building TeX Live packages; for convenience, it is also added by default to the native inputs. However, this can be troublesome when building a dependency of `texlive-latex-bin` itself. In this particular situation, the `#:texlive-latex-bin?` argument should be set to `#f`.

`ruby-build-system` [Variable]

Esta variable se exporta en `(guix build-system ruby)`. Implementa el procedimiento de construcción de RubyGems usado por los paquetes Ruby, que implica la ejecución de `gem build` seguida de `gem install`.

El campo `source` de un paquete que usa este sistema de construcción típicamente se refiere a un archivo gem, ya que este es el formato usado por las desarrolladoras Ruby cuando publican su software. El sistema de construcción desempaqueta el archivo gem, potencialmente parchea las fuentes, ejecuta la batería de pruebas, vuelve a empaquetar el archivo gem y lo instala. Adicionalmente, se puede hacer referencia a directorios y archivadores tar para permitir la construcción de archivos gem no publicados desde Git o un archivador tar de publicación de fuentes tradicional.

Se puede especificar el paquete Ruby usado mediante el parámetro `#:ruby`. Una lista de opciones adicionales pueden pasarse a la orden `gem` en el parámetro `#:gem-flags`.

`waf-build-system` [Variable]

Esta variable se exporta en `(guix build-system waf)`. Implementa un procedimiento de construcción alrededor del guión `waf`. Las fases comunes—`configure`, `build` y `install`—se implementan pasando sus nombres como parámetros al guión `waf`.

El guión `waf` es ejecutado por el intérprete Python. El paquete Python usado para la ejecución puede ser especificado con el parámetro `#:python`.

`zig-build-system` [Variable]

This variable is exported by `(guix build-system zig)`. It implements the build procedures for the Zig (`https://ziglang.org/`) build system (`zig build` command).

Selecting this build system adds `zig` to the package inputs, in addition to the packages of `gnu-build-system`.

There is no `configure` phase because Zig packages typically do not need to be configured. The `#:zig-build-flags` parameter is a list of flags that are passed to the `zig` command during the build. The `#:zig-test-flags` parameter is a list of flags that are passed to the `zig test` command during the `check` phase. The default compiler package can be overridden with the `#:zig` argument.

The optional `zig-release-type` parameter declares the type of release. Possible values are: `safe`, `fast`, or `small`. The default value is `#f`, which causes the release flag to be omitted from the `zig` command. That results in a `debug` build.

`scons-build-system` [Variable]

Esta variable se exporta en `(guix build-system scons)`. Implementa en procedimiento de construcción usado por la herramienta de construcción de software SCons. Este sistema de construcción ejecuta `scons` para construir el paquete, `scons test` para ejecutar las pruebas y después `scons install` para instalar el paquete.

Las opciones adicionales a pasar a `scons` se pueden especificar con el parámetro `#:scons-flags`. Los objetivos predeterminados de construcción (build) e instalación (install) pueden modificarse con `#:build-targets` y `#:install-targets` respectivamente. La versión de Python usada para ejecutar SCons puede especificarse seleccionando el paquete SCons apropiado con el parámetro `#:scons`.

`haskell-build-system` [Variable]

Esta variable se exporta en `(guix build-system haskell)`. Implementa el procedimiento de construcción Cabal usado por paquetes Haskell, el cual implica la ejecución de `runhaskell Setup.hs configure --prefix=/gnu/store/...` y `runhaskell Setup.hs build`. En vez de instalar el paquete ejecutando `runhaskell`

`Setup.hs install`, para evitar el intento de registro de bibliotecas en el directorio de solo-lectura del compilador en el almacén, el sistema de construcción usa `runhaskell Setup.hs copy`, seguido de `runhaskell Setup.hs register`. Además, el sistema de construcción genera la documentación del paquete ejecutando `runhaskell Setup.hs haddock`, a menos que se pasase `#:haddock? #f`. Parámetros opcionales de Haddock pueden proporcionarse con la ayuda del parámetro `#:haddock-flags`. Si el archivo `Setup.hs` no es encontrado, el sistema de construcción busca `Setup.lhs` a su vez.

El compilador Haskell usado puede especificarse con el parámetro `#:haskell` cuyo valor predeterminado es `ghc`.

`dub-build-system` [Variable]

Esta variable se exporta en `(guix build-system dub)`. Implementa el procedimiento de construcción Dub usado por los paquetes D, que implica la ejecución de `dub build` y `dub run`. La instalación se lleva a cabo con la copia manual de los archivos.

El compilador D usado puede ser especificado con el parámetro `#:ldc` cuyo valor predeterminado es `ldc`.

`emacs-build-system` [Variable]

Esta variable se exporta en `(guix build-system emacs)`. Implementa un procedimiento de instalación similar al propio sistema de empaquetado de Emacs (see Section "Packages" in *The GNU Emacs Manual*).

Primero crea el archivo *`paquete`*`-autoloads.el`, tras lo que compila todos los archivos Emacs Lisp. De manera diferente al sistema de paquetes de Emacs, los archivos de documentación Info se mueven al directorio estándar de documentación y se borra el archivo `dir`. Los archivos del paquete Elisp se instalan directamente en `share/emacs/site-lisp`.

`font-build-system` [Variable]

Esta variable se exporta en `(guix build-system font)`. Implementa un procedimiento de instalación para paquetes de fuentes donde las proveedoras originales proporcionan archivos de tipografía TrueType, OpenType, etc. precompilados que simplemente necesitan copiarse en su lugar. Copia los archivos de tipografías a las localizaciones estándar en el directorio de salida.

`meson-build-system` [Variable]

Esta variable se exporta en `(guix build-system meson)`. Implementa el procedimiento de construcción para paquetes que usan Meson (`https://mesonbuild.com`) como su sistema de construcción.

It adds both Meson and Ninja (`https://ninja-build.org/`) to the set of inputs, and they can be changed with the parameters `#:meson` and `#:ninja` if needed.

Este sistema de construcción es una extensión de *gnu-build-system*, pero con las siguientes fases cambiadas por otras específicas para Meson:

`configure`

Esta fase ejecuta `meson` con las opciones especificadas en `#:configure-flags`. La opción `--buildtype` recibe el valor `debugoptimized` excepto cuando se especifique algo distinto en `#:build-type`.

`build` Esta fase ejecuta `ninja` para construir el paquete en paralelo por defecto, pero esto puede cambiarse con `#:parallel-build?`.

`check` The phase runs '`meson test`' with a base set of options that cannot be overridden. This base set of options can be extended via the `#:test-options` argument, for example to select or skip a specific test suite.

`install` Esta fase ejecuta `ninja install` y no puede cambiarse.

Aparte de estas, el sistema de ejecución también añade las siguientes fases:

`fix-runpath`
This phase ensures that all binaries can find the libraries they need. It searches for required libraries in subdirectories of the package being built, and adds those to `RUNPATH` where needed. It also removes references to libraries left over from the build phase by `meson`, such as test dependencies, that aren't actually required for the program to run.

`glib-or-gtk-wrap`
Esta fase es la fase proporcionada por `glib-or-gtk-build-system`, y no está activa por defecto. Puede activarse con `#:glib-or-gtk`.

`glib-or-gtk-compile-schemas`
Esta fase es la fase proporcionada por `glib-or-gtk-build-system`, y no está activa por defecto. Puede activarse con `#:glib-or-gtk`.

`linux-module-build-system` [Variable]
*linux-module-build-system* permite la construcción de módulos del núcleo Linux.

Este sistema de construcción es una extensión de *gnu-build-system*, pero con las siguientes fases cambiadas:

`configure`
Esta fase configura el entorno de modo que el Makefile del núcleo Linux pueda usarse para la construcción del módulo externo del núcleo.

`build` Esta fase usa el Makefile del núcleo Linux para construir el módulo externo del núcleo.

`install` Esta fase usa el Makefile del núcleo Linux para instalar el módulo externo del núcleo.

Es posible y útil especificar el núcleo Linux usado para la construcción del módulo (para ello debe usar el parámetro `#:linux` a través de la forma `arguments` en un paquete que use `linux-module-build-system`).

`node-build-system` [Variable]
Esta variable se exporta en `(guix build-system node)`. Implementa el procedimiento de construcción usado por Node.js (`https://nodejs.org`), que implementa una aproximación de la orden `npm install`, seguida de una orden `npm test`.

El paquete Node.js usado para interpretar las órdenes `npm` puede especificarse a través del parámetro `#:node` cuyo valor predeterminado es `node`.

`tree-sitter-build-system` [Variable]

This variable is exported by `(guix build-system tree-sitter)`. It implements procedures to compile grammars for the Tree-sitter (`https://tree-sitter.github.io/tree-sitter/`) parsing library. It essentially runs `tree-sitter generate` to translate `grammar.js` grammars to JSON and then to C. Which it then compiles to native code.

Tree-sitter packages may support multiple grammars, so this build system supports a `#:grammar-directories` keyword to specify a list of locations where a `grammar.js` file may be found.

Grammars sometimes depend on each other, such as C++ depending on C and TypeScript depending on JavaScript. You may use inputs to declare such dependencies.

Por último, para paquetes que no necesiten nada tan sofisticado se proporciona un sistema de construcción "trivial". Es trivial en el sentido de que no proporciona prácticamente funcionalidad: no incorpora entradas implícitas y no tiene una noción de fases de construcción.

`trivial-build-system` [Variable]

Esta variable se exporta en `(guix build-system trivial)`.

Este sistema de construcción necesita un parámetro `#:builder`. Este parámetro debe ser una expresión Scheme que construya la(s) salida(s) del paquete—como en `build-expression->derivation` (see Section 8.10 [Derivaciones], page 159).

`channel-build-system` [Variable]

This variable is exported by `(guix build-system channel)`.

This build system is meant primarily for internal use. A package using this build system must have a channel specification as its `source` field (see Chapter 6 [Canales], page 69); alternatively, its source can be a directory name, in which case an additional `#:commit` argument must be supplied to specify the commit being built (a hexadecimal string).

Optionally, a `#:channels` argument specifying additional channels can be provided.

The resulting package is a Guix instance of the given channel(s), similar to how `guix time-machine` would build it.

## 8.6 Fases de construcción

Prácticamente todos los sistemas de construcción de paquetes implementan una noción de *fases de construcción*: una secuencia de acciones ejecutadas por el sistema de construcción, cuando usted construya el paquete, que conducen a la instalación de su producción en el almacén. Una excepción notable es el sistema de construcción trivial `trivial-build-system` (see Section 8.5 [Sistemas de construcción], page 123).

As discussed in the previous section, those build systems provide a standard list of phases. For `gnu-build-system`, the main build phases are the following:

`set-paths`

Define search path environment variables for all the input packages, including `PATH` (see Section 8.8 [Search Paths], page 154).

`unpack` Extrae el archivador tar de la fuente, y cambia el directorio actual al directorio recién extraído. Si la fuente es realmente un directorio, lo copia al árbol de construcción y entra en ese directorio.

`patch-source-shebangs`
Sustituye secuencias "#!" encontradas al inicio de los archivos de fuentes para que hagan referencia a los nombres correctos de archivos del almacén. Por ejemplo, esto cambia `#!/bin/sh` por `#!/gnu/store/...-bash-4.3/bin/sh`.

`configure`
Ejecuta el guión `configure` con algunas opciones predeterminadas, como `--prefix=/gnu/store/...`, así como las opciones especificadas por el parámetro `#:configure-flags`.

`build` Ejecuta `make` con la lista de opciones especificadas en `#:make-flags`. Si el parámetro `#:parallel-build?` es verdadero (por defecto), construye con `make -j`.

`check` Ejecuta `make check`, u otro objetivo especificado con `#:test-target`, a menos que se pasase `#:tests? #f`. Si el parámetro `#:parallel-tests?` es verdadero (por defecto), ejecuta `make check -j`.

`install` Ejecuta `make install` con las opciones enumeradas en `#:make-flags`.

`patch-shebangs`
Sustituye las secuencias "#!" en los archivos ejecutables instalados.

`strip` Extrae los símbolos de depuración de archivos ELF (a menos que el valor de `#:strip-binaries?` sea falso), y los copia a la salida `debug` cuando esté disponible (see Chapter 17 [Instalación de archivos de depuración], page 718).

`validate-runpath`
Validate the `RUNPATH` of ELF binaries, unless `#:validate-runpath?` is false (see Section 8.5 [Sistemas de construcción], page 123).

This validation step consists in making sure that all the shared libraries needed by an ELF binary, which are listed as `DT_NEEDED` entries in its `PT_DYNAMIC` segment, appear in the `DT_RUNPATH` entry of that binary. In other words, it ensures that running or using those binaries will not result in a "file not found" error at run time. See Section "Options" in *The GNU Linker*, for more information on `RUNPATH`.

Other build systems have similar phases, with some variations. For example, `cmake-build-system` has same-named phases but its `configure` phases runs `cmake` instead of `./configure`. Others, such as `python-build-system`, have a wholly different list of standard phases. All this code runs on the *build side*: it is evaluated when you actually build the package, in a dedicated build process spawned by the build daemon (see Section 2.3 [Invocación de guix-daemon], page 12).

Las fases de construcción se representan como listas asociativas o "alists" (see Section "Association Lists" in *GNU Guile Reference Manual*) donde cada clave es un símbolo que nombra a la fase y el valor asociado es un procedimiento que acepta un número arbitrario de parámetros. Por convención, estos procedimientos reciben información sobre la construcción en forma de *parámetros que usan palabras clave*, de los cuales pueden hacer uso o ignorarlos.

Por ejemplo, esta es la forma en la que `(guix build gnu-build-system)` define `%standard-phases`, la variable que contiene su lista asociativa de fases de construcción[3]:

```
;; Las fases de construcción de 'gnu-build-system'.

(define* (unpack #:key source #:allow-other-keys)
  ;; Extracción del archivador tar de las fuentes.
  (invoke "tar" "xvf" source))

(define* (configure #:key outputs #:allow-other-keys)
  ;; Ejecución del guión 'configure'. Instalación de la salida
  ;; en "out".
  (let ((out (assoc-ref outputs "out")))
    (invoke "./configure"
            (string-append "--prefix=" out))))

(define* (build #:allow-other-keys)
  ;; Compilación.
  (invoke "make"))

(define* (check #:key (test-target "check") (tests? #true)
                #:allow-other-keys)
  ;; Ejecución de la batería de pruebas.
  (if tests?
      (invoke "make" test-target)
      (display "test suite not run\n")))

(define* (install #:allow-other-keys)
  ;; Instalación de los archivos en el prefijo especificado
  ;; al guión 'configure'.
  (invoke "make" "install"))

(define %standard-phases
  ;; La lista de las fases estándar (algunas se omiten por
  ;; brevedad). Cada elemento es un par símbolo/procedimiento.
  (list (cons 'unpack unpack)
        (cons 'configure configure)
        (cons 'build build)
        (cons 'check check)
        (cons 'install install)))
```

Aquí se muestra como `%standard-phases` se define como una lista de pares símbolo/procedimiento (see Section "Pairs" in *GNU Guile Reference Manual*). El primer par asocia el procedimiento `unpack` con el símbolo `unpack`—un nombre; el segundo par define de manera similar la fase `configure`, etcétera. Cuando se construye un paquete que usa `gnu-build-system`, con su lista predeterminada de fases, estas fases se ejecutan de

[3] Presentamos una visión simplificada de las fases de dichas construcción, ¡eche un vistazo al módulo `(guix build gnu-build-system)` para ver todos los detalles!

manera secuencial. Puede ver el nombre de cada fase a su inicio y tras su finalización en el registro de construcción de los paquetes que construya.

Echemos un vistazo a los propios procedimientos. Cada uno se define con `define*`: `#:key` enumera parámetros con palabras clave que el procedimiento acepta, con la posibilidad de proporcionar un valor predeterminado, y `#:allow-other-keys` especifica que se ignora cualquier otra palabra clave en los parámetros (see Section "Optional Arguments" in *GNU Guile Reference Manual*).

El procedimiento `unpack` utiliza el valor proporcionado al parámetro `source`, usado por el sistema de construcción usa para proporcionar el nombre de archivo del archivador de fuentes (o la copia de trabajo del sistema de control de versiones), e ignora otros parámetros. La fase `configure` únicamente tiene en cuenta el parámetro `outputs`, una lista asociativa de nombres de salida de paquetes con su nombre de archivo en el almacén (see Section 5.4 [Paquetes con múltiples salidas], page 51). Para ello se extrae el nombre de archivo de `out`, la salida predeterminada, y se lo proporciona a la orden `./configure` como el prefijo de la instalación (`./configure --prefix=`*`out`*), lo que significa que `make install` acabará copiando todos los archivos en dicho directorio (see Section "Configuration" in *GNU Coding Standards*). Tanto `build` como `install` ignoran todos los parámetros. `check` utiliza el parámetro `test-target`, que especifica el nombre del objetivo del archivo Makefile que debe ejecutarse para ejecutar las pruebas; se imprime un mensaje y se omiten las pruebas cuando el parámetro `tests?` tiene falso como valor.

Se puede cambiar con el parámetro `#:phases` la lista de fases usada por el sistema de construcción para un paquete en particular. El cambio del conjunto de fases de construcción se realiza mediante la construcción de una nueva lista asociativa basada en la lista asociativa `%standard-phases` descrita previamente. Esto se puede llevar a cabo con procedimientos estándar para la manipulación de listas asociativas como `alist-delete` (see Section "SRFI-1 Association Lists" in *GNU Guile Reference Manual*); no obstante es más conveniente hacerlo con `modify-phases` (see Section 8.7 [Utilidades de construcción], page 147).

Aquí se encuentra un ejemplo de una definición de paquete que borra la fase `configure` de `%standard-phases` e inserta una nueva fase antes de la fase `build`, llamada `proporciona-prefijo-en-makefile`:

```
(define-public example
  (package
    (name "example")
    ;; other fields omitted
    (build-system gnu-build-system)
    (arguments
     (list
      #:phases
      #~(modify-phases %standard-phases
          (delete 'configure)
          (add-before 'build 'set-prefix-in-makefile
            (lambda* (#:key inputs #:allow-other-keys)
              ;; Modify the makefile so that its
              ;; 'PREFIX' variable points to #$output and
              ;; 'XMLLINT' points to the correct path.
              (substitute* "Makefile"
```

```
(("PREFIX =.*")
 (string-append "PREFIX = " #$output "\n"))
(("XMLLINT =.*")
 (string-append "XMLLINT = "
                (search-input-file inputs "/bin/xmllint")
                "\n")))))))))))
```

The new phase that is inserted is written as an anonymous procedure, introduced with `lambda*`; it looks for the `xmllint` executable under a `/bin` directory among the package's inputs (see Section 8.2.1 [Referencia de package], page 105). It also honors the `outputs` parameter we have seen before. See Section 8.7 [Utilidades de construcción], page 147, for more about the helpers used by this phase, and for more examples of `modify-phases`.

> **Tip:** You can inspect the code associated with a package's `#:phases` argument interactively, at the REPL (see Section 8.14 [Using Guix Interactively], page 178).

Tenga en cuenta que las fases de construcción son código que se evalúa cuando se realiza la construcción real del paquete. Esto explica por qué la expresión `modify-phases` al completo se encuentra escapada (viene precedida de `'`, un apóstrofe): se ha *preparado* para una ejecución posterior. See Section 8.12 [Expresiones-G], page 167, para obener una explicación sobre esta preparación del código para las distintas fases de ejecución y los distintos *estratos de código* implicados.

## 8.7 Utilidades de construcción

En cuanto empiece a escribir definiciones de paquete no-triviales (see Section 8.2 [Definición de paquetes], page 102) u otras acciones de construcción (see Section 8.12 [Expresiones-G], page 167), es probable que empiece a buscar funciones auxiliares parecidas a las habituales en el intérprete de ordenes—creación de directorios, borrado y copia recursiva de archivos, manipulación de fases de construcción, etcétera. El módulo `(guix build utils)` proporciona dichos procedimientos auxiliares.

La mayoría de sistemas de construcción cargan `(guix build utils)` (see Section 8.5 [Sistemas de construcción], page 123). Por tanto, cuando construya fases de construcción personalizadas para sus definiciones de paquetes, habitualmente puede asumir que dichos procedimientos ya han sido incorporados al ámbito de ejecución.

Cuando escriba G-expressions, puede importar `(guix build utils)` en el "lado de la construcción" mediante el uso `with-imported-modules` e importandolos al ámbito actual con la forma sintáctica `use-modules` (see Section "Using Guile Modules" in *GNU Guile Reference Manual*):

```
(with-imported-modules '((guix build utils))  ; Se importa.
  (computed-file "empty-tree"
                 #~(begin
                     ;; Se añade al ámbito actual.
                     (use-modules (guix build utils))

                     ;; Se usa su procedimiento 'mkdir-p'.
                     (mkdir-p (string-append #$output "/a/b/c")))))
```

El resto de esta sección es la referencia de la mayoría de las procedimientos de utilidad proporcionados por `(guix build utils)`.

### 8.7.1 Tratamiento de nombres de archivo del almacén

Esta sección documenta procedimientos para el manejo de nombres de archivo del almacén.

`%store-directory` [Procedimiento]
Devuelve el nombre del directorio del almacén.

`store-file-name?` *archivo* [Procedimiento]
Devuelve verdadero si *archivo* está en el almacén.

`strip-store-file-name` *archivo* [Procedimiento]
Elimina `/gnu/store` y el hash de *archivo*, un nombre de archivo del almacén. El resultado es habitualmente una cadena "*paquete-versión*".

`package-name>name+version` *nombre* [Procedimiento]
Cuando se proporciona *nombre*, un nombre de paquete como `"foo-0.9.1b"`, devuelve dos valores: `"foo"` y `"0.9.1b"`. Cuando la perte de la versión no está disponible, se devuelve *nombre* y `#f`. Se considera que el primer guión seguido de un dígito introduce la parte de la versión.

### 8.7.2 Tipos de archivo

Los siguientes procedimientos tratan con archivos y tipos de archivos.

`directory-exists?` *dir* [Procedimiento]
Devuelve `#t` si *dir* existe y es un directorio.

`executable-file` *archivo* [Procedimiento]
Devuelve `#t` si *archivo* existe y es ejecutable.

`symbolic-link?` *archivo* [Procedimiento]
Devuelve `#t` si *archivo* es enlace simbólico ("symlink").

`elf-file?` *archivo* [Procedimiento]
`ar-file?` *archivo* [Procedimiento]
`gzip-file?` *archivo* [Procedimiento]
Devuelve `#t` si *archivo* es, respectivamente, un archivo ELF, un archivador `ar` (como una biblioteca estática `.a`), o un archivo comprimido con gzip.

`reset-gzip-timestamp` *archivo* [*#:keep-mtime? #t*] [Procedimiento]
Si *archivo* es un archivo gzip, reinicia su marca de tiempo embebida (como con `gzip --no-name`) y devuelve un valor verdadero. En otro caso devuelve `#f`. Cuando *keep-mtime?* es verdadero, se preserva el tiempo de modificación del *archivo*.

### 8.7.3 Manipulación de archivos

Los siguientes procedimientos y macros sirven de ayuda en la creación, modificación y borrado de archivos. Proporcionan funcionalidades comparables con las herrambientas comunes del intérprete de órdenes como `mkdir -p`, `cp -r`, `rm -r` y `sed`. Sirven de complemento a la interfaz de sistema de archivos de Guile, la cual es extensa pero de bajo nivel (see Section "POSIX" in *GNU Guile Reference Manual*).

`with-directory-excursion` *directorio cuerpo . . .* [Macro]
Ejecuta *cuerpo* con *directorio* como el directorio actual del proceso.

Esecialmente este macro cambia el directorio actual a *directorio* antes de evaluar *cuerpo*, usando `chdir` (see Section "Processes" in *GNU Guile Reference Manual*). Se vuelve al directorio inicial en cuanto se abandone el ámbito dinámico de *cuerpo*, ya sea a través de su finalización normal o de una salida no-local como pueda ser una excepción.

`mkdir-p` *dir* [Procedimiento]
Crea el directorio *dir* y todos sus predecesores.

`install-file` *file directorio* [Procedimiento]
Crea *directorio* si no existe y copia *archivo* allí con el mismo nombre.

`make-file-writable` *archivo* [Procedimiento]
Activa el permiso de escritura en *archivo* para su propietaria.

`copy-recursively` *fuente destino* [*#:log* (*current-output-port*)] [Procedimiento]
[*#:follow-symlinks? #f*] [*#:copy-file*
copy-file] [#:keep-mtime? #f] [#:keep-permissions? #t] Copy *source* directory to *destination*. Follow symlinks if *follow-symlinks?* is true; otherwise, just preserve them. Call *copy-file* to copy regular files. When *keep-mtime?* is true, keep the modification time of the files in *source* on those of *destination*. When *keep-permissions?* is true, preserve file permissions. Write verbose output to the *log* port.

`delete-file-recursively` *dir* [*#:follow-mounts? #f*] [Procedimiento]
Borra *dir* recursivamente, como `rm -rf`, sin seguir los enlaces simbólicos. No atraviesa puntos de montaje tampoco, a no ser que *follow-mounts?* sea verdadero. Informa de los errores, pero no devuelve error por ellos.

`substitute*` *archivo* ((*expreg var-encontrada. . .*) *cuerpo. . .*) . . . [Macro]
Sustituye *expreg* en *archivo* con la cadena que devuelve *cuerpo*. La evaluación de *cuerpo* se realiza con cada *var-encontrada* asociada con la subexpresión posicional correspondiente de la expresión regular. Por ejemplo:

```
(substitute* archivo
  (("hola")
   "buenos días\n")
  (("algo([a-z]+)otro(.*)$" todo letras fin)
   (string-append "cosa" letras fin)))
```

En este ejemplo, cada ver que una línea de *archivo* contiene `hola`, esto se sustituye por `buenos días`. Cada vez que una línea del *archivo* corresponde con la segunda expresión regular, `todo` se asocia con la cadena encontrada al completo, `letras` toma el valor de la primera sub-expresión, y `fin` se asocia con la última.

Cuando una de las *var-encontrada* es `_`, no se asocia ninguna variable con la correspondiente subcadena.

También puede proporcionarse una lista como *archivo*, en cuyo caso los nombres de archivo que contenga serán los que se sometan a las sustituciones.

Be careful about using `$` to match the end of a line; by itself it won't match the terminating newline of a line. For example, to match a whole line ending with a backslash, one needs a regex like `"(.*)\\\\\n$"`.

### 8.7.4 Búsqueda de archivos

Esta sección documenta procedimientos de búsqueda y filtrado de archivos.

`file-name-predicate` *expreg* [Procedimiento]
Devuelve un predicado que devuelve un valor verdadero cuando el nombre del archivo proporcionado sin la parte del directorio corresponde con *expreg*.

`find-files` *dir* [*pred*] [*#:stat lstat*] [*#:directories? #f*] [*#:fail-on-error? #f*] [Procedimiento]
Devuelve una lista ordenada lexicográficamente de los archivos que se encuentran en *dir* para los cuales *pred* devuelve verdadero. Se le proporcionan dos parámetros a *pred*: la ruta absoluta del archivo y su búfer de stat; el predicado predeteminado siempre devuelve verdadero. *pred* también puede ser una expresión regular, en cuyo caso es equivalente a escribir `(file-name-predicate pred)`. *stat* se usa para obtener información del archivo; el uso de `lstat` significa que no se siguen los enlaces simbólicos. Si *directories?* es verdadero se incluyen también los directorios. Si *fail-on-error?* es verdadero, se emite una excepción en caso de error.

Aquí se pueden encontrar algunos ejemplos en los que se asume que el directorio actual es la raíz del arbol de fuentes de Guix:

```
;; Enumera todos los archivos regulares en el directorio actual.
(find-files ".")
⇒ ("./.dir-locals.el" "./.gitignore" ...)

;; Enumera todos los archivos .scm en gnu/services.
(find-files "gnu/services" "\\.scm$")
⇒ ("gnu/services/admin.scm" "gnu/services/audio.scm" ...)

;; Enumera los archivos ar en el directorio actual.
(find-files "." (lambda (file stat) (ar-file? file)))
⇒ ("./libformat.a" "./libstore.a" ...)
```

`which` *programa* [Procedimiento]
Devuelve el nombre de archivo completo para *programa* tal y como se encuentra en `$PATH`, o `#f` si no se ha encontrado *programa*.

`search-input-file` *inputs name* [Procedure]
`search-input-directory` *inputs name* [Procedure]
Return the complete file name for *name* as found in *inputs*; `search-input-file` searches for a regular file and `search-input-directory` searches for a directory. If *name* could not be found, an exception is raised.

Here, *inputs* must be an association list like `inputs` and `native-inputs` as available to build phases (see Section 8.6 [Fases de construcción], page 143).

Here is a (simplified) example of how `search-input-file` is used in a build phase of the `wireguard-tools` package:

```
(add-after 'install 'wrap-wg-quick
  (lambda* (#:key inputs outputs #:allow-other-keys)
    (let ((coreutils (string-append (assoc-ref inputs "coreutils")
                                    "/bin")))
      (wrap-program (search-input-file outputs "bin/wg-quick")
        #:sh (search-input-file inputs "bin/bash")
        `("PATH" ":" prefix ,(list coreutils))))))
```

### 8.7.5 Program Invocation

You'll find handy procedures to spawn processes in this module, essentially convenient wrappers around Guile's `system*` (see Section "Processes" in *GNU Guile Reference Manual*).

`invoke` *program args...* [Procedure]

Invoke *program* with the given *args*. Raise an `&invoke-error` exception if the exit code is non-zero; otherwise return `#t`.

The advantage compared to `system*` is that you do not need to check the return value. This reduces boilerplate in shell-script-like snippets for instance in package build phases.

`invoke-error?` *c* [Procedure]

Return true if *c* is an `&invoke-error` condition.

`invoke-error-program` *c* [Procedure]
`invoke-error-arguments` *c* [Procedure]
`invoke-error-exit-status` *c* [Procedure]
`invoke-error-term-signal` *c* [Procedure]
`invoke-error-stop-signal` *c* [Procedure]

Access specific fields of *c*, an `&invoke-error` condition.

`report-invoke-error` *c* [*port*] [Procedure]

Report to *port* (by default the current error port) about *c*, an `&invoke-error` condition, in a human-friendly way.

Typical usage would look like this:

```
(use-modules (srfi srfi-34) ;for 'guard'
             (guix build utils))

(guard (c ((invoke-error? c)
           (report-invoke-error c)))
  (invoke "date" "--imaginary-option"))

⊣ command "date" "--imaginary-option" failed with status 1
```

`invoke/quiet` *program args...* [Procedure]

Invoke *program* with *args* and capture *program*'s standard output and standard error. If *program* succeeds, print nothing and return the unspecified value; otherwise, raise a `&message` error condition that includes the status code and the output of *program*.

Here's an example:

```
(use-modules (srfi srfi-34) ;for 'guard'
             (srfi srfi-35) ;for 'message-condition?'
             (guix build utils))

(guard (c ((message-condition? c)
           (display (condition-message c))))
  (invoke/quiet "date")  ;all is fine
  (invoke/quiet "date" "--imaginary-option"))

⊣ 'date --imaginary-option' exited with status 1; output follows:

    date: unrecognized option '--imaginary-option'
    Try 'date --help' for more information.
```

### 8.7.6 Fases de construcción

`(guix build utils)` también contiene herramientas para la manipulación de las fases de construcción usadas por los sistemas de construcción (see Section 8.5 [Sistemas de construcción], page 123). Las fases de construcción se representan como listas asociativas o "alists" (see Section "Association Lists" in *GNU Guile Reference Manual*) donde cada clave es un símbolo que nombra a la fase, y el valor asociado es un procedimiento (see Section 8.6 [Fases de construcción], page 143).

Tanto el propio Guile como el módulo `(srfi srfi-1)` proporcionan herramientas para la manipulación de listas asociativas. El módulo `(guix build utils)` complementa estas con herramientas pensadas para las fases de construcción.

`modify-phases` *fases cláusula...* [Macro]

Modifica *fases* de manera secuencial com cada indique cada *cláusula*, que puede tener una de las siguentes formas:

```
(delete nombre-fase)
(replace nombre-fase nueva-fase)
(add-before nombre-fase nombre-nueva-fase nueva-fase)
(add-after nombre-fase nombre-nueva-fase nueva-fase)
```

Donde cada *nombre-fase* y *nombre-nueva-fase* es una expresión que evalúa aun símbolo, y *nueva-fase* es una expresión que evalúa a un procedimiento.

El siguiente ejemplo se ha tomado de la definición del paquete `grep`. Añade una fase que se ejecuta tras la fase `install`, llamada `fix-egrep-and-fgrep`. Dicha fase es un procedimiento (`lambda*` genera procedimientos anónimos) que toma un parámetro de palabra clave `#:outputs` e ignora el resto (see Section "Optional Arguments" in *GNU Guile Reference Manual* para obtener más información sobre `lambda*` y los parámetros opcionales y de palabras clave). La fase usa `substitute*` para modificar los guiones `egrep` y `fgrep` instalados para que hagan referencia a `grep` con su ruta de archivo absoluta:

```
(modify-phases %standard-phases
  (add-after 'install 'fix-egrep-and-fgrep
    ;; Patch 'egrep' and 'fgrep' to execute 'grep' via its
```

```
;; absolute file name instead of searching for it in $PATH.
(lambda* (#:key outputs #:allow-other-keys)
  (let* ((out (assoc-ref outputs "out"))
         (bin (string-append out "/bin")))
    (substitute* (list (string-append bin "/egrep")
                       (string-append bin "/fgrep"))
      (("^exec grep")
       (string-append "exec " bin "/grep")))))))
```

En el siguiente ejemplo se modifican las fases de dos maneras: se elimina la fase estándar `configure`, posiblemente porque el paquete no tiene un guión `configure` ni nada similar, y la fase `install` predeterminada se sustituye por una que copia manualmente el archivo ejecutable que se debe instalar:

```
(modify-phases %standard-phases
  (delete 'configure)      ;no 'configure' script
  (replace 'install
    (lambda* (#:key outputs #:allow-other-keys)
      ;; The package's Makefile doesn't provide an "install"
      ;; rule so do it by ourselves.
      (let ((bin (string-append (assoc-ref outputs "out")
                                "/bin")))
        (install-file "footswitch" bin)
        (install-file "scythe" bin)))))
```

### 8.7.7 Wrappers

It is not unusual for a command to require certain environment variables to be set for proper functioning, typically search paths (see Section 8.8 [Search Paths], page 154). Failing to do that, the command might fail to find files or other commands it relies on, or it might pick the "wrong" ones—depending on the environment in which it runs. Examples include:

- a shell script that assumes all the commands it uses are in `PATH`;
- a Guile program that assumes all its modules are in `GUILE_LOAD_PATH` and `GUILE_LOAD_COMPILED_PATH`;
- a Qt application that expects to find certain plugins in `QT_PLUGIN_PATH`.

For a package writer, the goal is to make sure commands always work the same rather than depend on some external settings. One way to achieve that is to *wrap* commands in a thin script that sets those environment variables, thereby ensuring that those run-time dependencies are always found. The wrapper would be used to set `PATH`, `GUILE_LOAD_PATH`, or `QT_PLUGIN_PATH` in the examples above.

To ease that task, the `(guix build utils)` module provides a couple of helpers to wrap commands.

`wrap-program` *program* [*#:sh sh*] [*#:rest variables*] [Procedure]

Make a wrapper for *program*. *variables* should look like this:

```
'(variable delimiter position list-of-directories)
```

where *delimiter* is optional. `:` will be used if *delimiter* is not given.

For example, this call:

```
(wrap-program "foo"
              '("PATH" ":" = ("/gnu/.../bar/bin"))
              '("CERT_PATH" suffix ("/gnu/.../baz/certs"
                                    "/qux/certs")))
```

will copy `foo` to `.foo-real` and create the file `foo` with the following contents:

```
#!location/of/bin/bash
export PATH="/gnu/.../bar/bin"
export CERT_PATH="$CERT_PATH${CERT_PATH:+:}/gnu/.../baz/certs:/qux/certs"
exec -a $0 location/of/.foo-real "$@"
```

If *program* has previously been wrapped by `wrap-program`, the wrapper is extended with definitions for *variables*. If it is not, *sh* will be used as the interpreter.

`wrap-script` *program* [*#:guile guile*] [*#:rest variables*] [Procedure]

Wrap the script *program* such that *variables* are set first. The format of *variables* is the same as in the `wrap-program` procedure. This procedure differs from `wrap-program` in that it does not create a separate shell script. Instead, *program* is modified directly by prepending a Guile script, which is interpreted as a comment in the script's language.

Special encoding comments as supported by Python are recreated on the second line.

Note that this procedure can only be used once per file as Guile scripts are not supported.

## 8.8 Search Paths

Many programs and libraries look for input data in a *search path*, a list of directories: shells like Bash look for executables in the command search path, a C compiler looks for `.h` files in its header search path, the Python interpreter looks for `.py` files in its search path, the spell checker has a search path for dictionaries, and so on.

Search paths can usually be defined or overridden *via* environment variables (see Section "Environment Variables" in *The GNU C Library Reference Manual*). For example, the search paths mentioned above can be changed by defining the `PATH`, `C_INCLUDE_PATH`, `PYTHONPATH` (or `GUIX_PYTHONPATH`), and `DICPATH` environment variables—you know, all these something-PATH variables that you need to get right or things "won't be found".

You may have noticed from the command line that Guix "knows" which search path environment variables should be defined, and how. When you install packages in your default profile, the file `~/.guix-profile/etc/profile` is created, which you can "source" from the shell to set those variables. Likewise, if you ask `guix shell` to create an environment containing Python and NumPy, a Python library, and if you pass it the `--search-paths` option, it will tell you about `PATH` and `GUIX_PYTHONPATH` (see Section 7.1 [Invoking guix shell], page 79):

```
$ guix shell python python-numpy --pure --search-paths
export PATH="/gnu/store/...-profile/bin"
export GUIX_PYTHONPATH="/gnu/store/...-profile/lib/python3.9/site-packages"
```

When you omit `--search-paths`, it defines these environment variables right away, such that Python can readily find NumPy:

```
$ guix shell python python-numpy -- python3
Python 3.9.6 (default, Jan  1 1970, 00:00:01)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.version.version
'1.20.3'
```

For this to work, the definition of the `python` package *declares* the search path it cares about and its associated environment variable, `GUIX_PYTHONPATH`. It looks like this:

```
(package
  (name "python")
  (version "3.9.9")
  ;; some fields omitted...
  (native-search-paths
   (list (search-path-specification
          (variable "GUIX_PYTHONPATH")
          (files (list "lib/python/3.9/site-packages"))))))
```

What this `native-search-paths` field says is that, when the `python` package is used, the `GUIX_PYTHONPATH` environment variable must be defined to include all the `lib/python/3.9/site-packages` sub-directories encountered in its environment. (The `native-` bit means that, if we are in a cross-compilation environment, only native inputs may be added to the search path; see Section 8.2.1 [Referencia de package], page 105.) In the NumPy example above, the profile where `python` appears contains exactly one such sub-directory, and `GUIX_PYTHONPATH` is set to that. When there are several `lib/python/3.9/site-packages`—this is the case in package build environments—they are all added to `GUIX_PYTHONPATH`, separated by colons (`:`).

> **Nota:** Notice that `GUIX_PYTHONPATH` is specified as part of the definition of the `python` package, and *not* as part of that of `python-numpy`. This is because this environment variable "belongs" to Python, not NumPy: Python actually reads the value of that variable and honors it.
>
> Corollary: if you create a profile that does not contain `python`, `GUIX_PYTHONPATH` will *not* be defined, even if it contains packages that provide `.py` files:
>
> ```
> $ guix shell python-numpy --search-paths --pure
> export PATH="/gnu/store/...-profile/bin"
> ```
>
> This makes a lot of sense if we look at this profile in isolation: no software in this profile would read `GUIX_PYTHONPATH`.

Of course, there are many variations on that theme: some packages honor more than one search path, some use separators other than colon, some accumulate several directories in their search path, and so on. A more complex example is the search path of libxml2: the value of the `XML_CATALOG_FILES` environment variable is space-separated, it must contain a list of `catalog.xml` files (not directories), which are to be found in `xml` sub-directories—nothing less. The search path specification looks like this:

```
(package
  (name "libxml2")
  ;; some fields omitted
  (native-search-paths
   (list (search-path-specification
          (variable "XML_CATALOG_FILES")
          (separator " ")
          (files '("xml"))
          (file-pattern "^catalog\\.xml$")
          (file-type 'regular)))))
```

Worry not, search path specifications are usually not this tricky.

The `(guix search-paths)` module defines the data type of search path specifications and a number of helper procedures. Below is the reference of search path specifications.

`search-path-specification` [Data Type]

The data type for search path specifications.

`variable` The name of the environment variable for this search path (a string).

`files` The list of sub-directories (strings) that should be added to the search path.

`separator` (default: `":"`)
The string used to separate search path components.

As a special case, a `separator` value of `#f` specifies a "single-component search path"—in other words, a search path that cannot contain more than one element. This is useful in some cases, such as the `SSL_CERT_DIR` variable (honored by OpenSSL, cURL, and a few other packages) or the `ASPELL_DICT_DIR` variable (honored by the GNU Aspell spell checker), both of which must point to a single directory.

`file-type` (default: `'directory`)
The type of file being matched—`'directory` or `'regular`, though it can be any symbol returned by `stat:type` (see Section "File System" in *GNU Guile Reference Manual*).

In the libxml2 example above, we would match regular files; in the Python example, we would match directories.

`file-pattern` (default: `#f`)
This must be either `#f` or a regular expression specifying files to be matched *within* the sub-directories specified by the `files` field.

Again, the libxml2 example shows a situation where this is needed.

Some search paths are not tied by a single package but to many packages. To reduce duplications, some of them are pre-defined in `(guix search-paths)`.

`$SSL_CERT_DIR` [Variable]
`$SSL_CERT_FILE` [Variable]

These two search paths indicate where X.509 certificates can be found (see Section 11.12 [Certificados X.509], page 621).

These pre-defined search paths can be used as in the following example:

```
(package
  (name "curl")
  ;; some fields omitted ...
  (native-search-paths (list $SSL_CERT_DIR $SSL_CERT_FILE)))
```

How do you turn search path specifications on one hand and a bunch of directories on the other hand in a set of environment variable definitions? That's the job of `evaluate-search-paths`.

`evaluate-search-paths` *search-paths directories* [*getenv*] [Procedure]
Evaluate *search-paths*, a list of search-path specifications, for *directories*, a list of directory names, and return a list of specification/value pairs. Use *getenv* to determine the current settings and report only settings not already effective.

The `(guix profiles)` provides a higher-level helper procedure, `load-profile`, that sets the environment variables of a profile.

## 8.9 El almacén

Conceptualmente, el *almacén* es el lugar donde se almacenan las derivaciones cuya construcción fue satisfactoria—por defecto, `/gnu/store`. Los subdirectorios en el almacén se denominan *elementos del almacén* o *rutas del almacén* en ocasiones. El almacén tiene una base de datos asociada que contiene información como las rutas del almacén a las que referencia cada ruta del almacén, y la lista de elementos *válidos* del almacén—los resultados de las construcciones satisfactorias. Esta base de datos reside en *`localstatedir`*`/guix/db`, donde *localstatedir* es el directorio de estado especificado *vía* `--localstatedir` en tiempo de configuración, normalmente `/var`.

El almacén *siempre* es accedido a través del daemon en delegación de sus clientes (see Section 2.3 [Invocación de guix-daemon], page 12). Para manipular el almacén, los clientes se conectan al daemon por un socket de dominio Unix, le envían peticiones y leen el resultado—esto son llamadas a procedimientos remotos, o RPC.

> **Nota:** Las usuarias *nunca* deben modificar archivos directamente bajo el directorio `/gnu/store`. Esto llevaría a inconsistencias y rompería las premisas de inmutabilidad del modelo funcional de Guix (see Chapter 1 [Introducción], page 1).
>
> See Section 5.6 [Invocación de guix gc], page 53, para información sobre cómo comprobar la integridad del almacén e intentar recuperarse de modificaciones accidentales.

El módulo `(guix store)` proporciona procedimientos para conectarse al daemon y realizar RPCs. Estos se describen más adelante. Por defecto, `open-connection`, y por tanto todas las órdenes `guix`, se conectan al daemon local o a la URI especificada en la variable de entorno `GUIX_DAEMON_SOCKET`.

`GUIX_DAEMON_SOCKET` [Variable de entorno]
Cuando se ha definido, el valor de esta variable debe ser un nombre de archivo o una URI designando el punto de conexión del daemon. Cuando es un nombre de archivo,

denota un socket de dominio Unix al que conectarse. Además de nombres de archivos, los esquemas de URI aceptados son:

`file`
`unix` Estos son equivalentes a los sockets de dominio Unix. `file:///var/guix/daemon-socket/socket` es equivalente a `/var/guix/daemon-socket/socket`.

`guix` Estas URI denotan conexiones sobre TCP/IP, sin cifrado ni verificación de la máquina remota. La URI debe especificar el nombre de máquina y opcionalmente un número de puerto (por defecto se usa el puerto 44146):

```
guix://principal.guix.example.org:1234
```

Esta configuración es apropiada para redes locales, como clusters, donde únicamente los nodos de confianza pueden conectarse al daemon de construcción en `principal.guix.example.org`.

La opción `--listen` de `guix-daemon` puede usarse para indicarle que escuche conexiones TCP (see Section 2.3 [Invocación de guix-daemon], page 12).

`ssh` These URIs allow you to connect to a remote daemon over SSH. This feature requires Guile-SSH (see Section 22.1 [Requisitos], page 734) and a working `guile` binary in `PATH` on the destination machine. It supports public key and GSSAPI authentication. A typical URL might look like this:

```
ssh://carlos@guix.example.org:22
```

Como con `guix copy`, se tienen en cuenta los archivos habituales de configuración del cliente OpenSSH (see Section 9.13 [Invocación de guix copy], page 233).

Esquemas URI adicionales pueden ser aceptados en el futuro.

> **Nota:** La conexión con daemon de construcción remotos se considera experimental en c7888f5. Por favor, contacte con nosotras para compartir cualquier problema o sugerencias que pueda tener (see Chapter 22 [Contribuir], page 734).

`open-connection` [*uri*] [*#:reserve-space? #t*] [Procedimiento]
Abre una conexión al daemon a través del socket de dominio Unix apuntado por *uri* (una cadena). Cuando *reserve-space?* es verdadero, le indica que reserve un poco de espacio extra en el sistema de archivos de modo que el recolector de basura pueda operar incluso cuando el disco se llene. Devuelve un objeto servidor.

El valor por defecto de *uri* es `%default-socket-path`, que ese la ruta esperada según las opciones proporcionadas a `configure`.

`close-connection` *servidor* [Procedimiento]
Cierra la conexión al *servidor*.

`current-build-output-port` [Variable]
Esta variable está enlazada a un parámetro SRFI-39, que referencia al puerto donde los logs de construcción y error enviados por el daemon deben escribirse.

Los procedimientos que realizan RPCs toman todos como primer parámetro un objeto servidor.

`valid-path?` *servidor ruta* [Procedimiento]
Devuelve `#t` cuando *ruta* designa un elemento válido del almacén y `#f` en otro caso (un elemento no-válido puede existir en el disco pero aun así no ser válido, por ejemplo debido a que es el resultado de una construcción que se interrumpió o falló).

Una condición `&store-protocol-error` se eleva si *ruta* no contiene como prefijo el directorio del almacén (`/gnu/store`).

`add-text-to-store` *servidor nombre texto* [*referencias*] [Procedimiento]
Añade *texto* bajo el archivo *nombre* en el almacén, y devuelve su ruta en el almacén. *referencias* es la lista de rutas del almacén a las que hace referencia la ruta del almacén resultante.

`build-derivations` *almacén derivaciones* [*modo*] [Procedimiento]
Construye *derivaciones*, una lista de objetos `<derivation>`, nombres de archivo `.drv`, o pares derivación/salida, usando el *modo* especificado—`(build-mode normal)` en caso de omisión.

Fíjese que el módulo `(guix monads)` proporciona una mónada así como versiones monádicas de los procedimientos previos, con el objetivo de hacer más conveniente el trabajo con código que accede al almacén (see Section 8.11 [La mónada del almacén], page 162).

*Esta sección actualmente está incompleta.*

## 8.10 Derivaciones

Las acciones de construcción a bajo nivel y el entorno en el que se realizan se representan mediante *derivaciones*. Una derivación contiene las siguientes piezas de información:

- Las salidas de la derivación—las derivaciones producen al menos un archivo o directorio en el almacén, pero pueden producir más.
- The inputs of the derivation—i.e., its build-time dependencies—which may be other derivations or plain files in the store (patches, build scripts, etc.).
- El tipo de sistema objetivo de la derivación—por ejemplo, `x86_64-linux`.
- El nombre de archivo del guión de construcción en el almacén, junto a los parámetros que se le deben pasar.
- Una lista de variables de entorno a ser definidas.

Las derivaciones permiten a los clientes del daemon comunicar acciones de construcción al almacén. Existen en dos formas: como una representación en memoria, tanto en el lado del cliente como el del daemon, y como archivos en el almacén cuyo nombre termina en `.drv`—estos archivos se conocen como *rutas de derivación*. Las rutas de derivación se pueden proporcionar al procedimiento `build-derivations` para que realice las acciones de construcción prescritas (see Section 8.9 [El almacén], page 157).

Operaciones como la descarga de archivos y las instantáneas de un control de versiones para las cuales el hash del contenido esperado se conoce previamente se modelan como

*derivaciones de salida fija*. Al contrario que las derivaciones normales, las salidas de una derivación de salida fija son independientes de sus entradas—por ejemplo, la descarga del código fuente produce el mismo resultado independientemente del método de descarga y las herramientas usadas.

Las derivaciones de salida—es decir, los resultados de construcción—tienen un conjunto de *referencias*, del que informa la RPC `references` o la orden `guix gc --references` (see Section 5.6 [Invocación de guix gc], page 53). Las referencias son el conjunto de dependencias en tiempo de ejecución de los resultados de construcción. Las referencias son un subconjunto de las entradas de la derivación; el daemon de construcción calcula este subconjunto de forma automática mediante el procesado de todos los archivos en las salidas.

El módulo `(guix derivations)` proporciona una representación de derivaciones como objetos Scheme, junto a procedimientos para crear y manipular de otras formas derivaciones. La primitiva de más bajo nivel para crear una derivación es el procedimiento `derivation`:

`derivation` *almacén nombre constructor args* [Procedimiento]
[#:outputs '("out")] [#:hash #f] [#:hash-algo #f] [#:recursive? #f] [#:inputs '()] [#:env-vars '()] [#:system (%current-system)] [#:references-graphs #f] [#:allowed-references #f] [#:disallowed-references #f] [#:leaked-env-vars #f] [#:local-build? #f] [#:substitutable? #t] [#:properties '()] Construye una derivación con los parámetros proporcionados, y devuelve el objeto `<derivation>` resultante.

Cuando se proporcionan *hash* y *hash-algo*, una *derivación de salida fija* se crea—es decir, una cuyo resultado se conoce de antemano, como la descarga de un archivo. Si, además, *recursive?* es verdadero, entonces la salida fijada puede ser un archivo ejecutable o un directorio y *hash* debe ser el hash de un archivador que contenga esta salida.

Cuando *references-graphs* es verdadero, debe ser una lista de pares de nombre de archivo/ruta del almacén. En ese caso, el grafo de referencias de cada ruta del almacén se exporta en el entorno de construcción del archivo correspondiente, en un formato de texto simple.

Cuando *allowed-references* es verdadero, debe ser una lista de elementos del almacén o salidas a las que puede hacer referencia la salida de la derivación. Del mismo modo, *disallowed-references*, en caso de ser verdadero, debe ser una lista de cosas a las que las salidas *no* pueden hacer referencia.

Cuando *leaked-env-vars* es verdadero, debe ser una lista de cadenas que denoten variables de entorno que se permite "escapar" del entorno del daemon al entorno de construcción. Esto es únicamente aplicable a derivaciones de salida fija—es decir, cuando *hash* es verdadero. El uso principal es permitir que variables como `http_proxy` sean pasadas a las derivaciones que descargan archivos.

Cuando *local-build?* es verdadero, declara que la derivación no es una buena candidata para delegación y debe ser construida localmente (see Section 2.2.2 [Configuración de delegación del daemon], page 7). Este es el caso para pequeñas derivaciones donde los costes de transferencia de datos sobrepasarían los beneficios.

Cuando *substitutable?* es falso, declara que las sustituciones de la salida de la derivación no deben usarse (see Section 5.3 [Sustituciones], page 46). Esto es útil,

por ejemplo, cuando se construyen paquetes que capturan detalles sobre el conjunto de instrucciones de la CPU anfitriona.

*properties* debe ser una lista asociada que describe "propiedades" de la derivación. Debe mantenerse tal cual, sin interpretar, en la derivación.

Esto es un ejemplo con un guión de shell como constructor, asumiendo que *almacén* es una conexión abierta al daemon, *bash* apunta al ejecutable Bash en el almacén:

```
(use-modules (guix utils)
             (guix store)
             (guix derivations))

(let ((constructor   ; añade el guión de Bash al almacén
        (add-text-to-store store "mi-constructor.sh"
                           "echo hola mundo > $out\n" '())))
  (derivation almacen "foo"
              bash `("-e" ,builder)
              #:inputs `((,bash) (,constructor))
              #:env-vars '(("HOME" . "/sindirectorio"))))
⇒ #<derivation /gnu/store/...-foo.drv => /gnu/store/...-foo>
```

Como puede suponerse, el uso directo de esta primitiva es algo enrevesado. Una mejor aproximación es escribir guiones de construcción en Scheme, ¡por supuesto! La mejor forma de hacerlo es escribir el código de construcción como una "expresión-G", y pasarla a `gexp->derivation`. Para más información, see Section 8.12 [Expresiones-G], page 167.

En otros tiempos, `gexp->derivation` no existía y la creación de derivaciones con código de construcción escrito en Scheme se conseguía con `build-expression->derivation`, documentada más adelante. Este procedimiento está ahora obsoleto en favor del procedimiento `gexp->derivation` mucho más conveniente.

`build-expression->derivation` *almacén nombre exp* [Procedimiento]
[*#:system* (*%current-system*)] [*#:inputs* '()] [*#:outputs* '("out")] [*#:hash* *#f*] [*#:hash-algo* *#f*] [*#:recursive?* *#f*] [*#:env-vars* '()] [*#:modules* '()] [*#:references-graphs* *#f*] [*#:allowed-references* *#f*] [*#:disallowed-references* *#f*] [*#:local-build?* *#f*] [*#:substitutable?* *#t*] [*#:guile-for-build* *#f*]

Devuelve una derivación que ejecuta la expresión Scheme *exp* como un constructor para la derivación *nombre*. *inputs* debe ser una lista de tuplas `(nombre ruta-drv sub-drv)`; cuando *sub-drv* se omite, se asume `"out"`. *modules* es una lista de nombres de módulos Guile de la ruta actual de búsqueda a copiar en el almacén, compilados, y poner a disposición en la ruta de carga durante la ejecución de *exp*—por ejemplo, `((guix build utils) (guix build gnu-build-system))`.

*exp* se evalúa en un entorno donde `%outputs` está asociada a una lista de pares salida/ruta, y donde `%build-inputs` está asociada a una lista de pares cadena/ruta-de-salida que provienen de *inputs*. De manera opcional, *env-vars* es una lista de pares de cadenas que especifican el nombre y el valor de las variables de entorno visibles al constructor. El constructor termina pasando el resultado de *exp* a `exit`; por tanto, cuando *exp* devuelve `#f`, la construcción se considera fallida.

*exp* se construye usando *guile-for-build* (una derivación). Cuando *guile-for-build* se omite o es `#f`, el valor del fluido `%guile-for-build` se usa en su lugar.

Véase el procedimiento `derivation` para el significado de *references-graphs*, *allowed-references*, *disallowed-references*, *local-build?* y *substitutable?*.

Aquí está un ejemplo de derivación de salida única que crea un directorio que contiene un archivo:

```
(let ((constructor '(let ((salida (assoc-ref %outputs "out")))
                      (mkdir salida)    ; crea /gnu/store/...-goo
                      (call-with-output-file (string-append salida "/prueba")
                        (lambda (p)
                          (display '(hola guix) p))))))
  (build-expression->derivation almacen "goo" constructor))

⇒ #<derivation /gnu/store/...-goo.drv => ...>
```

## 8.11 La mónada del almacén

Los procedimientos que operan en el almacén descritos en la sección previa toman todos una conexión abierta al daemon de construcción en su primer parámetro. Aunque el modelo subyacente es funcional, tienen o bien efectos secundarios o dependen del estado actual del almacén.

Lo anterior es inconveniente: la conexión al daemon de construcción tiene que proporcionarse en todas estas funciones, haciendo imposible la composición de funciones que no toman ese parámetro con funciones que sí lo hacen. Lo último puede ser problemático: ya que las operaciones del almacén tienen efectos secundarios y/o dependen del estado externo, deben ser secuenciadas de manera adecuada.

Aquí es donde entra en juego el módulo `(guix monads)`. Este módulo proporciona un entorno para trabajar con *mónadas*, y una mónada particularmente útil para nuestros usos, la *mónada del almacén*. Las mónadas son una construcción que permite dos cosas: asociar "contexto" con valores (en nuestro caso, el contexto es el almacén), y la construcción de secuencias de computaciones (aquí computaciones incluye accesos al almacén). Los valores en una mónada—valores que transportan este contexto adicional—se llaman *valores monádicos*; los procedimientos que devuelven dichos valores se llaman *procedimientos monádicos*.

Considere este procedimiento "normal":

```
(define (enlace-sh almacen)
  ;; Devuelve una derivación que enlaza el ejecutable 'bash'.
  (let* ((drv (package-derivation store bash))
         (out (derivation->output-path drv))
         (sh  (string-append out "/bin/bash")))
    (build-expression->derivation store "sh"
                                  `(symlink ,sh %output))))
```

Mediante el uso de `(guix monads)` y `(guix gexp)`, puede reescribirse como una función monádica:

```
(define (enlace-sh)
```

```
;; Lo mismo, pero devuelve un valor monádico.
(mlet %store-monad ((drv (package->derivation bash)))
  (gexp->derivation "sh"
                    #~(symlink (string-append #$drv "/bin/bash")
                               #$output))))
```

Hay varias cosas a tener en cuenta en la segunda versión: el parámetro `store` ahora es implícito y es "hilado en las llamadas a los procedimientos monádicos `package->derivation` y `gexp->derivation`, y el valor monádico devuelto por `package->derivation` es *asociado* mediante el uso de `mlet` en vez de un simple `let`.

Al final, la llamada a `package->derivation` puede omitirse ya que tendrá lugar implícitamente, como veremos más adelante (see Section 8.12 [Expresiones-G], page 167):

```
(define (enlace-sh)
  (gexp->derivation "sh"
                    #~(symlink (string-append #$bash "/bin/bash")
                               #$output)))
```

La ejecución del procedimiento monádico `enlace-para-sh` no tiene ningún efecto. Como alguien dijo una vez, "sales de una mónada como sales de un edificio en llamas: corriendo" (run en inglés). Por tanto, para salir de la mónada y obtener el efecto deseado se debe usar `run-with-store`:

```
(run-with-store (open-connection) (enlace-sh))
⇒ /gnu/store/...-enlace-para-sh
```

Note that the `(guix monad-repl)` module extends the Guile REPL with new "commands" to make it easier to deal with monadic procedures: `run-in-store`, and `enter-store-monad` (see Section 8.14 [Using Guix Interactively], page 178). The former is used to "run" a single monadic value through the store:

```
scheme@(guile-user)> ,run-in-store (package->derivation hello)
$1 = #<derivation /gnu/store/...-hello-2.9.drv => ...>
```

El último entra en un entorno interactivo recursivo, donde todos los valores devueltos se ejecutan automáticamente a través del almacén:

```
scheme@(guile-user)> ,enter-store-monad
store-monad@(guile-user) [1]> (package->derivation hello)
$2 = #<derivation /gnu/store/...-hello-2.9.drv => ...>
store-monad@(guile-user) [1]> (text-file "foo" "Hello!")
$3 = "/gnu/store/...-foo"
store-monad@(guile-user) [1]> ,q
scheme@(guile-user)>
```

Fíjese que los valores no-monádicos no pueden devolverse en el entorno interactivo `store-monad`.

Other meta-commands are available at the REPL, such as `,build` to build a file-like object (see Section 8.14 [Using Guix Interactively], page 178).

Las formas sintácticas principales para tratar con mónadas en general se proporcionan por el módulo `(guix monads)` y se describen a continuación.

`with-monad` *monad body ...* [Macro]

Evalúa cualquier forma `>>=` o `return` en *cuerpo* como estando en *mónada*.

`return` *val* [Macro]

Devuelve el valor monádico que encapsula *val*.

`>>=` *mval mproc . . .* [Macro]

*Asocia* el valor monádico *mval*, pasando su "contenido" a los procedimientos monádicos *mproc. . .*[4]. Puede haber un *mproc* o varios, como en este ejemplo:

```
(run-with-state
    (with-monad %state-monad
      (>>= (return 1)
           (lambda (x) (return (+ 1 x)))
           (lambda (x) (return (* 2 x)))))
  'un-estado)

⇒ 4
⇒ un-estado
```

`mlet` *mónada* ((*var mval*) . . .) *cuerpo . . .* [Macro]
`mlet*` *mónada* ((*var mval*) . . .) *cuerpo . . .* [Macro]

Asocia las variables *var* a los valores monádicos *mval* en *cuerpo*, el cual es una secuencia de expresiones. Como con el operador bind, esto puede pensarse como el "desempaquetado" del valor crudo no-monádico dentro del ámbito del *cuerpo*. La forma (*var* `->` *val*) asocia *var* al valor "normal" *val*, como en `let`. Las operaciones de asociación ocurren en secuencia de izquierda a derecha. La última expresión de *cuerpo* debe ser una expresión monádica, y su resultado se convertirá en el resultado de `mlet` o `mlet*` cuando se ejecute en la *mónada*.

`mlet*` es a `mlet` lo que `let*` es a `let` (see Section "Local Bindings" in *GNU Guile Reference Manual*).

`mbegin` *monad mexp . . .* [Macro]

Asocia *mexp* y las siguientes expresiones monádicas en secuencia, devolviendo el resultado de la última expresión. Cada expresión en la secuencia debe ser una expresión monádica.

Esto es similar a `mlet`, excepto que los valores devueltos por las expresiones monádicas se ignoran. En ese sentido el funcionamiento es análogo a `begin` pero aplicado a expresiones monádicas.

`mwhen` *condition mexp0 mexp* . . .* [Macro]

Cuando *condición* es verdadero, evalúa la secuencia de expresiones monádicas *mexp0..mexp** como dentro de `mbegin`. Cuando *condición* es falso, devuelve `*unespecified*` en la mónada actual. Todas las expresiones en la secuencia deben ser expresiones monádicas.

`munless` *condition mexp0 mexp* . . .* [Macro]

Cuando *condición* es falso, evalúa la secuencia de expresiones monádicas *mexp0..mexp** como dentro de `mbegin`. Cuando *condición* es verdadero, devuelve

[4] Esta operación es habitualmente conocida como "bind" (asociación), pero ese nombre denota un procedimiento no relacionado en Guile. Por tanto usamos este símbolo en cierto modo críptico heredado del lenguaje Haskell.

`*unespecified*` en la mónada actual. Todas las expresiones en la secuencia deben ser expresiones monádicas.

El módulo `(guix monads)` proporciona la *mónada de estado*, que permite que un valor adicional—el estado—sea *hilado* a través de las llamadas a procedimientos monádicos.

`%state-monad` [Variable]

La mónada de estado. Procedimientos en la mónada de estado pueden acceder y cambiar el estado hilado.

Considere el siguiente ejemplo. El procedimiento `cuadrado` devuelve un valor en la mónada de estado.

```
(define (cuadrado x)
  (mlet %state-monad ((count (current-state)))
    (mbegin %state-monad
      (set-current-state (+ 1 count))
      (return (* x x)))))

(run-with-state (sequence %state-monad (map cuadrado (iota 3))) 0)
⇒ (0 1 4)
⇒ 3
```

Cuando se "ejecuta" a través de `%state-monad`, obtenemos un valor adicional de estado, que es el número de llamadas a `cuadrado`.

`current-state` [Procedimiento monádico]

Devuelve el estado actual como un valor monádico.

`set-current-state` *valor* [Procedimiento monádico]

Establece el estado actual a *valor* y devuelve el estado previo como un valor monádico.

`state-push` *valor* [Procedimiento monádico]

Apila *valor* al estado actual, que se asume que es una lista, y devuelve el estado previo como un valor monádico.

`state-pop` [Procedimiento monádico]

Extrae un valor del estado actual y lo devuelve como un valor monádico. Se asume que el estado es una lista.

`run-with-state` *mval* [*estado*] [Procedimiento]

Ejecuta un valor monádico *mval* comenzando con *estado* como el estado inicial. Devuelve dos valores: el valor resultante y el estado resultante.

La interfaz principal a la mónada del almacén, proporcionada por el módulo `(guix store)`, es como sigue.

`%store-monad` [Variable]

La mónada del almacén—un alias para `%state-monad`.

Los valores en la mónada del almacén encapsulan los accesos al almacén. Cuando su efecto es necesario, un valor de la mónada del almacén será "evaluado" cuando se proporcione al procedimiento `run-with-store` (véase a continuación).

`run-with-store` *almacén mval* [*#:guile-for-build*] [*#:system* (*%current-system*)] *Ejecuta* `mval`, *un* [Procedimiento]
valor monádico en la mónada del almacén, en *almacén*, una conexión abierta al almacén.

`text-file` *nombre texto* [*referencias*] [Procedimiento monádico]
Devuelve como un valor monádico el nombre absoluto del archivo en el almacén del archivo que contiene *exto*, una cadena. *referencias* es una lista de elementos del almacén a los que el archivo de texto referencia; su valor predeterminado es la lista vacía.

`binary-file` *nombre datos* [*referencias*] [Procedimiento monádico]
Devuelve como un valor monádico el nombre absoluto del archivo en el almacén del archivo que contiene *datos*, un vector de bytes. *referencias* es una lista de elementos del almacén a los que el archivo binario referencia; su valor predeterminado es la lista vacía.

`interned-file` *archivo* [*nombre*] [*#:recursive? #t*] [*#:select?* (*const #t*)] [Procedimiento monádico]
Devuelve el nombre del *archivo* una vez internado en el almacén. Usa *nombre* como su nombre del almacén, o el nombre base de *archivo* si *nombre* se omite.

Cuando *recursive?* es verdadero, los contenidos del *archivo* se añaden recursivamente; si *archivo* designa un archivo plano y *recursive?* es verdadero, sus contenidos se añaden, y sus bits de permisos se mantienen.

Cuando *recursive?* es verdadero, llama a `(`*select? archivo stat*`)` por cada entrada del directorio, donde *archivo* es el nombre absoluto de archivo de la entrada y *stat* es el resultado de `lstat`; excluyendo las entradas para las cuales *select?* no devuelve verdadero.

El ejemplo siguiente añade un archivo al almacén, bajo dos nombres diferentes:

```
(run-with-store (open-connection)
  (mlet %store-monad ((a (interned-file "README"))
                      (b (interned-file "README" "LEGU-MIN")))
    (return (list a b))))

⇒ ("/gnu/store/rwm...-README" "/gnu/store/44i...-LEGU-MIN")
```

El módulo `(guix packages)` exporta los siguientes procedimientos monádicos relacionados con paquetes:

`package-file` *paquete* [*archivo*] [*#:system* (*%current-system*)] [*#:target #f*] [*#:output "out"*] [Procedimiento monádico]
Devuelve como un valor monádico el nombre absoluto de archivo de *archivo* dentro del directorio de salida *output* del *paquete*. Cuando se omite *archivo*, devuelve el nombre del directorio de salida *output* del *paquete*. Cuando *target* es verdadero, se usa como una tripleta de compilación cruzada.

Tenga en cuenta que este procedimiento *no* construye *paquete*. Por lo tanto, el resultado puede designar o no un archivo existente. Le recomendamos que no use este procedimiento a no ser que sepa qué está haciendo.

`package->derivation` *paquete* [*sistema*] [Procedimiento monádico]
`package->cross-derivation` *paquete objetivo* [*sistema*] [Procedimiento monádico]

Versión monádica de `package-derivation` y `package-cross-derivation` (see Section 8.2 [Definición de paquetes], page 102).

## 8.12 Expresiones-G

Por tanto tenemos "derivaciones", que representan una secuencia de acciones de construcción a realizar para producir un elemento en el almacén (see Section 8.10 [Derivaciones], page 159). Estas acciones de construcción se llevan a cabo cuando se solicita al daemon construir realmente la derivación; se ejecutan por el daemon en un contenedor (see Section 2.3 [Invocación de guix-daemon], page 12).

No debería ser ninguna sorpresa que nos guste escribir estas acciones de construcción en Scheme. Cuando lo hacemos, terminamos con dos *estratos* de código Scheme[5]: el "código anfitrión"—código que define paquetes, habla al daemon, etc.—y el "código de construcción"—código que realmente realiza las acciones de construcción, como la creación de directorios, la invocación de `make`, etcétera (see Section 8.6 [Fases de construcción], page 143).

Para describir una derivación y sus acciones de construcción, típicamente se necesita embeber código de construcción dentro del código anfitrión. Se resume en la manipulación de código de construcción como datos, y la homoiconicidad de Scheme—el código tiene representación directa como datos—es útil para ello. Pero necesitamos más que el mecanismo normal de `quasiquote` en Scheme para construir expresiones de construcción.

El módulo `(guix gexp)` implementa las *expresiones-G*, una forma de expresiones-S adaptada para expresiones de construcción. Las expresiones-G, o *gexps*, consiste esencialmente en tres formas sintácticas: `gexp`, `ungexp` y `ungexp-splicing` (o simplemente: `#~`, `#$` y `#$@`), que son comparables a `quasiquote`, `unquote` y `unquote-splicing`, respectivamente (see Section "Expression Syntax" in *GNU Guile Reference Manual*). No obstante, hay importantes diferencias:

- Las expresiones-G están destinadas a escribirse en un archivo y ser ejecutadas o manipuladas por otros procesos.
- Cuando un objeto de alto nivel como un paquete o una derivación se expande dentro de una expresión-G, el resultado es el mismo que la introducción de su nombre de archivo de salida.
- Las expresiones-G transportan información acerca de los paquetes o derivaciones que referencian, y estas referencias se añaden automáticamente como entradas al proceso de construcción que las usa.

Este mecanismo no se limita a objetos de paquete ni derivación: pueden definirse *compiladores* capaces de "bajar el nivel" de otros objetos de alto nivel a derivaciones o archivos en el almacén, de modo que esos objetos puedan introducirse también en expresiones-G. Por ejemplo, un tipo útil de objetos de alto nivel que pueden insertarse en una expresión-G

[5] El término *estrato* en este contexto se debe a Manuel Serrano et al. en el contexto de su trabajo en Hop. Oleg Kiselyov, quien ha escrito profundos ensayos sobre el tema (`http://okmij.org/ftp/meta-programming/#meta-scheme`), se refiere a este tipo de generación de código como separación en etapas o *staging*.

son los "objetos tipo-archivo", los cuales facilitan la adición de archivos al almacén y su referencia en derivaciones y demás (vea `local-file` y `plain-file` más adelante).

Para ilustrar la idea, aquí está un ejemplo de expresión-G:

```
(define exp-construccion
  #~(begin
      (mkdir #$output)
      (chdir #$output)
      (symlink (string-append #$coreutils "/bin/ls")
               "enumera-archivos")))
```

Esta expresión-G puede pasarse a `gexp->derivation`; obtenemos una derivación que construye un directorio que contiene exactamente un enlace simbólico a `/gnu/store/...-coreutils-8.22/bin/ls`:

```
(gexp->derivation "la-cosa" exp-construccion)
```

Como se puede esperar, la cadena `"/gnu/store/...-coreutils-8.22"` se sustituye por la referencia al paquete *coreutils* en el código de construcción real, y *coreutils* se marca automáticamente como una entrada a la derivación. Del mismo modo, `#$output` (equivalente a `(ungexp output)`) se reemplaza por una cadena que contiene el nombre del directorio de la salida de la derivación.

En un contexto de compilación cruzada, es útil distinguir entre referencias a construcciones *nativas* del paquete—que pueden ejecutarse en el sistema anfitrión—de referencias de compilaciones cruzadas de un paquete. Para dicho fin, `#+` tiene el mismo papel que `#$`, pero es una referencia a una construcción nativa del paquete:

```
(gexp->derivation "vi"
   #~(begin
       (mkdir #$output)
       (mkdir (string-append #$output "/bin"))
       (system* (string-append #+coreutils "/bin/ln")
                "-s"
                (string-append #$emacs "/bin/emacs")
                (string-append #$output "/bin/vi")))
   #:target "aarch64-linux-gnu")
```

En el ejemplo previo, se usa la construcción nativa de *coreutils*, de modo que `ln` pueda realmente ejecutarse en el anfitrión; pero se hace referencia a la construcción de compilación cruzada de *emacs*.

Otra característica de las expresiones-G son los *módulos importados*: a veces deseará ser capaz de usar determinados módulos Guile del "entorno anfitrión" en la expresión-G, de modo que esos módulos deban ser importados en el "entorno de construcción". La forma `with-imported-modules` le permite expresarlo:

```
(let ((build (with-imported-modules '((guix build utils))
               #~(begin
                   (use-modules (guix build utils))
                   (mkdir-p (string-append #$output "/bin"))))))
  (gexp->derivation "directorio-vacio"
                    #~(begin
                        #$build
```

```
                          (display "éxito!\n")
                          #t)))
```

En este ejemplo, el módulo `(guix build utils)` se incorpora automáticamente dentro del entorno de construcción aislado de nuestra expresión-G, de modo que `(use-modules (guix build utils))` funciona como se espera.

De manera habitual deseará que la *clausura* del módulo se importe—es decir, el módulo en sí y todos los módulos de los que depende—en vez del módulo únicamente; si no se hace, cualquier intento de uso del módulo fallará porque faltan módulos dependientes. El procedimiento `source-module-closure` computa la clausura de un módulo mirando en las cabeceras de sus archivos de fuentes, lo que es útil en este caso:

```
(use-modules (guix modules))   ;para 'source-module-closure'

(with-imported-modules (source-module-closure
                        '((guix build utils)
                          (gnu build image)))
  (gexp->derivation "something-with-vms"
                    #~(begin
                        (use-modules (guix build utils)
                                     (gnu build image))
                        ...)))
```

De la misma manera, a veces deseará importar no únicamente módulos puros de Scheme, pero también "extensiones" como enlaces Guile a bibliotecas C u otros paquetes "completos". Si, digamos, necesitase el paquete `guile-json` disponible en el lado de construcción, esta sería la forma de hacerlo:

```
(use-modules (gnu packages guile))  ;para 'guile-json'

(with-extensions (list guile-json)
  (gexp->derivation "algo-con-json"
                    #~(begin
                        (use-modules (json))
                        ...)))
```

La forma sintáctica para construir expresiones-G se resume a continuación.

`#~`*exp* [Macro]
`(gexp` *exp*`)` [Macro]

Devuelve una expresión-G que contiene *exp*. *exp* puede contener una o más de las siguientes formas:

`#$`*obj*
`(ungexp` *obj*`)`

Introduce una referencia a *obj*. *obj* puede tener uno de los tipos permitidos, por ejemplo un paquete o derivación, en cuyo caso la forma `ungexp` se substituye por el nombre de archivo de su salida—por ejemplo, `"/gnu/store/...-coreutils-8.22`.

Si *obj* es una lista, se recorre y las referencias a objetos permitidos se substituyen de manera similar.

Si *obj* es otra expresión-G, su contenido se inserta y sus dependencias se añaden a aquellas de la expresión-G que la contiene.

Si *obj* es otro tipo de objeto, se inserta tal cual es.

`#$obj:salida`
`(ungexp obj salida)`
Como la forma previa, pero referenciando explícitamente la *salida* de *obj*—esto es útil cuando *obj* produce múltiples salidas (see Section 5.4 [Paquetes con múltiples salidas], page 51).

Sometimes a gexp unconditionally refers to the `"out"` output, but the user of that gexp would still like to insert a reference to another output. The `gexp-input` procedure aims to address that. See [gexp-input], page 176.

`#+obj`
`#+obj:salida`
`(ungexp-native obj)`
`(ungexp-native obj salida)`
Igual que `ungexp`, pero produce una referencia a la construcción *nativa* de *obj* cuando se usa en un contexto de compilación cruzada.

`#$output[:salida]`
`(ungexp output [salida])`
Inserta una referencia a la salida de la derivación *salida*, o a la salida principal cuando *salida* se omite.

Esto únicamente tiene sentido para expresiones-G pasadas a `gexp->derivation`.

`#$@lst`
`(ungexp-splicing lst)`
Lo mismo que la forma previa, pero expande el contenido de la lista *lst* como parte de la lista que la contiene.

`#+@lst`
`(ungexp-native-splicing lst)`
Lo mismo que la forma previa, pero hace referencia a las construcciones nativas de los objetos listados en *lst*.

Las expresiones-G creadas por `gexp` o `#~` son objetos del tipo `gexp?` en tiempo de ejecución (véase a continuación).

`with-imported-modules` *módulos cuerpo...* [Macro]
Marca las expresiones-G definidas en el *cuerpo...* como si requiriesen *módulos* en su entorno de ejecución.

Cada elemento en *módulos* puede ser el nombre de un módulo, como `(guix build utils)`, o puede ser el nombre de un módulo, seguido de una flecha, seguido de un objeto tipo-archivo:

```
`((guix build utils)
  (guix gcrypt)
  ((guix config) => ,(scheme-file "config.scm"
```

```
#~(define-module ...))))
```

En el ejemplo previo, los dos primeros módulos se toman de la ruta de búsqueda, y el último se crea desde el objeto tipo-archivo proporcionado.

Esta forma tiene ámbito *léxico*: tiene efecto en las expresiones-G definidas en *cuerpo*..., pero no en aquellas definidas, digamos, en procedimientos llamados por *cuerpo*....

`with-extensions` *extensiones cuerpo*... [Macro]

Marca que las expresiones definidas en *cuerpo*... requieren *extensiones* en su entorno de construcción y ejecución. *extensiones* es típicamente una lista de objetos de paquetes como los que se definen en el módulo `(gnu packages guile)`.

De manera concreta, los paquetes listados en *extensiones* se añaden a la ruta de carga mientras se compilan los módulos importados en *cuerpo*...; también se añaden a la ruta de carga en la expresión-G devuelta por *cuerpo*....

`gexp?` *obj* [Procedimiento]

Devuelve `#t` si *obj* es una expresión-G.

Las expresiones-G están destinadas a escribirse en disco, tanto en código que construye alguna derivación, como en archivos planos en el almacén. Los procedimientos monádicos siguientes le permiten hacerlo (see Section 8.11 [La mónada del almacén], page 162, para más información sobre mónadas).

`gexp->derivation` *nombre exp* [*#:system* (*%current-system*)] [*#:target #f*] [*#:graft? #t*] [*#:hash #f*] [*#:hash-algo #f*] [*#:recursive? #f*] [*#:env-vars '()*] [*#:modules '()*] [*#:module-path* `%load-path`] [*#:effective-version* `"2.2"`] [*#:references-graphs #f*] [*#:allowed-references #f*] [*#:disallowed-references #f*] [*#:leaked-env-vars #f*] [*#:script-name* (*string-append* *nombre* `"-builder"`)] [*#:deprecation-warnings #f*] [*#:local-build? #f*] [*#:substitutable? #t*] [*#:properties '()*] [*#:guile-for-build #f*] [Procedimiento monádico]

Devuelve una derivación *nombre* que ejecuta *exp* (una expresión-G) con *guile-for-build* (una derivación) en el sistema *system*; *exp* se almacena en un archivo llamado *script-name*. Cuando *target* tiene valor verdadero, se usa como tripleta de compilación cruzada para paquetes a los que haga referencia *exp*.

*modules* está obsoleto en favor de `with-imported-modules`. Su significado es hacer que los módulos *modules* estén disponibles en el contexto de evaluación de *exp*; *modules* es una lista de nombres de módulos Guile buscados en *module-path* para ser copiados al almacén, compilados y disponibles en la ruta de carga durante la ejecución de *exp*—por ejemplo, `((guix build utils) (gui build gnu-build-system))`.

*effective-version* determina la cadena usada cuando se añaden las extensiones de *exp* (vea `with-extensions`) a la ruta de búsqueda—por ejemplo, `"2.2"`.

*graft?* determina si los paquetes a los que *exp* hace referencia deben ser injertados cuando sea posible.

Cuando *references-graphs* es verdadero, debe ser una lista de tuplas de una de las formas siguientes:

```
(file-name obj)
```

```
(file-name obj output)
(file-name gexp-input)
(file-name store-item)
```

El lado derecho de cada elemento de *references-graphs* se convierte automáticamente en una entrada del proceso de construcción de *exp*. En el entorno de construcción, cada *nombre-archivo* contiene el grafo de referencias del elemento correspondiente, en un formato de texto simple.

*allowed-references* debe ser o bien `#f` o una lista de nombres y paquetes de salida. En el último caso, la lista denota elementos del almacén a los que el resultado puede hacer referencia. Cualquier referencia a otro elemento del almacén produce un error de construcción. De igual manera con *disallowed-references*, que enumera elementos a los que las salidas no deben hacer referencia.

*deprecation-warnings* determina si mostrar avisos de obsolescencia durante la compilación de los módulos. Puede ser `#f`, `#t` o `'detailed`.

El resto de parámetros funcionan como en `derivation` (see Section 8.10 [Derivaciones], page 159).

Los procedimientos `local-file`, `plain-file`, `computed-file`, `program-file` y `scheme-file` a continuación devuelven *objetos tipo-archivo*. Esto es, cuando se expanden en una expresión-G, estos objetos dirigen a un archivo en el almacén. Considere esta expresión-G:

```
#~(system* #$(file-append glibc "/sbin/nscd") "-f"
           #$(local-file "/tmp/mi-nscd.conf"))
```

El efecto aquí es el "internamiento" de `/tmp/mi-nscd.conf` mediante su copia al almacén. Una vez expandida, por ejemplo *vía* `gexp->derivation`, la expresión-G hace referencia a la copia bajo `/gnu/store`; por tanto, la modificación o el borrado del archivo en `/tmp` no tiene ningún efecto en lo que la expresión-G hace. `plain-file` puede usarse de manera similar; se diferencia en que el contenido del archivo se proporciona directamente como una cadena.

`local-file` *archivo* [*nombre*] [*#:recursive? #f*] [*#:select? (const #t)*] [Procedimiento]

Devuelve un objeto que representa el archivo local *archivo* a añadir al almacén; este objeto puede usarse en una expresión-G. Si *archivo* es un nombre de archivo relativo, se busca de forma relativa al archivo fuente donde esta forma aparece; si *archivo* no es una cadena literal, se buscará de manera relativa al directorio de trabajo durante la ejecución. *archivo* se añadirá al almacén bajo *nombre*—de manera predeterminada el nombre de *archivo* sin los directorios.

Cuando *recursive?* es verdadero, los contenidos del *archivo* se añaden recursivamente; si *archivo* designa un archivo plano y *recursive?* es verdadero, sus contenidos se añaden, y sus bits de permisos se mantienen.

Cuando *recursive?* es verdadero, llama a `(select? archivo stat)` por cada entrada del directorio, donde *archivo* es el nombre absoluto de archivo de la entrada y *stat* es el resultado de `lstat`; excluyendo las entradas para las cuales *select?* no devuelve verdadero.

*file* can be wrapped in the `assume-valid-file-name` syntactic keyword. When this is done, there will not be a warning when `local-file` is used with a non-literal path. The path is still looked up relative to the current working directory at run time. Wrapping is done like this:

```
(define alice-key-file-path "alice.pub")
;; ...
(local-file (assume-valid-file-name alice-key-file-path))
```

Esta es la contraparte declarativa del procedimiento monádico `interned-file` (see Section 8.11 [La mónada del almacén], page 162).

`plain-file` *nombre contenido* [Procedimiento]

Devuelve un objeto que representa un archivo de texto llamado *nombre* con el *contenido* proporcionado (una cadena o un vector de bytes) para ser añadido al almacén.

Esta es la contraparte declarativa de `text-file`.

`computed-file` *nombre gexp* [*#:local-build? #t*] [*#:options '()*] [Procedimiento]

Devuelve un objeto que representa el elemento del almacén *nombre*, un archivo o un directorio computado por *gexp*. Cuando *local-build?* tiene valor verdadero (el caso predeterminado), la derivación se construye de manera local. *options* es una lista de parámetros adicionales proporcionados a `gexp->derivation`.

Esta es la contraparte declarativa de `gexp->derivation`.

`gexp->script` *nombre exp* [*#:guile (default-guile)*] [Procedimiento monádico] [*#:module-path %load-path*] [*#:system* (%current-system)] [#:target #f] Devuelve un guión ejecutable *nombre* que ejecuta *exp* usando *guile*, con los módulos importados por *exp* en su ruta de búsqueda. Busca los módulos de *exp* en *module-path*.

El ejemplo siguiente construye un guión que simplemente invoca la orden `ls`:

```
(use-modules (guix gexp) (gnu packages base))

(gexp->script "enumera-archivos"
              #~(execl #$(file-append coreutils "/bin/ls")
                       "ls"))
```

Cuando se ejecuta a través del almacén (see Section 8.11 [La mónada del almacén], page 162), obtenemos una derivación que produce un archivo ejecutable `/gnu/store/...-enumera-archivos` más o menos así:

```
#!/gnu/store/...-guile-2.0.11/bin/guile -ds
!#
(execl "/gnu/store/...-coreutils-8.22"/bin/ls" "ls")
```

`program-file` *nombre exp* [*#:guile #f*] [*#:module-path %load-path*] [Procedimiento]

Devuelve un objeto que representa el elemento ejecutable del almacén *nombre* que ejecuta *gexp*. *guile* es el paquete Guile usado para ejecutar el guión. Los módulos importados por *gexp* se buscan en *module-path*.

Esta es la contraparte declarativa de `gexp->script`.

`gexp->file` *nombre exp* [*#:set-load-path? #t*] [Procedimiento monádico]
[*#:module-path %load-path*] [*#:splice? #f*] [*#:guile* (*default-guile*)]

Devuelve una derivación que construye un archivo *nombre* que contiene *exp*. Cuando *splice?* es verdadero, se considera que *exp* es una lista de expresiones que deben ser expandidas en el archivo resultante.

Cuando *set-load-path* es verdadero, emite código en el archivo resultante para establecer `%load-path` y `%load-compiled-path` de manera que respeten los módulos importados por *exp*. Busca los módulos de *exp* en *module-path*.

El archivo resultante hace referencia a todas las dependencias de *exp* o a un subconjunto de ellas.

`scheme-file` *nombre exp* [*#:splice? #f*] [*#:guile #f*] [Procedimiento]
[*#:set-load-path? #t*] *Return an object representing the Scheme*

file *name* that contains *exp*. *guile* is the Guile package used to produce that file.

Esta es la contraparte declarativa de `gexp->file`.

`text-file*` *nombre texto ...* [Procedimiento monádico]

Devuelve como un valor monádico una derivación que construye un archivo de texto que contiene todo *texto*. *texto* puede ser una lista de, además de cadenas, objetos de cualquier tipo que pueda ser usado en expresiones-G: paquetes, derivaciones, archivos locales, objetos, etc. El archivo del almacén resultante hace referencia a todos ellos.

Esta variante debe preferirse sobre `text-file` cuando el archivo a crear haga referencia a elementos del almacén. Esto es el caso típico cuando se construye un archivo de configuración que embebe nombres de archivos del almacén, como este:

```
(define (perfil.sh)
  ;; Devuelve el nombre de un guión shell en el almacén
  ;; que establece la variable de entorno 'PATH'
  (text-file* "perfil.sh"
              "export PATH=" coreutils "/bin:"
              grep "/bin:" sed "/bin\n"))
```

En este ejemplo, el archivo `/gnu/store/...-perfil.sh` resultante hará referencia a *coreutils*, *grep* y *sed*, por tanto evitando que se recolecten como basura durante su tiempo de vida.

`mixed-text-file` *nombre texto ...* [Procedimiento]

Devuelve un objeto que representa el archivo del almacén *nombre* que contiene *texto*. *texto* es una secuencia de cadenas y objetos tipo-archivo, como en:

```
(mixed-text-file "perfil"
                 "export PATH=" coreutils "/bin:" grep "/bin")
```

Esta es la contraparte declarativa de `text-file*`.

`file-union` *nombre archivos* [Procedimiento]

Devuelve un `<computed-file>` que construye un directorio que contiene todos los *archivos*. Cada elemento en *archivos* debe ser una lista de dos elementos donde el primer elemento es el nombre de archivo usado en el nuevo directorio y el segundo elemento es una expresión-G que denota el archivo de destino. Aquí está un ejemplo:

```
(file-union "etc"
```

```
                `(("hosts" ,(plain-file "hosts"
                                        "127.0.0.1 localhost"))
                  ("bashrc" ,(plain-file "bashrc"
                                         "alias ls='ls --color=auto'"))))
```

Esto emite un directorio `etc` que contiene estos dos archivos.

`directory-union` *nombre cosas* [Procedimiento]

Devuelve un directorio que es la unión de *cosas*, donde *cosas* es una lista de objetos tipo-archivo que denotan directorios. Por ejemplo:

```
(directory-union "guile+emacs" (list guile emacs))
```

emite un directorio que es la unión de los paquetes `guile` y `emacs`.

`file-append` *obj sufijo ...* [Procedimientos]

Devuelve un objeto tipo-archivo que se expande a la concatenación de *obj* y *sufijo*, donde *obj* es un objeto que se puede bajar de nivel y cada *sufijo* es una cadena.

Como un ejemplo, considere esta expresión-G:

```
(gexp->script "ejecuta-uname"
              #~(system* #$(file-append coreutils
                                        "/bin/uname")))
```

El mismo efecto podría conseguirse con:

```
(gexp->script "ejecuta-uname"
              #~(system* (string-append #$coreutils
                                        "/bin/uname")))
```

Hay una diferencia no obstante: en el caso de `file-append`, el guión resultante contiene una ruta absoluta de archivo como una cadena, mientras que en el segundo caso, el guión resultante contiene una expresión `(string-append ...)` para construir el nombre de archivo *en tiempo de ejecución*.

`let-system` *sistema cuerpo...* [Macro]
`let-system` (*sistema objetivo*) *cuerpo...* [Macro]

Asocia *sistema* al sistema objetivo actual—por ejemplo, `"x86_64-linux"`—dentro de *cuerpo*.

En el segundo caso, asocia también *objetivo* al objetivo actual de compilación cruzada—una tripleta de GNU como `"arm-linux-gnueabihf"`—o `#f` si no se trata de una compilación cruzada.

`let-system` es útil en el caso ocasional en el que el objeto introducido en la expresión-G depende del sistema objetivo, como en este ejemplo:

```
#~(system*
   #+(let-system system
       (cond ((string-prefix? "armhf-" system)
              (file-append qemu "/bin/qemu-system-arm"))
             ((string-prefix? "x86_64-" system)
              (file-append qemu "/bin/qemu-system-x86_64"))
             (else
              (error "¡ni idea!"))))
   "-net" "user" #$image)
```

`with-parameters` ((*parámetro valor*) ...) *exp* [Macro]
Este macro es similar a la forma `parameterize` para *parámetros* asociados de forma dinámica (see Section "Parameters" in *GNU Guile Reference Manual*). La principal diferencia es que se hace efectivo cuando el objeto tipo-archivo devuelto por *exp* se baja de nivel a una derivación o un elemento del almacén.

Un uso típico de `with-parameters` es para forzar el sistema efectivo de cierto objeto:

```
(with-parameters ((%current-system "i686-linux"))
  coreutils)
```

El ejemplo previo devuelve un objeto que corresponde a la construcción en i686 de Coreutils, independientemente del valor actual de `%current-system`.

`gexp-input` *obj* [*output*] [*#:native? #f*] [Procedure]
Return a *gexp input* record for the given *output* of file-like object *obj*, with `#:native?` determining whether this is a native reference (as with `ungexp-native`) or not.

This procedure is helpful when you want to pass a reference to a specific output of an object to some procedure that may not know about that output. For example, assume you have this procedure, which takes one file-like object:

```
(define (make-symlink target)
  (computed-file "the-symlink"
                 #~(symlink #$target #$output)))
```

Here `make-symlink` can only ever refer to the default output of *target*—the `"out"` output (see Section 5.4 [Paquetes con múltiples salidas], page 51). To have it refer to, say, the `"lib"` output of the `hwloc` package, you can call it like so:

```
(make-symlink (gexp-input hwloc "lib"))
```

You can also compose it like any other file-like object:

```
(make-symlink
  (file-append (gexp-input hwloc "lib") "/lib/libhwloc.so"))
```

Por supuesto, además de expresiones-G embebidas en código "anfitrión", hay también módulos que contienen herramientas de construcción. Para clarificar que están destinados para su uso en el estrato de construcción, estos módulos se mantienen en el espacio de nombres `(guix build ...)`.

Internamente, los objetos de alto nivel se *bajan de nivel*, usando su compilador, a derivaciones o elementos del almacén. Por ejemplo, bajar de nivel un paquete emite una derivación, y bajar de nivel un *plain-file* emite un elemento del almacén. Esto se consigue usando el procedimiento monádico `lower-object`.

`lower-object` *obj* [*sistema*] [*#:target #f*] [Procedimiento monádico]
Devuelve como un valor en `%store-monad` la derivación o elemento del almacén que corresponde a *obj* en *sistema*, compilando de manera cruzada para *target* si *target* es verdadero. *obj* debe ser un objeto que tiene asociado un compilador de expresiones-G, como por ejemplo un objeto del tipo `<package>`.

`gexp->approximate-sexp` *gexp* [Procedure]
Sometimes, it may be useful to convert a G-exp into a S-exp. For example, some linters (see Section 9.8 [Invocación de guix lint], page 216) peek into the build phases

of a package to detect potential problems. This conversion can be achieved with this procedure. However, some information can be lost in the process. More specifically, lowerable objects will be silently replaced with some arbitrary object – currently the list `(*approximate*)`, but this may change.

## 8.13 Invocación de `guix repl`

La orden `guix repl` lanza una *sesión interactiva* Guile (REPL) para la programación interactiva (see Section "Using Guile Interactively" in *GNU Guile Reference Manual*), o para la ejecución de guiones de Guile. Comparado a simplemente lanzar la orden `guile`, `guix repl` garantiza que todos los módulos Guix y todas sus dependencias están disponibles en la ruta de búsqueda.

La sintaxis general es:

```
guix repl opciones [archivo parámetros]
```

Cuando se proporciona *archivo*, *archivo* se ejecuta como un guión de Guile:

```
guix repl mi-guion.scm
```

Para proporcionar parámetros al guión, use `--` para evitar que se interpreten como parámetros específicos de `guix repl`:

```
guix repl -- mi-guion.scm --input=foo.txt
```

Pare hacer que un guión sea ejecutable directamente desde el shell, mediante el uso del ejecutable de guix que se encuentre en la ruta de búsqueda de la usuaria, escriba las siguientes dos líneas al inicio del archivo:

```
#!/usr/bin/env -S guix repl --
!#
```

To make a script that launches an interactive REPL directly from the shell, use the `--interactive` flag:

```
#!/usr/bin/env -S guix repl --interactive
!#
```

Without a file name argument, a Guile REPL is started, allowing for interactive use (see Section 8.14 [Using Guix Interactively], page 178):

```
$ guix repl
scheme@(guile-user)> ,use (gnu packages base)
scheme@(guile-user)> coreutils
$1 = #<package coreutils@8.29 gnu/packages/base.scm:327 3e28300>
```

Además, `guix repl` implementa un protocolo del REPL simple legible por máquinas para su uso por `(guix inferior)`, una facilidad para interactuar con *inferiores*, procesos separados que ejecutan una revisión de Guix potencialmente distinta.

Las opciones disponibles son las siguientes:

`--list-types`
: Display the *TYPE* options for `guix repl --type=TYPE` and exit.

`--type=`*tipo*
`-t` *tipo*
: Inicia un REPL del *TIPO* dado, que puede ser uno de los siguientes:

  `guile`
  : Es el predeterminado, y lanza una sesión interactiva Guile estándar con todas las características.

`machine` Lanza un REPL que usa el protocolo legible por máquinas. Este es el protocolo con el que el módulo `(guix inferior)` se comunica.

`--listen=`*destino*
Por defecto, `guix repl` lee de la entrada estándar y escribe en la salida estándar. Cuando se pasa esta opción, en vez de eso escuchará las conexiones en *destino*. Estos son ejemplos de opciones válidas:

`--listen=tcp:37146`
Acepta conexiones locales por el puerto 37146.

`--listen=unix:/tmp/socket`
Acepta conexiones a través del socket de dominio Unix `/tmp/socket`.

`--interactive`
`-i` Launch the interactive REPL after *file* is executed.

`--load-path=`*directorio*
`-L` *directorio*
Añade *directorio* al frente de la ruta de búsqueda de módulos de paquetes (see Section 8.1 [Módulos de paquetes], page 101).

Esto permite a las usuarias definir sus propios paquetes y hacerlos visibles al guión o a la sesión interactiva.

`-q` Inhibe la carga del archivo `~/.guile`. De manera predeterminada, dicho archivo de configuración se carga al lanzar una sesión interactiva de `guile`.

## 8.14 Using Guix Interactively

The `guix repl` command gives you access to a warm and friendly *read-eval-print loop* (REPL) (see Section 8.13 [Invocación de guix repl], page 177). If you're getting into Guix programming—defining your own packages, writing manifests, defining services for Guix System or Guix Home, etc.—you will surely find it convenient to toy with ideas at the REPL.

If you use Emacs, the most convenient way to do that is with Geiser (see Section 22.5 [La configuración perfecta], page 740), but you do not have to use Emacs to enjoy the REPL. When using `guix repl` or `guile` in the terminal, we recommend using Readline for completion and Colorized to get colorful output. To do that, you can run:

```
guix install guile guile-readline guile-colorized
```

... and then create a `.guile` file in your home directory containing this:

```
(use-modules (ice-9 readline) (ice-9 colorized))

(activate-readline)
(activate-colorized)
```

The REPL lets you evaluate Scheme code; you type a Scheme expression at the prompt, and the REPL prints what it evaluates to:

```
$ guix repl
scheme@(guix-user)> (+ 2 3)
```

```
$1 = 5
scheme@(guix-user)> (string-append "a" "b")
$2 = "ab"
```

It becomes interesting when you start fiddling with Guix at the REPL. The first thing you'll want to do is to "import" the `(guix)` module, which gives access to the main part of the programming interface, and perhaps a bunch of useful Guix modules. You could type `(use-modules (guix))`, which is valid Scheme code to import a module (see Section "Using Guile Modules" in *GNU Guile Reference Manual*), but the REPL provides the `use` *command* as a shorthand notation (see Section "REPL Commands" in *GNU Guile Reference Manual*):

```
scheme@(guix-user)> ,use (guix)
scheme@(guix-user)> ,use (gnu packages base)
```

Notice that REPL commands are introduced by a leading comma. A REPL command like `use` is not valid Scheme code; it's interpreted specially by the REPL.

Guix extends the Guile REPL with additional commands for convenience. Among those, the `build` command comes in handy: it ensures that the given file-like object is built, building it if needed, and returns its output file name(s). In the example below, we build the `coreutils` and `grep` packages, as well as a "computed file" (see Section 8.12 [Expresiones-G], page 167), and we use the `scandir` procedure to list the files in Grep's `/bin` directory:

```
scheme@(guix-user)> ,build coreutils
$1 = "/gnu/store/...-coreutils-8.32-debug"
$2 = "/gnu/store/...-coreutils-8.32"
scheme@(guix-user)> ,build grep
$3 = "/gnu/store/...-grep-3.6"
scheme@(guix-user)> ,build (computed-file "x" #~(mkdir #$output))
building /gnu/store/...-x.drv...
$4 = "/gnu/store/...-x"
scheme@(guix-user)> ,use(ice-9 ftw)
scheme@(guix-user)> (scandir (string-append $3 "/bin"))
$5 = ("." ".." "egrep" "fgrep" "grep")
```

As a packager, you may be willing to inspect the build phases or flags of a given package; this is particularly useful when relying a lot on inheritance to define package variants (see Section 8.3 [Definición de variantes de paquetes], page 114) or when package arguments are a result of some computation, both of which can make it harder to foresee what ends up in the package arguments. Additional commands let you inspect those package arguments:

```
scheme@(guix-user)> ,phases grep
$1 = (modify-phases %standard-phases
       (add-after 'install 'fix-egrep-and-fgrep
         (lambda* (#:key outputs #:allow-other-keys)
           (let* ((out (assoc-ref outputs "out"))
                  (bin (string-append out "/bin")))
             (substitute* (list (string-append bin "/egrep")
                                (string-append bin "/fgrep"))
               (("^exec grep")
                (string-append "exec " bin "/grep")))))))
```

```
scheme@(guix-user)> ,configure-flags findutils
$2 = (list "--localstatedir=/var")
scheme@(guix-user)> ,make-flags binutils
$3 = '("MAKEINFO=true")
```

At a lower-level, a useful command is `lower`: it takes a file-like object and "lowers" it into a derivation (see Section 8.10 [Derivaciones], page 159) or a store file:

```
scheme@(guix-user)> ,lower grep
$6 = #<derivation /gnu/store/...-grep-3.6.drv => /gnu/store/...-grep-3.6 7f0e639115f0>
scheme@(guix-user)> ,lower (plain-file "x" "Hello!")
$7 = "/gnu/store/...-x"
```

The full list of REPL commands can be seen by typing `,help guix` and is given below for reference.

`build` *object* [REPL command]
: Lower *object* and build it if it's not already built, returning its output file name(s).

`lower` *object* [REPL command]
: Lower *object* into a derivation or store file name and return it.

`verbosity` *level* [REPL command]
: Change build verbosity to *level*.

  This is similar to the `--verbosity` command-line option (see Section 9.1.1 [Opciones comunes de construcción], page 181): level 0 means total silence, level 1 shows build events only, and higher levels print build logs.

`phases` *package* [REPL command]
`configure-flags` *package* [REPL command]
`make-flags` *package* [REPL command]
: These REPL commands return the value of one element of the `arguments` field of *package* (see Section 8.2.1 [Referencia de package], page 105): the first one show the staged code associated with `#:phases` (see Section 8.6 [Fases de construcción], page 143), the second shows the code for `#:configure-flags`, and `,make-flags` returns the code for `#:make-flags`.

`run-in-store` *exp* [REPL command]
: Run *exp*, a monadic expression, through the store monad. See Section 8.11 [La mónada del almacén], page 162, for more information.

`enter-store-monad` [REPL command]
: Enter a new REPL to evaluate monadic expressions (see Section 8.11 [La mónada del almacén], page 162). You can quit this "inner" REPL by typing `,q`.

# 9 Utilidades

Esta sección describe las utilidades de línea de órdenes de Guix. Algunas de ellas están orientadas principalmente para desarrolladoras y usuarias que escriban definiciones de paquetes nuevas, mientras que otras son útiles de manera más general. Complementan la interfaz programática Scheme de Guix de modo conveniente.

## 9.1 Invocación de `guix build`

La orden `guix build` construye paquetes o derivaciones y sus dependencias, e imprime las rutas del almacén resultantes. Fíjese que no modifica el perfil de la usuaria—este es el trabajo de la orden `guix package` (see Section 5.2 [Invocación de guix package], page 36). Por tanto, es útil principalmente para las desarrolladoras de la distribución.

La sintaxis general es:

```
guix build opciones paquete-o-derivación...
```

Como ejemplo, la siguiente orden construye las últimas versiones de Emacs y Guile, muestra sus log de construcción, y finalmente muestra los directorios resultantes:

```
guix build emacs guile
```

De forma similar, la siguiente orden construye todos los paquetes disponibles:

```
guix build --quiet --keep-going \
  $(guix package -A | awk '{ print $1 "@" $2 }')
```

*paquete-o-derivación* puede ser tanto el nombre de un paquete que se encuentra en la distribución de software como `coreutils` o `coreutils@8.20`, o una derivación como `/gnu/store/...-coreutils-8.19.drv`. En el primer caso, el paquete de nombre (y opcionalmente versión) correspondiente se busca entre los módulos de la distribución GNU (see Section 8.1 [Módulos de paquetes], page 101).

De manera alternativa, la opción `--expression` puede ser usada para especificar una expresión Scheme que evalúa a un paquete; esto es útil para diferenciar entre varios paquetes con el mismo nombre o si se necesitan variaciones del paquete.

Puede haber cero o más *opciones*. Las opciones disponibles se describen en la subsección siguiente.

### 9.1.1 Opciones comunes de construcción

Un número de opciones que controlan el proceso de construcción son comunes a `guix build` y otras órdenes que pueden lanzar construcciones, como `guix package` o `guix archive`. Son las siguientes:

`--load-path=`*directorio*

`-L` *directorio*

: Añade *directorio* al frente de la ruta de búsqueda de módulos de paquetes (see Section 8.1 [Módulos de paquetes], page 101).

  Esto permite a las usuarias definir sus propios paquetes y hacerlos visibles a las herramientas de línea de órdenes.

`--keep-failed`

`-K`

: Mantiene los árboles de construcción de las construcciones fallidas. Por tanto, si una construcción falla, su árbol de construcción se mantiene bajo `/tmp`, en un

directorio cuyo nombre se muestra al final del log de construcción. Esto es útil cuando se depuran problemas en la construcción. See Section 9.1.4 [Depuración de fallos de construcción], page 194, para trucos y consejos sobre cómo depurar problemas en la construcción.

Esta opción implica `--no-offload`, y no tiene efecto cuando se conecta a un daemon remoto con una URI `guix://` (see Section 8.9 [El almacén], page 157).

`--keep-going`
`-k` Seguir adelante cuando alguna de las derivaciones de un fallo durante la construcción; devuelve una única vez todas las construcciones que se han completado o bien han fallado.

El comportamiento predeterminado es parar tan pronto una de las derivaciones especificadas falle.

`--dry-run`
`-n` No construye las derivaciones.

`--fallback`
Cuando la sustitución de un binario preconstruido falle, intenta la construcción local de paquetes (see Section 5.3.6 [Fallos en las sustituciones], page 50).

`--substitute-urls=`*urls*
Considera *urls* la lista separada por espacios de URLs de fuentes de sustituciones, anulando la lista predeterminada de URLs de `guix-daemon` (see [`guix-daemon URLs`], page 13).

Significa que las sustituciones puede ser descargadas de *urls*, mientras que estén firmadas por una clave autorizada por la administradora del sistema (see Section 5.3 [Sustituciones], page 46).

Cuando *urls* es la cadena vacía, las sustituciones están efectivamente desactivadas.

`--no-substitutes`
No usa sustituciones para la construcción de productos. Esto es, siempre realiza las construcciones localmente en vez de permitir la descarga de binarios preconstruidos (see Section 5.3 [Sustituciones], page 46).

`--no-grafts`
No "injerta" paquetes. En la práctica esto significa que las actualizaciones de paquetes disponibles como injertos no se aplican. See Chapter 19 [Actualizaciones de seguridad], page 723, para más información sobre los injertos.

`--rounds=`*n*
Construye cada derivación *n* veces seguidas, y lanza un error si los resultados de las construcciones consecutivas no son idénticos bit-a-bit.

Esto es útil para la detección de procesos de construcción no-deterministas. Los procesos de construcción no-deterministas son un problema puesto que prácticamente imposibilitan a las usuarias la *verificación* de la autenticidad de binarios proporcionados por terceras partes. See Section 9.12 [Invocación de guix challenge], page 230, para más sobre esto.

Cuando se usa conjuntamente con `--keep-failed`, la salida que difiere se mantiene en el almacén, bajo `/gnu/store/...-check`. Esto hace fácil buscar diferencias entre los dos resultados.

`--no-offload`
: No usa la delegación de construcciones en otras máquinas (see Section 2.2.2 [Configuración de delegación del daemon], page 7). Es decir, siempre realiza las construcciones de manera local en vez de delegar construcciones a máquinas remotas.

`--max-silent-time=`*segundos*
: Cuando la construcción o sustitución permanece en silencio más de *segundos*, la finaliza e informa de un fallo de construcción.

  Por defecto, se respeta la configuración del daemon (see Section 2.3 [Invocación de guix-daemon], page 12).

`--timeout=`*segundos*
: Del mismo modo, cuando el proceso de construcción o sustitución dura más de *segundos*, lo termina e informa un fallo de construcción.

  Por defecto, se respeta la configuración del daemon (see Section 2.3 [Invocación de guix-daemon], page 12).

`-v` *nivel*
`--verbosity=`*nivel*
: Use the given verbosity *level*, an integer. Choosing 0 means that no output is produced, 1 is for quiet output; 2 is similar to 1 but it additionally displays download URLs; 3 shows all the build log output on standard error.

`--cores=`*n*
`-c` *n*
: Permite usar *n* núcleos de la CPU para la construcción. El valor especial `0` significa usar tantos como núcleos haya en la CPU.

`--max-jobs=`*n*
`-M` *n*
: Permite como máximo *n* trabajos de construcción en paralelo. See Section 2.3 [Invocación de guix-daemon], page 12, para detalles acerca de esta opción y la opción equivalente de `guix-daemon`.

`--debug=`*nivel*
: Usa el nivel de detalle proporcionado en los mensajes procedentes del daemon de construcción. *nivel* debe ser un entero entre 0 y 5; valores mayores indican una salida más detallada. Establecer un nivel de 4 o superior puede ser útil en la depuración de problemas de configuración con el daemon de construcción.

Tras las cortinas, `guix build` es esencialmente una interfaz al procedimiento `package-derivation` del módulo `(guix packages)`, y al procedimiento `build-derivations` del módulo `(guix derivations)`.

Además de las opciones proporcionadas explícitamente en la línea de órdenes, `guix build` y otras órdenes `guix` que permiten la construcción respetan el contenido de la variable de entorno `GUIX_BUILD_OPTIONS`.

`GUIX_BUILD_OPTIONS` [Variable de entorno]

Las usuarias pueden definir esta variable para que contenga una lista de opciones de línea de órdenes que se usarán automáticamente por `guix build` y otras órdenes `guix` que puedan realizar construcciones, como en el ejemplo siguiente:

```
$ export GUIX_BUILD_OPTIONS="--no-substitutes -c 2 -L /foo/bar"
```

Estas opciones se analizan independientemente, y el resultado se añade a continuación de las opciones de línea de órdenes.

### 9.1.2 Opciones de transformación de paquetes

Otro conjunto de opciones de línea de órdenes permitidas por `guix build` y también `guix package` son las *opciones de transformación de paquetes*. Son opciones que hacen posible la definición de *variaciones de paquetes*—por ejemplo, paquetes construidos con un código fuente diferente. Es una forma conveniente de crear paquetes personalizados al vuelo sin tener que escribir las definiciones de las variaciones del paquete (see Section 8.2 [Definición de paquetes], page 102).

Las opciones de transformación del paquete se preservan con las actualizaciones: `guix upgrade` intenta aplicar las opciones de transformación usadas inicialmente al crear el perfil para actualizar los paquetes.

Las opciones disponibles se enumeran a continuación. La mayor parte de las ordenes los aceptan, así como la opción `--help-transform` que enumera todas las opciones disponibles y una sinópsis (estas opciones no se muestran en la salida de `--help` por brevedad).

`--tune[=`*cpu*`]`

Use versions of the packages marked as "tunable" optimized for *cpu*. When *cpu* is `native`, or when it is omitted, tune for the CPU on which the `guix` command is running.

Valid *cpu* names are those recognized by the underlying compiler, by default the GNU Compiler Collection. On x86_64 processors, this includes CPU names such as `nehalem`, `haswell`, and `skylake` (see Section "x86 Options" in *Using the GNU Compiler Collection (GCC)*).

As new generations of CPUs come out, they augment the standard instruction set architecture (ISA) with additional instructions, in particular instructions for single-instruction/multiple-data (SIMD) parallel processing. For example, while Core2 and Skylake CPUs both implement the x86_64 ISA, only the latter supports AVX2 SIMD instructions.

The primary gain one can expect from `--tune` is for programs that can make use of those SIMD capabilities *and* that do not already have a mechanism to select the right optimized code at run time. Packages that have the `tunable?` property set are considered *tunable packages* by the `--tune` option; a package definition with the property set looks like this:

```
(package
  (name "hello-simd")
  ;; ...

  ;; This package may benefit from SIMD extensions so
```

```
;; mark it as "tunable".
(properties '((tunable? . #t))))
```

Other packages are not considered tunable. This allows Guix to use generic binaries in the cases where tuning for a specific CPU is unlikely to provide any gain.

Tuned packages are built with `-march=`*`CPU`*; under the hood, the `-march` option is passed to the actual wrapper by a compiler wrapper. Since the build machine may not be able to run code for the target CPU micro-architecture, the test suite is not run when building a tuned package.

To reduce rebuilds to the minimum, tuned packages are *grafted* onto packages that depend on them (see Chapter 19 [Actualizaciones de seguridad], page 723). Thus, using `--no-grafts` cancels the effect of `--tune`.

We call this technique *package multi-versioning*: several variants of tunable packages may be built, one for each CPU variant. It is the coarse-grain counterpart of *function multi-versioning* as implemented by the GNU tool chain (see Section "Function Multiversioning" in *Using the GNU Compiler Collection (GCC)*).

`--with-source=`*`fuente`*
`--with-source=`*`paquete=fuente`*
`--with-source=`*`paquete@versión=fuente`*

Usa *fuente* como la fuente de *paquete*, y *versión* como su número de versión. *fuente* debe ser un nombre de archivo o una URL, como en `guix download` (see Section 9.3 [Invocación de guix download], page 196).

Cuando se omite *paquete*, se toma el nombre de paquete especificado en la línea de ordenes que coincide con el nombre base de *fuente*—por ejemplo, si *fuente* fuese `/src/guile-2.0.10.tar.gz`, el paquete correspondiente sería `guile`.

Del mismo modo, si se omite *versión*, la cadena de versión se deduce de *ðuente*; en el ejemplo previo sería `2.0.10`.

Esta opción permite a las usuarias probar versiones del paquete distintas a las proporcionadas en la distribución. El ejemplo siguiente descarga `ed-1.7.tar.gz` de un espejo GNU y lo usa como la fuente para el paquete `ed`:

```
guix build ed --with-source=mirror://gnu/ed/ed-1.4.tar.gz
```

As a developer, `--with-source` makes it easy to test release candidates, and even to test their impact on packages that depend on them:

```
guix build elogind --with-source=.../shepherd-0.9.0rc1.tar.gz
```

. . . o la construcción desde una revisión en un entorno limpio:

```
$ git clone git://git.sv.gnu.org/guix.git
$ guix build guix --with-source=guix@1.0=./guix
```

`--with-input=`*`paquete=reemplazo`*

Substituye dependencias de *paquete* por dependencias de *reemplazo*. *paquete* debe ser un nombre de paquete, y *reemplazo* debe ser una especificación de paquete como `guile` o `guile@1.8`.

For instance, the following command builds Guix, but replaces its dependency on the current stable version of Guile with a dependency on the legacy version of Guile, `guile@2.2`:

```
guix build --with-input=guile=guile@2.2 guix
```

This is a recursive, deep replacement. So in this example, both `guix` and its dependency `guile-json` (which also depends on `guile`) get rebuilt against `guile@2.2`.

This is implemented using the `package-input-rewriting/spec` Scheme procedure (see Section 8.2 [Definición de paquetes], page 102).

`--with-graft=`*paquete*`=`*reemplazo*

Es similar a `--with-input` pero con una diferencia importante: en vez de reconstruir la cadena de dependencias completa, *reemplazo* se construye y se *injerta* en los binarios que inicialmente hacían referencia a *paquete*. See Chapter 19 [Actualizaciones de seguridad], page 723, para más información sobre injertos.

Por ejemplo, la orden siguiente injerta la versión 3.5.4 de GnuTLS en Wget y todas sus dependencias, substituyendo las referencias a la versión de GnuTLS que tienen actualmente:

```
guix build --with-graft=gnutls=gnutls@3.5.4 wget
```

Esta opción tiene la ventaja de ser mucho más rápida que la reconstrucción de todo. Pero hay una trampa: funciona si y solo si *paquete* y *reemplazo* son estrictamente compatibles—por ejemplo, si proporcionan una biblioteca, la interfaz binaria de aplicación (ABI) de dichas bibliotecas debe ser compatible. Si *reemplazo* es incompatible de alguna manera con *paquete*, el paquete resultante puede no ser usable. ¡Úsela con precaución!

`--with-debug-info=`*paquete*

Construye *paquete* de modo que preserve su información de depuración y lo injerta en los paquetes que dependan de él. Es útil si *paquete* no proporciona ya información de depuración como una salida `debug` (see Chapter 17 [Instalación de archivos de depuración], page 718).

Por ejemplo, supongamos que está experimentando un fallo en Inkscape y querría ver qué pasa en GLib, una biblioteca con mucha profundidad en el grafo de dependencias de Inkscape. GLib no tiene una salida `debug`, de modo que su depuración es difícil. Afortunadamente puede reconstruir GLib con información de depuración e incorporarla a Inkscape:

```
guix install inkscape --with-debug-info=glib
```

Únicamente GLib necesita una reconstrucción por lo que esto tarda un tiempo razonable. See Chapter 17 [Instalación de archivos de depuración], page 718, para obtener más información.

> **Nota:** Esta opción funciona en su implementación interna proporcionando '`#:strip-binaries? #f`' al sistema de construcción del paquete en cuestión (see Section 8.5 [Sistemas de construcción], page 123). La mayor parte de sistemas de construcción implementan dicha opción, pero algunos no lo hacen. En este caso caso se emite un error.

De igual modo, si se construye un paquete C/C++ sin la opción `-g` (lo que no es habitual que ocurra), la información de depuración seguirá sin estar disponible incluso cuando `#:strip-binaries?` sea falso.

`--with-c-toolchain=`*paquete*`=`*cadena*

Esta opción cambia la compilación de *paquete* y todo lo que dependa de él de modo que se constuya con *cadena* en vez de la cadena de herramientas de construcción para C/C++ de GNU predeterminada.

Considere este ejemplo:

```
guix build octave-cli \
  --with-c-toolchain=fftw=gcc-toolchain@10 \
  --with-c-toolchain=fftwf=gcc-toolchain@10
```

La orden anterior construye una variante de los paquetes `fftw` y `fftwf` usando la versión 10 de `gcc-toolchain` en vez de la cadena de herramientas de construcción predeterminada, y construye una variante de la interfaz de línea de órdenes GNU Octave que hace uso de ellos. El propio paquete de GNU Octave también se construye con `gcc-toolchain@10`.

Este otro ejemplo construye la biblioteca Hardware Locality (`hwloc`) y los paquetes que dependan de ella hasta `intel-mpi-benchmarks` con el compilador de C Clang:

```
guix build --with-c-toolchain=hwloc=clang-toolchain \
           intel-mpi-benchmarks
```

**Nota:** There can be application binary interface (ABI) incompatibilities among tool chains. This is particularly true of the C++ standard library and run-time support libraries such as that of OpenMP. By rebuilding all dependents with the same tool chain, `--with-c-toolchain` minimizes the risks of incompatibility but cannot entirely eliminate them. Choose *package* wisely.

`--with-git-url=`*paquete*`=`*url*

Construye *paquete* desde la última revisión de la rama `master` del repositorio Git en *url*. Los submódulos del repositorio Git se obtienen de forma recursiva.

Por ejemplo, la siguiente orden construye la biblioteca NumPy de Python contra la última revisión de la rama master de Python en sí:

```
guix build python-numpy \
  --with-git-url=python=https://github.com/python/cpython
```

Esta opción también puede combinarse con `--with-branch` o `--with-commit` (véase más adelante).

Obviamente, ya que se usa la última revisión de la rama proporcionada, el resultado de dicha orden varia con el tiempo. No obstante es una forma conveniente de reconstruir una pila completa de software contra las últimas revisiones de uno o varios paquetes. Esto es particularmente útil en el contexto de integración continua (CI).

Los directorios de trabajo se conservan en caché en `~/.cache/guix/checkouts` para agilizar accesos consecutivos al mismo repositorio. Puede desear limpiarla de vez en cuando para ahorrar espacio en el disco.

`--with-branch=`*paquete*`=`*rama*

Construye *paquete* desde la última revisión de *rama*. Si el campo `source` de *paquete* es un origen con el método `git-fetch` (see Section 8.2.2 [Referencia de origin], page 110) o un objeto `git-checkout`, la URL del repositorio se toma de dicho campo `source`. En otro caso, se debe especificar la URL del repositorio Git mediante el uso de `--with-git-url`.

Por ejemplo, la siguiente orden construye `guile-sqlite3` desde la última revisión de su rama `master` y, una vez hecho, construye `guix` (que depende de él) y `cuirass` (que depende de `guix`) en base a esta construcción específica de `guile-sqlite3`:

```
guix build --with-branch=guile-sqlite3=master cuirass
```

`--with-commit=`*paquete*`=`*revisión*

This is similar to `--with-branch`, except that it builds from *commit* rather than the tip of a branch. *commit* must be a valid Git commit SHA1 identifier, a tag, or a `git describe` style identifier such as `1.0-3-gabc123`.

`--with-patch=`*package*`=`*file*

Add *file* to the list of patches applied to *package*, where *package* is a spec such as `python@3.8` or `glibc`. *file* must contain a patch; it is applied with the flags specified in the `origin` of *package* (see Section 8.2.2 [Referencia de origin], page 110), which by default includes `-p1` (see Section "patch Directories" in *Comparing and Merging Files*).

As an example, the command below rebuilds Coreutils with the GNU C Library (glibc) patched with the given patch:

```
guix build coreutils --with-patch=glibc=./glibc-frob.patch
```

In this example, glibc itself as well as everything that leads to Coreutils in the dependency graph is rebuilt.

`--with-configure-flag=`*package*`=`*flag*

Append *flag* to the configure flags of *package*, where *package* is a spec such as `guile@3.0` or `glibc`. The build system of *package* must support the `#:configure-flags` argument.

For example, the command below builds GNU Hello with the configure flag `--disable-nls`:

```
guix build hello --with-configure-flag=hello=--disable-nls
```

The following command passes an extra flag to `cmake` as it builds `lapack`:

```
guix build lapack \
  --with-configure-flag=lapack=-DBUILD_SHARED_LIBS=OFF
```

**Nota:** Under the hood, this option works by passing the '`#:configure-flags`' argument to the build system of the package of interest (see Section 8.5 [Sistemas de construcción], page 123). Most build systems support that option but some do not. In that case, an error is raised.

`--with-latest=`*package*
`--with-version=`*package*`=`*version*

So you like living on the bleeding edge? The `--with-latest` option is for you! It replaces occurrences of *package* in the dependency graph with its latest upstream version, as reported by `guix refresh` (see Section 9.6 [Invocación de guix refresh], page 207).

It does so by determining the latest upstream release of *package* (if possible), downloading it, and authenticating it *if* it comes with an OpenPGP signature.

As an example, the command below builds Guix against the latest version of Guile-JSON:

```
guix build guix --with-latest=guile-json
```

The `--with-version` works similarly except that it lets you specify that you want precisely *version*, assuming that version exists upstream. For example, to spawn a development environment with SciPy built against version 1.22.4 of NumPy (skipping its test suite because hey, we're not gonna wait this long), you would run:

```
guix shell python python-scipy --with-version=python-numpy=1.22.4
```

**Aviso:** Because they depend on source code published at a given point in time on upstream servers, deployments made with `--with-latest` and `--with-version` may be non-reproducible: source might disappear or be modified in place on the servers.

To deploy old software versions without compromising on reproducibility, see Section 5.8 [Invocación de guix time-machine], page 61.

There are limitations. First, in cases where the tool cannot or does not know how to authenticate source code, you are at risk of running malicious code; a warning is emitted in this case. Second, this option simply changes the source used in the existing package definitions, which is not always sufficient: there might be additional dependencies that need to be added, patches to apply, and more generally the quality assurance work that Guix developers normally do will be missing.

You've been warned! When those limitations are acceptable, it's a snappy way to stay on top. We encourage you to submit patches updating the actual package definitions once you have successfully tested an upgrade with `--with-latest` (see Chapter 22 [Contribuir], page 734).

`--without-tests=`*paquete*

Construye *paquete* sin ejecutar su batería de pruebas. Puede ser útil en situaciones en las que quiera omitir una larga batería de pruebas de un paquete intermedio, o si la batería de pruebas no falla de manera determinista. Debe usarse con cuidado, puesto que la ejecución de la batería de pruebas es una buena forma de asegurarse de que el paquete funciona como se espera.

La desactivación de las pruebas conduce a diferentes elementos en el almacén. Por tanto, cuando use esta opción, cualquier objeto que dependa de *paquete* debe ser reconstruido, como en este ejemplo:

```
guix install --without-tests=python python-notebook
```

La orden anterior instala `python-notebook` sobre un paquete `python` construido sin ejecutar su batería de pruebas. Para hacerlo, también reconstruye todos los paquetes que dependen de `python`, incluyendo el propio `pyhton-notebook`.

De manera interna, `--without-tests` depende del cambio de la opción `#:tests?` de la fase `check` del paquete (see Section 8.5 [Sistemas de construcción], page 123). Tenga en cuenta que algunos paquetes usan una fase `check` personalizada que no respeta el valor de configuración `#:tests? #f`. Por tanto, `--without-tests` no tiene ningún efecto en dichos paquetes.

¿Se pregunta cómo conseguir el mismo efecto usando código Scheme, por ejemplo en su manifiesto, o cómo escribir su propia transformación de paquetes? See Section 8.3 [Definición de variantes de paquetes], page 114, para obtener una descripción de la interfaz programática disponible.

### 9.1.3 Opciones de construcción adicionales

Las opciones de línea de ordenes presentadas a continuación son específicas de `guix build`.

`--quiet`
`-q`
: Construye silenciosamente, sin mostrar el registro de construcción; es equivalente a `--verbosity=0`. Al finalizar, el registro de construcción se mantiene en `/var` (o similar) y puede recuperarse siempre mediante el uso de la opción `--log-file`.

`--file=`*archivo*
`-f` *archivo*
: Construye el paquete, derivación u otro objeto tipo-archivo al que evalúa el código en *archivo* (see Section 8.12 [Expresiones-G], page 167).

Como un ejemplo, *archivo* puede contener una definición como esta (see Section 8.2 [Definición de paquetes], page 102):

```
(use-modules (guix)
             (guix build-system gnu)
             (guix licenses))

(package
  (name "hello")
  (version "2.10")
  (source (origin
            (method url-fetch)
            (uri (string-append "mirror://gnu/hello/hello-" version
                                ".tar.gz"))
            (sha256
             (base32
              "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i"))))
  (build-system gnu-build-system)
  (synopsis "Hello, GNU world: An example GNU package")
  (description "Guess what GNU Hello prints!")
  (home-page "http://www.gnu.org/software/hello/")
  (license gpl3+))
```

El *archivo* también puede contener una representación en JSON de una o más definiciones de paquete. Ejecutar `guix build -f` en `hello.json` con el siguiente contenido resultaría en la construcción de los paquetes `myhello` y `greeter`:

```
[
  {
    "name": "myhello",
    "version": "2.10",
    "source": "mirror://gnu/hello/hello-2.10.tar.gz",
    "build-system": "gnu",
    "arguments": {
      "tests?": false
    },
    "home-page": "https://www.gnu.org/software/hello/",
    "synopsis": "Hello, GNU world: An example GNU package",
    "description": "GNU Hello prints a greeting.",
    "license": "GPL-3.0+",
    "native-inputs": ["gettext"]
  },
  {
    "name": "greeter",
    "version": "1.0",
    "source": "mirror://gnu/hello/hello-2.10.tar.gz",
    "build-system": "gnu",
    "arguments": {
      "test-target": "foo",
      "parallel-build?": false
    },
    "home-page": "https://example.com/",
    "synopsis": "Greeter using GNU Hello",
    "description": "This is a wrapper around GNU Hello.",
    "license": "GPL-3.0+",
    "inputs": ["myhello", "hello"]
  }
]
```

`--manifest=`*manifiesto*
`-m` *manifiesto*
: Construye todos los paquetes listados en el *manifiesto* proporcionado (see [profile-manifest], page 40).

`--expression=`*expr*
`-e` *expr*
: Construye el paquete o derivación a la que evalúa *expr*.

  Por ejemplo, *expr* puede ser `(@ (gnu packages guile) guile-1.8)`, que designa sin ambigüedad a esta variante específica de la versión 1.8 de Guile.

  De manera alternativa, *expr* puede ser una expresión-G, en cuyo caso se usa como un programa de construcción pasado a `gexp->derivation` (see Section 8.12 [Expresiones-G], page 167).

Por último, *expr* puede hacer referencia a un procedimiento mónadico sin parámetros (see Section 8.11 [La mónada del almacén], page 162). El procedimiento debe devolver una derivación como un valor monádico, el cual después se pasa a través de `run-with-store`.

`--source`
`-S` Construye las derivaciones de las fuentes de los paquetes, en vez de los paquetes mismos.

Por ejemplo, `guix build -S gcc` devuelve algo como `/gnu/store/...-gcc-4.7.2.tar.bz2`, el cual es el archivador tar de fuentes de GCC.

El archivador tar devuelto es el resultado de aplicar cualquier parche y fragmento de código en el origen (campo `origin`) del paquete (see Section 8.2 [Definición de paquetes], page 102).

As with other derivations, the result of building a source derivation can be verified using the `--check` option (see [build-check], page 193). This is useful to validate that a (potentially already built or substituted, thus cached) package source matches against its declared hash.

Tenga en cuenta que `guix build -S` compila las fuentes únicamente de los paquetes especificados. Esto no incluye las dependencias enlazadas estáticamente y por sí mismas son insuficientes para reproducir los paquetes.

`--sources`
Obtiene y devuelve las fuentes de *paquete-o-derivación* y todas sus dependencias, de manera recursiva. Esto es útil para obtener una copia local de todo el código fuente necesario para construir los *paquetes*, le permite construirlos llegado el momento sin acceso a la red. Es una extensión de la opción `--source` y puede aceptar uno de los siguientes valores opcionales como parámetro:

`package` Este valor hace que la opción `--sources` se comporte de la misma manera que la opción `--source`.

`all` Construye las derivaciones de las fuentes de todos los paquetes, incluyendo cualquier fuente que pueda enumerarse como entrada (campo `inputs`). Este es el valor predeterminado.

```
$ guix build --sources tzdata
The following derivations will be built:
   /gnu/store/...-tzdata2015b.tar.gz.drv
   /gnu/store/...-tzcode2015b.tar.gz.drv
```

`transitive`
Construye las derivaciones de fuentes de todos los paquetes, así como todas las entradas transitivas de los paquetes. Esto puede usarse, por ejemplo, para obtener las fuentes de paquetes para una construcción posterior sin conexión a la red.

```
$ guix build --sources=transitive tzdata
The following derivations will be built:
   /gnu/store/...-tzcode2015b.tar.gz.drv
   /gnu/store/...-findutils-4.4.2.tar.xz.drv
```

```
/gnu/store/...-grep-2.21.tar.xz.drv
/gnu/store/...-coreutils-8.23.tar.xz.drv
/gnu/store/...-make-4.1.tar.xz.drv
/gnu/store/...-bash-4.3.tar.xz.drv
...
```

`--system=`*sistema*
`-s` *sistema*

Intenta la construcción para *sistema*—por ejemplo, `i686-linux`—en vez del tipo de sistema de la máquina de construcción. La orden `guix build` le permite repetir esta opción varias veces, en cuyo caso construye para todos los sistemas especificados; otras ordenes ignoran opciones `-s` extrañas.

> **Nota:** La opción `--system` es para compilación *nativa* y no debe confundirse con la compilación cruzada. Véase `--target` más adelante para información sobre compilación cruzada.

Un ejemplo de uso de esta opción es en sistemas basados en Linux, que pueden emular diferentes personalidades. Por ejemplo, proporcionar la opción `--system=i686-linux` en un sistema `x86_64-linux`, o la opción `--system=armhf-linux` en un sistema `aarch64-linux`, le permite construir paquetes en un entorno de 32-bits completo.

> **Nota:** La construcción para un sistema `armhf-linux` está disponible de manera incondicional en máquinas `aarch64-linux`, aunque determinadas familias de procesadores aarch64 no lo permitan, notablemente el ThunderX.

De manera similar, cuando la emulación transparente con QEMU y `binfmt_misc` está activada (see Section 11.10.30 [Servicios de virtualización], page 538), puede construir para cualquier sistema para el que un manejador QEMU de `binfmt_misc` esté instalado.

Las construcciones para un sistema distinto al de la máquina que usa se pueden delegar también a una máquina remota de la arquitectura correcta. See Section 2.2.2 [Configuración de delegación del daemon], page 7, para más información sobre delegación.

`--target=`*tripleta*

Compilación cruzada para la *tripleta*, que debe ser una tripleta GNU válida, cómo `"aarch64-linux-gnu"` (see Section “Specifying Target triplets” in *Autoconf*).

`--list-systems`

List all the supported systems, that can be passed as an argument to `--system`.

`--list-targets`

List all the supported targets, that can be passed as an argument to `--target`.

`--check` Reconstruye *paquete-o-derivación*, que ya está disponible en el almacén, y emite un error si los resultados de la construcción no son idénticos bit-a-bit.

Este mecanismo le permite comprobar si sustituciones previamente instaladas son genuinas (see Section 5.3 [Sustituciones], page 46), o si el resultado de la

construcción de un paquete es determinista. See Section 9.12 [Invocación de guix challenge], page 230, para más información de referencia y herramientas.

Cuando se usa conjuntamente con `--keep-failed`, la salida que difiere se mantiene en el almacén, bajo `/gnu/store/...-check`. Esto hace fácil buscar diferencias entre los dos resultados.

`--repair` Intenta reparar los elementos del almacén especificados, si están corruptos, volviendo a descargarlos o mediante su reconstrucción.

Esta operación no es atómica y por lo tanto está restringida a `root`.

`--derivations`
`-d` Devuelve las rutas de derivación, no las rutas de salida, de los paquetes proporcionados.

`--root=`*archivo*
`-r` *archivo*

Hace que *archivo* sea un enlace simbólico al resultado, y lo registra como una raíz del recolector de basura.

Consecuentemente, los resultados de esta invocación de `guix build` se protegen de la recolección de basura hasta que *archivo* se elimine. Cuando se omite esa opción, los resultados son candidatos a la recolección de basura en cuanto la construcción se haya completado. See Section 5.6 [Invocación de guix gc], page 53, para más sobre las raíces del recolector de basura.

`--log-file`

Devuelve los nombres de archivos o URL de los log de construcción para el *paquete-o-derivación* proporcionado, o emite un error si no se encuentran los log de construcción.

Esto funciona independientemente de cómo se especificasen los paquetes o derivaciones. Por ejemplo, las siguientes invocaciones son equivalentes:

```
guix build --log-file $(guix build -d guile)
guix build --log-file $(guix build guile)
guix build --log-file guile
guix build --log-file -e '(@ (gnu packages guile) guile-2.0)'
```

Si no está disponible un registro local, y a menos que se proporcione `--no-substitutes`, la orden busca el registro correspondiente en uno de los servidores de sustituciones (como se especificaron con `--substitute-urls`).

So for instance, imagine you want to see the build log of GDB on `aarch64`, but you are actually on an `x86_64` machine:

```
$ guix build --log-file gdb -s aarch64-linux
https://bordeaux.guix.gnu.org/log/...-gdb-7.10
```

¡Puede acceder libremente a una biblioteca inmensa de log de construcción!

### 9.1.4 Depuración de fallos de construcción

Cuando esté definiendo un paquete nuevo (see Section 8.2 [Definición de paquetes], page 102), probablemente se encuentre que dedicando algún tiempo a depurar y afinar la construcción hasta obtener un resultado satisfactorio. Para hacerlo, tiene que lanzar

manualmente las órdenes de construcción en un entorno tan similar como sea posible al que el daemon de construcción usa.

To that end, the first thing to do is to use the `--keep-failed` or `-K` option of `guix build`, which will keep the failed build tree in `/tmp` or whatever directory you specified as `TMPDIR` (see Section 9.1.1 [Opciones comunes de construcción], page 181).

De ahí en adelante, puede usar `cd` para ir al árbol de la construcción fallida y cargar el archivo `environment-variables`, que contiene todas las definiciones de variables de entorno que existían cuando la construcción falló. Digamos que está depurando un fallo en la construcción del paquete `foo`; una sesión típica sería así:

```
$ guix build foo -K
... build fails
$ cd /tmp/guix-build-foo.drv-0
$ source ./environment-variables
$ cd foo-1.2
```

Ahora puede invocar órdenes (casi) como si fuese el daemon y encontrar los errores en su proceso de construcción.

A veces ocurre que, por ejemplo, las pruebas de un paquete pasan cuando las ejecuta manualmente pero fallan cuando el daemon las ejecuta. Esto puede suceder debido a que el daemon construye dentro de contenedores donde, al contrario que en nuestro entorno previo, el acceso a la red no está disponible, `/bin/sh` no existe, etc. (see Section 2.2.1 [Configuración del entorno de construcción], page 6).

En esos casos, puede tener que inspeccionar el proceso de construcción desde un contenedor similar al creado por el daemon de construcción:

```
$ guix build -K foo
...
$ cd /tmp/guix-build-foo.drv-0
$ guix shell --no-grafts -C -D foo strace gdb
[env]# source ./environment-variables
[env]# cd foo-1.2
```

Here, `guix shell -C` creates a container and spawns a new shell in it (see Section 7.1 [Invoking guix shell], page 79). The `strace gdb` part adds the `strace` and `gdb` commands to the container, which you may find handy while debugging. The `--no-grafts` option makes sure we get the exact same environment, with ungrafted packages (see Chapter 19 [Actualizaciones de seguridad], page 723, for more info on grafts).

Para acercarnos más al contenedor usado por el daemon de construcción, podemos eliminar `/bin/sh`:

```
[env]# rm /bin/sh
```

(Don't worry, this is harmless: this is all happening in the throw-away container created by `guix shell`.)

La orden `strace` probablemente no esté en la ruta de búsqueda, pero podemos ejecutar:

```
[env]# $GUIX_ENVIRONMENT/bin/strace -f -o log make check
```

De este modo, no solo habrá reproducido las variables de entorno que usa el daemon, también estará ejecutando el proceso de construcción en un contenedor similar al usado por el daemon.

## 9.2 Invocación de `guix edit`

¡Tantos paquetes, tantos archivos de fuentes! La orden `guix edit` facilita la vida de las usuarias y empaquetadoras apuntando su editor al archivo de fuentes que contiene la definición de los paquetes especificados. Por ejemplo:

```
guix edit gcc@4.9 vim
```

ejecuta el programa especificado en la variable de entorno `VISUAL` o en `EDITOR` para ver la receta de GCC 4.9.3 y la de Vim.

Si está usando una copia de trabajo de Git de Guix (see Section 22.2 [Construcción desde Git], page 735), o ha creado sus propios paquetes en `GUIX_PACKAGE_PATH` (see Section 8.1 [Módulos de paquetes], page 101), será capaz de editar las recetas de los paquetes. En otros casos, podrá examinar las recetas en modo de lectura únicamente para paquetes actualmente en el almacén.

En vez de `GUIX_PACKAGE_PATH`, la opción de línea de ordenes `--load-path=`*`directorio`* (o en versión corta `-L` *`directorio`*) le permite añadir *directorio* al inicio de la ruta de búsqueda de módulos de paquete y hacer visibles sus propios paquetes.

## 9.3 Invocación de `guix download`

Durante la escritura de una definición de paquete, las desarrolladoras típicamente tienen que descargar un archivador tar de fuentes, calcular su hash SHA256 y escribir ese hash en la definición del paquete (see Section 8.2 [Definición de paquetes], page 102). La herramienta `guix download` ayuda con esta tarea: descarga un archivo de la URI proporcionada, lo añade al almacén e imprime tanto su nombre de archivo en el almacén como su hash SHA256.

El hecho de que el archivo descargado se añada al almacén ahorra ancho de banda: cuando el desarrollador intenta construir el paquete recién definido con `guix build`, el archivador de fuentes no tiene que descargarse de nuevo porque ya está en el almacén. También es una forma conveniente de conservar archivos temporalmente, que pueden ser borrados en un momento dado (see Section 5.6 [Invocación de guix gc], page 53).

La orden `guix download` acepta las mismas URI que las usadas en las definiciones de paquetes. En particular, permite URI `mirror://`. Las URI `https` (HTTP sobre TLS) se aceptan *cuando* el enlace Guile con GnuTLS está disponible en el entorno de la usuaria; cuando no está disponible se emite un error. See Section "Guile Preparations" in *GnuTLS-Guile*, para más información.

`guix download` verifica los certificados del servidor HTTPS cargando las autoridades X.509 del directorio al que apunta la variable de entorno `SSL_CERT_DIR` (see Section 11.12 [Certificados X.509], page 621), a menos que se use `--no-check-certificate`.

Alternatively, `guix download` can also retrieve a Git repository, possibly a specific commit, tag, or branch.

Las siguientes opciones están disponibles:

`--hash=`*`algoritmo`*
`-H` *`algoritmo`*
: Calcula el resultado del hash usando el *algoritmo* proporcionado. See Section 9.4 [Invocación de guix hash], page 197, para más información.

`--format=`*fmt*
`-f` *fmt* Escribe el hash en el formato especificado por *fmt*. Para más información sobre los valores aceptados en *fmt*, see Section 9.4 [Invocación de guix hash], page 197.

`--no-check-certificate`
No valida los certificados X.509 de los servidores HTTPS.

Cuando se usa esta opción, no tiene *absolutamente ninguna garantía* de que está comunicando con el servidor responsable de la URL auténtico, lo que le hace vulnerables a ataques de interceptación ("man-in-the-middle").

`--output=`*archivo*
`-o` *archivo*
Almacena el archivo descargado en *archivo* en vez de añadirlo al almacén.

`--git`
`-g` Checkout the Git repository at the latest commit on the default branch.

`--commit=`*commit-or-tag*
Checkout the Git repository at *commit-or-tag*.

*commit-or-tag* can be either a tag or a commit defined in the Git repository.

`--branch=`*rama*
Checkout the Git repository at *branch*.

The repository will be checked out at the latest commit of *branch*, which must be a valid branch of the Git repository.

`--recursive`
`-r` Recursively clone the Git repository.

## 9.4 Invocación de `guix hash`

The `guix hash` command computes the hash of a file. It is primarily a convenience tool for anyone contributing to the distribution: it computes the cryptographic hash of one or more files, which can be used in the definition of a package (see Section 8.2 [Definición de paquetes], page 102).

La sintaxis general es:

```
guix hash option file ...
```

Cuando *archivo* es `-` (un guión), `guix hash` calcula el hash de los datos leídos por la entrada estándar. `guix hash` tiene las siguientes opciones:

`--hash=`*algoritmo*
`-H` *algoritmo*
Calcula un hash usando el *algoritmo* especificado, `sha256` de manera predeterminada.

*algorithm* must be the name of a cryptographic hash algorithm supported by Libgcrypt *via* Guile-Gcrypt—e.g., `sha512` or `sha3-256` (see Section "Hash Functions" in *Guile-Gcrypt Reference Manual*).

`--format=`*fmt*
`-f` *fmt* Escribe el hash en el formato especificado por *fmt*.

Los formatos disponibles son: `base64`, `nix-base32`, `base32`, `base16` (se puede usar también `hex` y `hexadecimal`).

Si no se especifica la opción `--format`, `guix hash` mostrará el hash en `nix-base32`. Esta representación es la usada en las definiciones de paquetes.

`--recursive`
`-r` The `--recursive` option is deprecated in favor of `--serializer=nar` (see below); `-r` remains accepted as a convenient shorthand.

`--serializer=`*type*
`-S` *type* Compute the hash on *file* using *type* serialization.

*type* may be one of the following:

`none` This is the default: it computes the hash of a file's contents.

`nar` Compute the hash of a "normalized archive" (or "nar") containing *file*, including its children if it is a directory. Some of the metadata of *file* is part of the archive; for instance, when *file* is a regular file, the hash is different depending on whether *file* is executable or not. Metadata such as time stamps have no impact on the hash (see Section 5.11 [Invocación de guix archive], page 66, for more info on the nar format).

`git` Compute the hash of the file or directory as a Git "tree", following the same method as the Git version control system.

`--exclude-vcs`
`-x` Cuando se combina con `--recursive`, excluye los directorios del sistema de control de versiones (`.bzr`, `.git`, `.hg`, etc.).

Como un ejemplo, así es como calcularía el hash de una copia de trabajo Git, lo cual es útil cuando se usa el método `git-fetch` (see Section 8.2.2 [Referencia de origin], page 110):

```
$ git clone http://example.org/foo.git
$ cd foo
$ guix hash -x --serializer=nar .
```

## 9.5 Invocación de `guix import`

La orden `guix import` es útil para quienes desean añadir un paquete a la distribución con el menor trabajo posible—una demanda legítima. La orden conoce algunos repositorios de los que puede "importar" metadatos de paquetes. El resultado es una definición de paquete, o una plantilla de ella, en el formato que conocemos (see Section 8.2 [Definición de paquetes], page 102).

La sintaxis general es:

```
guix import [global-options...] importer package [options...]
```

*importer* specifies the source from which to import package metadata, and *options* specifies a package identifier and other options specific to *importer*. `guix import` itself has the following *global-options*:

`--insert=`*file*
`-i` *file* Insert the package definition(s) that the *importer* generated into the specified *file*, either in alphabetical order among existing package definitions, or at the end of the file otherwise.

Algunos de los importadores dependen de poder ejecutar la orden `gpgv`. Para ello, GnuPG debe estar instalado y en `$PATH`; ejecute `guix install gnupg` si es necesario.

Actualmente los "importadores" disponibles son:

`gnu` Importa los metadatos del paquete GNU seleccionado. Proporciona una plantilla para la última versión de dicho paquete GNU, incluyendo el hash de su archivador tar de fuentes, y su sinopsis y descripción canónica.

Información adicional como las dependencias del paquete y su licencia deben ser deducidas manualmente.

Por ejemplo, la siguiente orden devuelve una definición de paquete para GNU Hello.

```
guix import gnu hello
```

Las opciones específicas de línea de ordenes son:

`--key-download=`*política*
Como en `guix refresh`, especifica la política de tratamiento de las claves OpenPGP no encontradas cuando se verifica la firma del paquete. See Section 9.6 [Invocación de guix refresh], page 207.

`pypi` Importa metadatos desde el índice de paquetes Python (PyPI) (`https://pypi.python.org/`). La información se toma de la descripción con formato JSON disponible en `pypi.python.org` y habitualmente incluye toda la información relevante, incluyendo las dependencias del paquete. Para una máxima eficiencia, se recomienda la instalación de la utilidad `unzip`, de manera que el importador pueda extraer los archivos wheel de Python y obtener datos de ellos.

The command below imports metadata for the latest version of the `itsdangerous` Python package:

```
guix import pypi itsdangerous
```

You can also ask for a specific version:

```
guix import pypi itsdangerous@1.1.0
```

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`gem` Importa metadatos desde RubyGems (`https://rubygems.org/`). La información se extrae de la descripción en formato JSON disponible en `rubygems.org` e incluye la información más relevante, incluyendo las dependencias en tiempo de ejecución. Hay algunos puntos a tener en cuenta, no obstante. Los metadatos no distinguen entre sinopsis y descripción, por lo que se usa la misma cadena para ambos campos. Adicionalmente, los detalles de las dependencias no-Ruby necesarias para construir extensiones nativas no está disponible y se deja como ejercicio a la empaquetadora.

La siguiente orden importa los meta-datos para el paquete de Ruby `rails`:

```
guix import gem rails
```

You can also ask for a specific version:

```
guix import gem rails@7.0.4
```

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`minetest` Import metadata from ContentDB (`https://content.minetest.net`). Information is taken from the JSON-formatted metadata provided through ContentDB's API (`https://content.minetest.net/help/api/`) and includes most relevant information, including dependencies. There are some caveats, however. The license information is often incomplete. The commit hash is sometimes missing. The descriptions are in the Markdown format, but Guix uses Texinfo instead. Texture packs and subgames are unsupported.

The command below imports metadata for the Mesecons mod by Jeija:

```
guix import minetest Jeija/mesecons
```

The author name can also be left out:

```
guix import minetest mesecons
```

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`cpan` Importa metadatos desde MetaCPAN (`https://www.metacpan.org/`). La información se extrae de la descripción en formato JSON disponible a través del API de MetaCPAN (`https://fastapi.metacpan.org/`) e incluye la información más relevante, como las dependencias de otros módulos. La información de la licencia debe ser comprobada atentamente. Si Perl está disponible en el almacén, se usará la utilidad `corelist` para borrar los módulos básicos de la lista de dependencias.

La siguiente orden importa los metadatos del módulo Perl Acme::Boolean:

```
guix import cpan Acme::Boolean
```

`cran` Importa metadatos desde CRAN (`https://cran.r-project.org/`), el repositorio central para el entorno estadístico y gráfico GNU R (`https://r-project.org`).

La información se extrae del archivo `DESCRIPTION` del paquete.

La siguiente orden importa los metadatos del paquete de R Cairo:

```
guix import cran Cairo
```

You can also ask for a specific version:

```
guix import cran rasterVis@0.50.3
```

Cuando se añade `--recursive`, el importador recorrerá el grafo de dependencias del paquete original proporcionado recursivamente y generará expresiones de paquetes para todos aquellos que no estén todavía en Guix.

When `--style=specification` is added, the importer will generate package definitions whose inputs are package specifications instead of references to package variables. This is useful when generated package definitions are to be appended to existing user modules, as the list of used package modules need not be changed. The default is `--style=variable`.

When `--prefix=license:` is added, the importer will prefix license atoms with `license:`, allowing a prefixed import of `(guix licenses)`.

Cuando se usa `--archive=bioconductor`, los metadatos se importan de Bioconductor (`https://www.bioconductor.org`), un repositorio de paquetes R para el análisis y comprensión de datos genéticos de alto caudal en bioinformática.

La información se extrae del archivo `DESCRIPTION` contenido en el archivo del paquete.

La siguiente orden importa los metadatos del paquete de R GenomicRanges:

```
guix import cran --archive=bioconductor GenomicRanges
```

Por último, también puede importar paquetes de R que no se hayan publicado todavía en CRAN o en Bioconductor siempre que estén en un repositorio git. Use `--archive=git` seguido de la URL del repositorio git:

```
guix import cran --archive=git https://github.com/immunogenomics/harmony
```

`texlive` Import TeX package information from the TeX Live package database for TeX packages that are part of the TeX Live distribution (`https://www.tug.org/texlive/`).

Information about the package is obtained from the TeX Live package database, a plain text file that is included in the `texlive-scripts` package. The source code is downloaded from possibly multiple locations in the SVN repository of the Tex Live project. Note that therefore SVN must be installed and in `$PATH`; run `guix install subversion` if needed.

La siguiente orden importa los metadatos del paquete de TeX `fontspec`:

```
guix import texlive fontspec
```

La opciones adicionales incluyen:

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`json` Importa metadatos de paquetes desde un archivo JSON local. Considere el siguiente ejemplo de definición de paquete en formato JSON:

```
{
  "name": "hello",
  "version": "2.10",
  "source": "mirror://gnu/hello/hello-2.10.tar.gz",
```

```
  "build-system": "gnu",
  "home-page": "https://www.gnu.org/software/hello/",
  "synopsis": "Hello, GNU world: An example GNU package",
  "description": "GNU Hello prints a greeting.",
  "license": "GPL-3.0+",
  "native-inputs": ["gettext"]
}
```

Los nombres de los campos son los mismos que para el registro `<package>` (See Section 8.2 [Definición de paquetes], page 102). Las referencias a otros paquetes se proporcionan como listas JSON de cadenas de especificación de paquete entrecomilladas como `guile` o `guile@2.0`.

El importador también permite una definición de fuentes más explícita usando los campos comunes de los registros `<origin>`:

```
{
  ...
  "source": {
    "method": "url-fetch",
    "uri": "mirror://gnu/hello/hello-2.10.tar.gz",
    "sha256": {
      "base32": "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i"
    }
  }
  ...
}
```

La siguiente orden importa los metadatos desde el archivo JSON `hello.json` y devuelve una expresión de "package":

```
guix import json hello.json
```

`hackage` Importa metadatos desde el archivo central de paquetes de la comunidad Haskell Hackage (`https://hackage.haskell.org/`). La información se obtiene de archivos Cabal e incluye toda la información relevante, incluyendo las dependencias del paquete.

Las opciones específicas de línea de ordenes son:

`--stdin`
`-s` Lee un archivo Cabal por la entrada estándar.

`--no-test-dependencies`
`-t` No incluye las dependencias necesarias únicamente para las baterías de pruebas.

`--cabal-environment=`*alist*
`-e` *alist* *alist* es una lista asociativa Scheme que define el entorno en el que los condicionales Cabal se evalúan. Los valores aceptados son: `os`, `arch`, `impl` y una cadena que representa el nombre de la condición. El valor asociado a la condición tiene que ser o bien el símbolo `true` o bien `false`. Los valores predeterminados asociados a las claves `os`, `arch` y `impl` son '`linux`', '`x86_64`' y '`ghc`', respectivamente.

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

La siguiente orden importa los metadatos de la última versión del paquete Haskell HTTP sin incluir las dependencias de las pruebas y especificando la opción '`network-uri`' con valor `false`:

```
guix import hackage -t -e "'((\"network-uri\" . false))" HTTP
```

Se puede especificar opcionalmente una versión específica del paquete añadiendo al nombre del paquete una arroba y el número de versión como en el siguiente ejemplo:

```
guix import hackage mtl@2.1.3.1
```

`stackage` El importador `stackage` es un recubrimiento sobre el de `hackage`. Toma un nombre de paquete, busca la versión de paquete incluida en una publicación de la versión de mantenimiento extendido (LTS) Stackage (`https://www.stackage.org`) y usa el importador `hackage` para obtener sus metadatos. Fíjese que es su decisión seleccionar una publicación LTS compatible con el compilador GHC usado en Guix.

Las opciones específicas de línea de ordenes son:

`--no-test-dependencies`
`-t` No incluye las dependencias necesarias únicamente para las baterías de pruebas.

`--lts-version=`*versión*
`-l` *versión*
*versión* es la versión LTS de publicación deseada. Si se omite se usa la última publicación.

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

La siguiente orden importa los metadatos del paquete Haskell HTTP incluido en la versión de publicación LTS de Stackage 7.18:

```
guix import stackage --lts-version=7.18 HTTP
```

`elpa` Importa metadatos desde el repositorio de archivos de paquetes Emacs Lisp (ELPA) (see Section "Packages" in *The GNU Emacs Manual*).

Las opciones específicas de línea de ordenes son:

`--archive=`*repo*
`-a` *repo* *repo* identifica el repositorio de archivos del que obtener la información. Actualmente los repositorios disponibles y sus identificadores son:

- GNU (`https://elpa.gnu.org/packages`), seleccionado con el identificador `gnu`. Utilizado de manera predeterminada.

Los paquetes de `elpa.gnu.org` están firmados con una de las claves que contiene el anillo de claves GnuPG en `share/emacs/25.1/etc/package-keyring.gpg` (o similar) en el paquete `emacs` (see Section "Package Installation" in *The GNU Emacs Manual*).

- NonGNU (`https://elpa.nongnu.org/nongnu/`), selected by the `nongnu` identifier.
- MELPA-Stable (`https://stable.melpa.org/packages`), seleccionado con el identificador `melpa-stable`.
- MELPA (`https://melpa.org/packages`), seleccionado con el identificador `melpa`.

`--recursive`
`-r`
: Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`crate`
: Importa metadatos desde el repositorio de paquetes Rust crates.io (`https://crates.io`), como en este ejemplo:

```
guix import crate blake2-rfc
```

El importador de crate también le permite especificar una cadena de versión:

```
guix import crate constant-time-eq@0.1.0
```

La opciones adicionales incluyen:

`--recursive`
`-r`
: Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`--recursive-dev-dependencies`
: If `--recursive-dev-dependencies` is specified, also the recursively imported packages contain their development dependencies, which are recursively imported as well.

`--allow-yanked`
: If no non-yanked version of a crate is available, use the latest yanked version instead instead of aborting.

`elm`
: Import metadata from the Elm package repository package.elm-lang.org (`https://package.elm-lang.org`), as in this example:

```
guix import elm elm-explorations/webgl
```

The Elm importer also allows you to specify a version string:

```
guix import elm elm-explorations/webgl@1.1.3
```

La opciones adicionales incluyen:

`--recursive`
`-r`
: Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`npm-binary`

Import metadata from the npm Registry (`https://registry.npmjs.org`), as in this example:

```
guix import npm-binary buffer-crc32
```

The npm-binary importer also allows you to specify a version string:

```
guix import npm-binary buffer-crc32@1.0.0
```

**Nota:** Generated package expressions skip the build step of the `node-build-system`. As such, generated package expressions often refer to transpiled or generated files, instead of being built from source.

La opciones adicionales incluyen:

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`opam` Importa metadatos desde el repositorio de paquetes OPAM (`https://opam.ocaml.org/`) usado por la comunidad OCaml.

La opciones adicionales incluyen:

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`composer` Import metadata from the Composer (`https://getcomposer.org/`) package archive used by the PHP community, as in this example:

```
guix import composer phpunit/phpunit
```

La opciones adicionales incluyen:

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`--repo` By default, packages are searched in the official OPAM repository. This option, which can be used more than once, lets you add other repositories which will be searched for packages. It accepts as valid arguments:

- the name of a known repository - can be one of `opam`, `coq` (equivalent to `coq-released`), `coq-core-dev`, `coq-extra-dev` or `grew`.
- the URL of a repository as expected by the `opam repository add` command (for instance, the URL equivalent of the above `opam` name would be `https://opam.ocaml.org`).
- the path to a local copy of a repository (a directory containing a `packages/` sub-directory).

Repositories are assumed to be passed to this option by order of preference. The additional repositories will not replace the default `opam` repository, which is always kept as a fallback.

Also, please note that versions are not compared across repositories. The first repository (from left to right) that has at least one version of a given package will prevail over any others, and the version imported will be the latest one found *in this repository only*.

`go` Import metadata for a Go module using proxy.golang.org (`https://proxy.golang.org`).

```
guix import go gopkg.in/yaml.v2
```

It is possible to use a package specification with a `@VERSION` suffix to import a specific version.

La opciones adicionales incluyen:

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`--pin-versions`
When using this option, the importer preserves the exact versions of the Go modules dependencies instead of using their latest available versions. This can be useful when attempting to import packages that recursively depend on former versions of themselves to build. When using this mode, the symbol of the package is made by appending the version to its name, so that multiple versions of the same package can coexist.

`egg` Import metadata for CHICKEN eggs (`https://wiki.call-cc.org/eggs`). The information is taken from `PACKAGE.egg` files found in the eggs-5-all (`git://code.call-cc.org/eggs-5-all`) Git repository. However, it does not provide all the information that we need, there is no "description" field, and the licenses used are not always precise (BSD is often used instead of BSD-N).

```
guix import egg sourcehut
```

You can also ask for a specific version:

```
guix import egg arrays@1.0
```

La opciones adicionales incluyen:

`--recursive`
`-r` Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

`hexpm` Import metadata from the hex.pm Erlang and Elixir package repository hex.pm (`https://hex.pm`), as in this example:

```
guix import hexpm stun
```

The importer tries to determine the build system used by the package.

The hexpm importer also allows you to specify a version string:

```
guix import hexpm cf@0.3.0
```

La opciones adicionales incluyen:

`--recursive`
`-r`
: Recorre el grafo de dependencias del paquete original proporcionado recursivamente y genera expresiones de paquete para todos aquellos paquetes que no estén todavía en Guix.

La estructura del código de `guix import` es modular. Sería útil tener más importadores para otros formatos de paquetes, y su ayuda es bienvenida aquí (see Chapter 22 [Contribuir], page 734).

## 9.6 Invocación de `guix refresh`

The primary audience of the `guix refresh` command is packagers. As a user, you may be interested in the `--with-latest` option, which can bring you package update superpowers built upon `guix refresh` (see Section 9.1.2 [Opciones de transformación de paquetes], page 184). By default, `guix refresh` reports any packages provided by the distribution that are outdated compared to the latest upstream version, like this:

```
$ guix refresh
gnu/packages/gettext.scm:29:13: gettext would be upgraded from 0.18.1.1 to 0.18.2.1
gnu/packages/glib.scm:77:12: glib would be upgraded from 2.34.3 to 2.37.0
```

De manera alternativa, se pueden especificar los paquetes a considerar, en cuyo caso se emite un aviso para paquetes que carezcan de actualizador:

```
$ guix refresh coreutils guile guile-ssh
gnu/packages/ssh.scm:205:2: warning: no updater for guile-ssh
gnu/packages/guile.scm:136:12: guile would be upgraded from 2.0.12 to 2.0.13
```

`guix refresh` navega por los repositorios oficiales de cada paquete y determina el número de versión mayor entre las publicaciones encontradas. La orden sabe cómo actualizar tipos específicos de paquetes: paquetes GNU, paquetes ELPA, etc.—vea la documentación de `--type` más adelante. Hay muchos paquetes, no obstante, para los que carece de un método para determinar si está disponible una versión oficial posterior. No obstante, el mecanismo es extensible, ¡no tenga problema en contactar con nosotras para añadir un método nuevo!

`--recursive`
: Considera los paquetes especificados, y todos los paquetes de los que dependen.

```
$ guix refresh --recursive coreutils
gnu/packages/acl.scm:40:13: acl would be upgraded from 2.2.53 to 2.3.1
gnu/packages/m4.scm:30:12: 1.4.18 is already the latest version of m4
gnu/packages/xml.scm:68:2: warning: no updater for expat
gnu/packages/multiprecision.scm:40:12: 6.1.2 is already the latest version o
...
```

If for some reason you don't want to update to the latest version, you can update to a specific version by appending an equal sign and the desired version number to the package specification. Note that not all updaters support this; an error is reported when an updater cannot refresh to the specified version.

```
$ guix refresh trytond-party
```

```
gnu/packages/guile.scm:392:2: guile would be upgraded from 3.0.3 to 3.0.5
$ guix refresh -u guile=3.0.4
...
gnu/packages/guile.scm:392:2: guile: updating from version 3.0.3 to version 3.0.4...
...
$ guix refresh -u guile@2.0=2.0.12
...
gnu/packages/guile.scm:147:2: guile: updating from version 2.0.10 to version 2.0.12...
...
```

In some specific cases, you may have many packages specified via a manifest or a module selection which should all be updated together; for these cases, the `--target-version` option can be provided to have them all refreshed to the same version, as shown in the examples below:

```
$ guix refresh qtbase qtdeclarative --target-version=6.5.2
gnu/packages/qt.scm:1248:13: qtdeclarative would be upgraded from 6.3.2 to 6.5.2
gnu/packages/qt.scm:584:2: qtbase would be upgraded from 6.3.2 to 6.5.2
$ guix refresh --manifest=qt5-manifest.scm --target-version=5.15.10
gnu/packages/qt.scm:1173:13: qtxmlpatterns would be upgraded from 5.15.8 to 5.15.10
gnu/packages/qt.scm:1202:13: qtdeclarative would be upgraded from 5.15.8 to 5.15.10
gnu/packages/qt.scm:1762:13: qtserialbus would be upgraded from 5.15.8 to 5.15.10
gnu/packages/qt.scm:2070:13: qtquickcontrols2 would be upgraded from 5.15.8 to 5.15.10
...
```

A veces el nombre oficial es diferente al nombre de paquete usado en Guix, y `guix refresh` necesita un poco de ayuda. La mayor parte de los actualizadores utilizan la propiedad `upstream-name` en las definiciones de paquetes, que puede usarse para obtener dicho efecto:

```
(define-public network-manager
  (package
    (name "network-manager")
    ;; ...
    (properties '((upstream-name . "NetworkManager")))))
```

When passed `--update`, it modifies distribution source files to update the version numbers and source code hashes of those package definitions, as well as possibly their inputs (see Section 8.2 [Definición de paquetes], page 102). This is achieved by downloading each package's latest source tarball and its associated OpenPGP signature, authenticating the downloaded tarball against its signature using `gpgv`, and finally computing its hash—note that GnuPG must be installed and in `$PATH`; run `guix install gnupg` if needed.

Cuando la clave pública usada para firmar el archivador no se encuentra en el anillo de claves de la usuaria, se intenta automáticamente su obtención desde un servidor de claves públicas; cuando se encuentra, la clave se añade al anillo de claves de la usuaria; en otro caso, `guix refresh` informa de un error.

Se aceptan las siguientes opciones:

`--expression=`*expr*
`-e` *expr* Considera el paquete al que evalúa *expr*

Es útil para hacer una referencia precisa de un paquete concreto, como en este ejemplo:

```
guix refresh -l -e '(@@ (gnu packages commencement) glibc-final)'
```

Esta orden enumera los paquetes que dependen de la libc "final" (esencialmente todos los paquetes).

`--update`
`-u`

Update distribution source files (package definitions) in place. This is usually run from a checkout of the Guix source tree (see Section 22.4 [Ejecución de Guix antes de estar instalado], page 739):

```
./pre-inst-env guix refresh -s non-core -u
```

See Section 8.2 [Definición de paquetes], page 102, for more information on package definitions. You can also run it on packages from a third-party channel:

```
guix refresh -L /path/to/channel -u package
```

See Section 6.7 [Creación de un canal], page 73, on how to create a channel.

This command updates the version and source code hash of the package. Depending on the updater being used, it can also update the various '`inputs`' fields of the package. In some cases, the updater might get inputs wrong—it might not know about an extra input that's necessary, or it might add an input that should be avoided.

To address that, packagers can add properties stating inputs that should be added to those found by the updater or inputs that should be ignored: the `updater-extra-inputs` and `updater-ignored-inputs` properties pertain to "regular" inputs, and there are equivalent properties for '`native`' and '`propagated`' inputs. In the example below, we tell the updater that we need '`openmpi`' as an additional input:

```
(define-public python-mpi4py
  (package
    (name "python-mpi4py")
    ;; ...
    (inputs (list openmpi))
    (properties
     '((updater-extra-inputs . ("openmpi"))))))
```

That way, `guix refresh -u python-mpi4py` will leave the '`openmpi`' input, even if it is not among the inputs it would normally add.

`--select=[`*subconjunto*`]`
`-s` *subconjunto*

Select all the packages in *subset*, one of `core`, `non-core` or `module:`*name*.

El subconjunto `core` hace referencia a todos los paquetes en el núcleo de la distribución—es decir, paquetes que se usan para construir "todo lo demás". Esto incluye GCC, libc, Binutils, Bash, etc. Habitualmente, cambiar uno de esos paquetes en la distribución conlleva la reconstrucción de todos los demás. Por tanto, esas actualizaciones son una inconveniencia para las usuarias en términos de tiempo de construcción o ancho de banda usado por la actualización.

El subconjunto `non-core` hace referencia a los paquetes restantes. Es típicamente útil en casos donde una actualización de paquetes básicos no sería conveniente.

The `module:`*`name`* subset refers to all the packages in a specified guile module. The module can be specified as `module:guile` or `module:(gnu packages guile)`, the former is a shorthand for the later.

`--manifest=`*archivo*
`-m` *archivo*
: Select all the packages from the manifest in *file*. This is useful to check if any packages of the user manifest can be updated.

`--type=`*actualizador*
`-t` *actualizador*
: Selecciona únicamente paquetes manejados por *actualizador* (puede ser una lista separada por comas de actualizadores). Actualmente, *actualizador* puede ser:

  `gnu` el actualizador de paquetes GNU;

  `savannah` el actualizador para paquetes alojados en Savannah (`https://savannah.gnu.org`);

  `sourceforge` the updater for packages hosted at SourceForge (`https://sourceforge.net`);

  `gnome` el actualizador para paquetes GNOME;

  `kde` el actualizador para paquetes KDE;

  `xorg` el actualizador para paquetes X.org;

  `kernel.org` el actualizador para paquetes alojados en kernel.org;

  `egg` the updater for Egg (`https://wiki.call-cc.org/eggs/`) packages;

  `elpa` el actualizador para paquetes ELPA (`https://elpa.gnu.org/`);

  `cran` el actualizador para paquetes CRAN (`https://cran.r-project.org/`);

  `bioconductor` el actualizador para paquetes R Bioconductor (`https://www.bioconductor.org/`);

  `cpan` el actualizador para paquetes CPAN (`https://www.cpan.org/`);

  `pypi` el actualizador para paquetes PyPI (`https://pypi.python.org`).

  `gem` el actualizador para paquetes RubyGems (`https://rubygems.org`).

  `github` el actualizador para paquetes GitHub (`https://github.com`).

`hackage` el actualizador para paquetes Hackage (`https://hackage.haskell.org`).

`stackage` el actualizador para paquetes Stackage (`https://www.stackage.org`).

`crate` el actualizador para paquetes Crates (`https://crates.io`).

`launchpad`
el actualizador para paquetes Launchpad (`https://launchpad.net`).

`generic-html`
a generic updater that crawls the HTML page where the source tarball of the package is hosted, when applicable, or the HTML page specified by the `release-monitoring-url` property of the package.

`generic-git`
a generic updater for packages hosted on Git repositories. It tries to be smart about parsing Git tag names, but if it is not able to parse the tag name and compare tags correctly, users can define the following properties for a package.

- `release-tag-prefix`: a regular expression for matching a prefix of the tag name.
- `release-tag-suffix`: a regular expression for matching a suffix of the tag name.
- `release-tag-version-delimiter`: a string used as the delimiter in the tag name for separating the numbers of the version.
- `accept-pre-releases`: by default, the updater will ignore pre-releases; to make it also look for pre-releases, set the this property to `#t`.

```
(package
  (name "foo")
  ;; ...
  (properties
    '((release-tag-prefix . "^release0-")
      (release-tag-suffix . "[a-z]?$")
      (release-tag-version-delimiter . ":"))))
```

Por ejemplo, la siguiente orden únicamente comprueba actualizaciones de paquetes Emacs alojados en `elpa.gnu.org` y actualizaciones de paquetes CRAN:

```
$ guix refresh --type=elpa,cran
gnu/packages/statistics.scm:819:13: r-testthat would be upgraded from 0.10.0
gnu/packages/emacs.scm:856:13: emacs-auctex would be upgraded from 11.88.6 t
```

`--list-updaters`
Enumera los actualizadores disponibles y finaliza (vea la opción previa `--type`).

Para cada actualizador, muestra la fracción de paquetes que cubre; al final muestra la fracción de paquetes cubiertos por todos estos actualizadores.

Además, `guix refresh` puede recibir uno o más nombres de paquetes, como en este ejemplo:

```
$ ./pre-inst-env guix refresh -u emacs idutils gcc@4.8
```

The command above specifically updates the `emacs` and `idutils` packages. The `--select` option would have no effect in this case. You might also want to update definitions that correspond to the packages installed in your profile:

```
$ ./pre-inst-env guix refresh -u \
       $(guix package --list-installed | cut -f1)
```

Cuando se considera la actualización de un paquete, a veces es conveniente conocer cuantos paquetes se verían afectados por la actualización y su compatibilidad debería comprobarse. Para ello la siguiente opción puede usarse cuando se proporcionan uno o más nombres de paquete a `guix refresh`:

`--list-dependent`
`-l`
: Enumera los paquetes de nivel superior dependientes que necesitarían una reconstrucción como resultado de la actualización de uno o más paquetes.

  See Section 9.10 [Invocación de guix graph], page 221, para información sobre cómo visualizar la lista de paquetes que dependen de un paquete.

Sea consciente de que la opción `--list-dependent` únicamente *aproxima* las reconstrucciones necesarias como resultado de una actualización. Más reconstrucciones pueden ser necesarias bajo algunas circunstancias.

```
$ guix refresh --list-dependent flex
Building the following 120 packages would ensure 213 dependent packages are rebuilt:
hop@2.4.0 emacs-geiser@0.13 notmuch@0.18 mu@0.9.9.5 cflow@1.4 idutils@4.6 ...
```

La orden previa enumera un conjunto de paquetes que puede ser construido para comprobar la compatibilidad con una versión actualizada del paquete `flex`.

`--list-transitive`
`-T`
: Enumera todos los paquetes de los que uno o más paquetes dependen.

```
$ guix refresh --list-transitive flex
flex@2.6.4 depends on the following 25 packages: perl@5.28.0 help2man@1.47.6
bison@3.0.5 indent@2.2.10 tar@1.30 gzip@1.9 bzip2@1.0.6 xz@5.2.4 file@5.33 .
```

La orden previa enumera un conjunto de paquetes que, en caso de cambiar, causarían la reconstrucción de `flex`.

Las siguientes opciones pueden usarse para personalizar la operación de GnuPG:

`--gpg=`*`orden`*
: Use *orden* como la orden de GnuPG 2.x. Se busca *orden* en `PATH`.

`--keyring=`*`archivo`*
: Usa *archivo* como el anillo de claves para claves de proveedoras. *archivo* debe estar en el *formato keybox*. Los archivos Keybox normalmente tienen un nombre terminado en `.kbx` y GNU Privacy Guard (GPG) puede manipular estos archivos (see Section "kbxutil" in *Using the GNU Privacy Guard*, para información sobre una herramienta para manipular archivos keybox).

Cuando se omite esta opción, `guix refresh` usa `~/.config/guix/upstream/trustedkeys.kbx` como el anillo de claves para las firmas de proveedoras. Las firmas OpenPGP son comprobadas contra claves de este anillo; las claves que falten son descargadas a este anillo de claves también (véase `--key-download` a continuación).

Puede exportar claves de su anillo de claves GPG predeterminado en un archivo keybox usando órdenes como esta:

```
gpg --export rms@gnu.org | kbxutil --import-openpgp >> mianillo.kbx
```

Del mismo modo, puede obtener claves de un archivo keybox específico así:

```
gpg --no-default-keyring --keyring mianillo.kbx \
  --recv-keys 3CE464558A84FDC69DB40CFB090B11993D9AEBB5
```

See Section "GPG Configuration Options" in *Using the GNU Privacy Guard*, for more information on GPG's `--keyring` option.

`--key-download=`*política*

Maneja las claves no encontradas de acuerdo a la *política*, que puede ser una de:

`always` Siempre descarga las claves OpenPGP no encontradas del servidor de claves, y las añade al anillo de claves GnuPG de la usuaria.

`never` Nunca intenta descargar claves OpenPGP no encontradas. Simplemente propaga el error.

`interactive`
Cuando se encuentra un paquete firmado por una clave OpenPGP desconocida, pregunta a la usuaria si descargarla o no. Este es el comportamiento predeterminado.

`--key-server=`*dirección*

Use *dirección* como el servidor de claves OpenPGP cuando se importa una clave pública.

`--load-path=`*directorio*
`-L` *directorio*

Añade *directorio* al frente de la ruta de búsqueda de módulos de paquetes (see Section 8.1 [Módulos de paquetes], page 101).

Esto permite a las usuarias definir sus propios paquetes y hacerlos visibles a las herramientas de línea de órdenes.

El actualizador `github` usa la API de GitHub (`https://developer.github.com/v3/`) para consultar nuevas publicaciones. Cuando se usa repetidamente, por ejemplo al comprobar todos los paquetes, GitHub terminará rechazando las peticiones siguientes a través de su API. Por defecto se permiten 60 peticiones por hora a través de su API, y una actualización completa de todos los paquetes de GitHub en Guix necesita más que eso. La identificación con GitHub a través del uso de un identificador de su API ("token") amplia esos límites. Para usar dicho identificador, establezca la variable de entorno `GUIX_GITHUB_TOKEN` al valor obtenido a través de `https://github.com/settings/tokens` o de otra manera.

## 9.7 Invoking `guix style`

The `guix style` command helps users and packagers alike style their package definitions and configuration files according to the latest fashionable trends. It can either reformat whole files, with the `--whole-file` option, or apply specific *styling rules* to individual package definitions. The command currently provides the following styling rules:

- formatting package definitions according to the project's conventions (see Section 22.9.4 [Formato del código], page 759);
- rewriting package inputs to the "new style", as explained below.

The way package inputs are written is going through a transition (see Section 8.2.1 [Referencia de package], page 105, for more on package inputs). Until version 1.3.0, package inputs were written using the "old style", where each input was given an explicit label, most of the time the package name:

```
(package
  ;; ...
  ;; The "old style" (deprecated).
  (inputs `(("libunistring" ,libunistring)
            ("libffi" ,libffi))))
```

Today, the old style is deprecated and the preferred style looks like this:

```
(package
  ;; ...
  ;; The "new style".
  (inputs (list libunistring libffi)))
```

Likewise, uses of `alist-delete` and friends to manipulate inputs is now deprecated in favor of `modify-inputs` (see Section 8.3 [Definición de variantes de paquetes], page 114, for more info on `modify-inputs`).

In the vast majority of cases, this is a purely mechanical change on the surface syntax that does not even incur a package rebuild. Running `guix style -S inputs` can do that for you, whether you're working on packages in Guix proper or in an external channel.

La sintaxis general es:

```
guix style [options] package...
```

This causes `guix style` to analyze and rewrite the definition of *package*... or, when *package* is omitted, of *all* the packages. The `--styling` or `-S` option allows you to select the style rule, the default rule being `format`—see below.

To reformat entire source files, the syntax is:

```
guix style --whole-file file...
```

Las opciones disponibles se enumeran a continuación.

`--dry-run`
`-n` Show source file locations that would be edited but do not modify them.

`--whole-file`
`-f` Reformat the given files in their entirety. In that case, subsequent arguments are interpreted as file names (rather than package names), and the `--styling` option has no effect.

As an example, here is how you might reformat your operating system configuration (you need write permissions for the file):

```
guix style -f /etc/config.scm
```

`--styling=`*rule*
`-S` *rule* Apply *rule*, one of the following styling rules:

`format` Format the given package definition(s)—this is the default styling rule. For example, a packager running Guix on a checkout (see Section 22.4 [Ejecución de Guix antes de estar instalado], page 739) might want to reformat the definition of the Coreutils package like so:

```
./pre-inst-env guix style coreutils
```

`inputs` Rewrite package inputs to the "new style", as described above. This is how you would rewrite inputs of package `whatnot` in your own channel:

```
guix style -L ~/my/channel -S inputs whatnot
```

Rewriting is done in a conservative way: preserving comments and bailing out if it cannot make sense of the code that appears in an inputs field. The `--input-simplification` option described below provides fine-grain control over when inputs should be simplified.

`arguments`

Rewrite package arguments to use G-expressions (see Section 8.12 [Expresiones-G], page 167). For example, consider this package definition:

```
(define-public my-package
  (package
    ;; ...
    (arguments      ;old-style quoted arguments
     '(#:make-flags '("V=1")
       #:phases (modify-phases %standard-phases
                  (delete 'build))))))
```

Running `guix style -S arguments` on this package would rewrite its `arguments` field like to:

```
(define-public my-package
  (package
    ;; ...
    (arguments
      (list #:make-flags #~'("V=1")
            #:phases #~(modify-phases %standard-phases
                         (delete 'build))))))
```

Note that changes made by the `arguments` rule do not entail a rebuild of the affected packages. Furthermore, if a package definition happens to be using G-expressions already, `guix style` leaves it unchanged.

`--list-stylings`
`-l` List and describe the available styling rules and exit.

`--load-path=`*directorio*
`-L` *directorio*
Añade *directorio* al frente de la ruta de búsqueda de módulos de paquetes (see Section 8.1 [Módulos de paquetes], page 101).

`--expression=`*expr*
`-e` *expr* Style the package *expr* evaluates to.

Por ejemplo, ejecutando:

```
guix style -e '(@ (gnu packages gcc) gcc-5)'
```

styles the `gcc-5` package definition.

`--input-simplification=`*policy*
When using the `inputs` styling rule, with '`-S inputs`', this option specifies the package input simplification policy for cases where an input label does not match the corresponding package name. *policy* may be one of the following:

`silent` Simplify inputs only when the change is "silent", meaning that the package does not need to be rebuilt (its derivation is unchanged).

`safe` Simplify inputs only when that is "safe" to do: the package might need to be rebuilt, but the change is known to have no observable effect.

`always` Simplify inputs even when input labels do not match package names, and even if that might have an observable effect.

The default is `silent`, meaning that input simplifications do not trigger any package rebuild.

## 9.8 Invocación de `guix lint`

La orden `guix lint` sirve para ayudar a las desarrolladoras de paquetes a evitar errores comunes y usar un estilo consistente. Ejecuta un número de comprobaciones en un conjunto de paquetes proporcionado para encontrar errores comunes en sus definiciones. Las *comprobaciones* disponibles incluyen (véase `--list-checkers` para una lista completa):

`synopsis`
`description`
Valida ciertas reglas tipográficas y de estilo en la descripción y sinopsis de cada paquete.

`inputs-should-be-native`
Identifica entradas que probablemente deberían ser entradas nativas.

`source`
`home-page`
`mirror-url`
`github-url`
`source-file-name`
Probe `home-page` and `source` URLs and report those that are invalid. Suggest a `mirror://` URL when applicable. If the `source` URL redirects to a GitHub

URL, recommend usage of the GitHub URL. Check that the source file name is meaningful, e.g. is not just a version number or "git-checkout", without a declared `file-name` (see Section 8.2.2 [Referencia de origin], page 110).

`source-unstable-tarball`
: Analiza la URL `source` para determinar si un archivador tar de GitHub se genera de forma automática o es una publicación oficial. Desafortunadamente los archivadores tar de GitHub a veces se regeneran.

`derivation`
: Comprueba que la derivación de los paquetes proporcionados pueden ser calculadas de manera satisfactoria en todos los sistemas implementados (see Section 8.10 [Derivaciones], page 159).

`profile-collisions`
: Comprueba si la instalación de los paquetes proporcionados en el perfil provocaría colisiones. Las colisiones se producen cuando se propagan varios paquetes con el mismo nombre pero una versión diferente o un nombre de archivo del almacén. See Section 8.2.1 [Referencia de package], page 105, para más información sobre entradas propagadas.

`archival` Comprueba si el código fuente del paquete se encuentra archivado en Software Heritage (`https://www.softwareheritage.org`).

Cuando el código fuente que no se encuentra archivado proviene de un sistema de control de versiones[1]—por ejemplo, se ha obtenido con `git-fetch`—, envía a Software Heritage una petición de almacenamiento de manera que se archive cuando sea posible. Esto asegura que las fuentes permanecen disponibles a largo plazo, y que Guix puede usar Software Heritage como respaldo en caso de que el código fuente desapareciese de la máquina que lo almacenaba originalmente. El estado de las peticiones de almacenamiento recientes puede verse en su página web (`https://archive.softwareheritage.org/save/#requests`).

Cuando el código fuente es un archivo comprimido que se obtiene con `url-fetch`, simplemente imprime un mensaje cuando no se encuentra archivado. En el momento de la escritura de este documento, Software Heritage no permite el almacenamiento de archivos comprimidos arbitrarios; estamos trabajando en formas de asegurar que también se archive el código que no se encuentra bajo control de versiones.

Software Heritage limita la tasa de peticiones por dirección IP (`https://archive.softwareheritage.org/api/#rate-limiting`). Cuando se alcanza dicho límite, `guix lint` imprime un mensaje y la comprobación `archival` no hace nada hasta que dicho límite se reinicie.

`cve` Informa de vulnerabilidades encontradas en las bases de datos de vulnerabilidades y exposiciones comunes (CVE) del año actual y el pasado publicadas por el NIST de EEUU (`https://nvd.nist.gov/vuln/data-feeds`).

Para ver información acerca de una vulnerabilidad particular, visite páginas como:

- 'https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-YYYY-ABCD'

[1] VCS en inglés

- 'https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-YYYY-ABCD'

donde CVE-YYYY-ABCD es el identificador CVE—por ejemplo, CVE-2015-7554.

Las desarrolladoras de paquetes pueden especificar en las recetas del paquete el nombre y versión en la plataforma común de enumeración (CPE) (https://nvd.nist.gov/cpe.cfm) del paquete cuando el nombre o versión que usa Guix son diferentes, como en este ejemplo:

```
(package
  (name "grub")
  ;; ...
  ;; CPE llama a este paquete "grub2".
  (properties '((cpe-name . "grub2")
                (cpe-version . "2.3"))))
```

Algunas entradas en la base de datos CVE no especifican a qué versión del paquete hacen referencia, y por lo tanto "permanecen visibles" para siempre. Las desarrolladoras de paquetes que encuentren alertas CVE y verifiquen que pueden ignorarse, pueden declararlas como en este ejemplo:

```
(package
  (name "t1lib")
  ;; ...
  ;; Estas alertas de CVE no aplican y pueden ignorarse
  ;; con seguridad.
  (properties `((lint-hidden-cve . ("CVE-2011-0433"
                                    "CVE-2011-1553"
                                    "CVE-2011-1554"
                                    "CVE-2011-5244")))))
```

`formatting`
: Avisa de problemas de formato obvios en el código fuente: espacios en blanco al final de las líneas, uso de tabuladores, etc.

`input-labels`
: Report old-style input labels that do not match the name of the corresponding package. This aims to help migrate from the "old input style". See Section 8.2.1 [Referencia de package], page 105, for more information on package inputs and input styles. See Section 9.7 [Invoking guix style], page 214, on how to migrate to the new style.

La sintaxis general es:

```
guix lint opciones paquete...
```

Si no se proporciona ningún paquete en la linea de órdenes, todos los paquetes se comprueban. Las *opciones* pueden ser cero o más de las siguientes:

`--list-checkers`
`-l`
: Enumera y describe todas las comprobaciones disponibles que se ejecutarán sobre los paquetes y finaliza.

`--checkers`
`-c`
: Únicamente activa las comprobaciones especificadas en una lista separada por comas que use los nombres devueltos por `--list-checkers`.

`--exclude`
`-x` Únicamente desactiva las comprobaciones especificadas en una lista separada por comas que use los nombres devueltos por `--list-checkers`.

`--expression=`*expr*
`-e` *expr* Considera el paquete al que evalúa *expr*

This is useful to unambiguously designate packages, as in this example:

```
guix lint -c archival -e '(@ (gnu packages guile) guile-3.0)'
```

`--no-network`
`-n` Activa únicamente las comprobaciones que no dependen del acceso a internet.

`--load-path=`*directorio*
`-L` *directorio*
Añade *directorio* al frente de la ruta de búsqueda de módulos de paquetes (see Section 8.1 [Módulos de paquetes], page 101).

Esto permite a las usuarias definir sus propios paquetes y hacerlos visibles a las herramientas de línea de órdenes.

## 9.9 Invocación de `guix size`

La orden `guix size` ayuda a las desarrolladoras de paquetes a perfilar el uso de disco de los paquetes. Es fácil pasar por encima el impacto que produce añadir una dependencia adicional a un paquete, o el impacto del uso de una salida única para un paquete que puede ser dividido fácilmente (see Section 5.4 [Paquetes con múltiples salidas], page 51). Estos son los problemas típicos que `guix size` puede resaltar.

Se le pueden proporcionar una o más especificaciones de paquete como `gcc@4.8` o `guile:debug`, o un nombre de archivo en el almacén. Considere este ejemplo:

```
$ guix size coreutils
store item                                total    self
/gnu/store/...-gcc-5.5.0-lib            60.4    30.1  38.1%
/gnu/store/...-glibc-2.27               30.3    28.8  36.6%
/gnu/store/...-coreutils-8.28           78.9    15.0  19.0%
/gnu/store/...-gmp-6.1.2                63.1     2.7   3.4%
/gnu/store/...-bash-static-4.4.12        1.5     1.5   1.9%
/gnu/store/...-acl-2.2.52               61.1     0.4   0.5%
/gnu/store/...-attr-2.4.47              60.6     0.2   0.3%
/gnu/store/...-libcap-2.25              60.5     0.2   0.2%
total: 78.9 MiB
```

Los elementos del almacén enumerados aquí constituyen la *clausura transitiva* de Coreutils—es decir, Coreutils y todas sus dependencias, recursivamente—como sería devuelto por:

```
$ guix gc -R /gnu/store/...-coreutils-8.23
```

Aquí la salida muestra tres columnas junto a los elementos del almacén. La primera columna, etiquetada “total”, muestra el tamaño en mebibytes (MiB) de la clausura del elemento del almacén—es decir, su propio tamaño sumado al tamaño de todas sus dependencias. La siguiente columna, etiquetada “self”, muestra el tamaño del elemento en sí.

La última columna muestra la relación entre el tamaño del elemento en sí frente al espacio ocupado por todos los elementos enumerados.

En este ejemplo, vemos que la clausura de Coreutils ocupa 79 MiB, cuya mayor parte son libc y las bibliotecas auxiliares de GCC para tiempo de ejecución. (Que libc y las bibliotecas de GCC representen una fracción grande de la clausura no es un problema en sí, puesto que siempre están disponibles en el sistema de todas maneras).

Puesto que la orden también acepta nombres de archivo del almacén, comprobar el tamaño del resultado de una construcción es una operación directa:

```
guix size $(guix system build config.scm)
```

Cuando los paquetes pasados a `guix size` están disponibles en el almacén[2] consultando al daemon para determinar sus dependencias, y mide su tamaño en el almacén, de forma similar a `du -ms --apparent-size` (see Section "du invocation" in *GNU Coreutils*).

Cuando los paquetes proporcionados *no* están en el almacén, `guix size` informa en base de las sustituciones disponibles (see Section 5.3 [Sustituciones], page 46). Esto hace posible perfilar el espacio en disco incluso de elementos del almacén que no están en el disco, únicamente disponibles de forma remota.

Puede especificar también varios nombres de paquetes:

```
$ guix size coreutils grep sed bash
store item                                 total    self
/gnu/store/...-coreutils-8.24          77.8    13.8  13.4%
/gnu/store/...-grep-2.22               73.1     0.8   0.8%
/gnu/store/...-bash-4.3.42             72.3     4.7   4.6%
/gnu/store/...-readline-6.3            67.6     1.2   1.2%
...
total: 102.3 MiB
```

En este ejemplo vemos que la combinación de los cuatro paquetes toma 102.3 MiB en total, lo cual es mucho menos que la suma de cada clausura, ya que tienen muchas dependencias en común.

Cuando tenga delante el perfil devuelto por `guix size` puede preguntarse cuál es la razón de que cierto paquete aparezca en el perfil. Para entenderlo puede usar `guix graph --path -t references` para mostrar la ruta más corta entre dos paquetes (see Section 9.10 [Invocación de guix graph], page 221).

Las opciones disponibles son:

`--substitute-urls=`*urls*
: Usa la información de sustituciones de *urls*. See [client-substitute-urls], page 182.

`--sort=`*clave*
: Ordena las líneas de acuerdo a *clave*, una de las siguientes opciones:

  `propio` el tamaño de cada elemento (predeterminada);

  `closure` el tamaño total de la clausura del elemento.

[2] Más precisamente, `guix size` busca la variante *sin injertos* de los paquetes, como el devuelto por `guix build` *paquete* `--no-grafts`. See Chapter 19 [Actualizaciones de seguridad], page 723, para información sobre injertos.

`--map-file=`*archivo*

Escribe un mapa gráfico del uso del disco en formato PNG en el *archivo*.

Para el ejemplo previo, el mapa tiene esta pinta:


Esta opción necesita que la biblioteca Guile-Charting (`https://wingolog.org/software/guile-charting/`) esté instalada y visible en la ruta de búsqueda de módulos Guile. Cuando no es el caso, `guix size` produce un error al intentar cargarla.

`--system=`*sistema*
`-s` *sistema*

Considera paquetes para *sistema*—por ejemplo, `x86_64-linux`.

`--load-path=`*directorio*
`-L` *directorio*

Añade *directorio* al frente de la ruta de búsqueda de módulos de paquetes (see Section 8.1 [Módulos de paquetes], page 101).

Esto permite a las usuarias definir sus propios paquetes y hacerlos visibles a las herramientas de línea de órdenes.

## 9.10 Invocación de `guix graph`

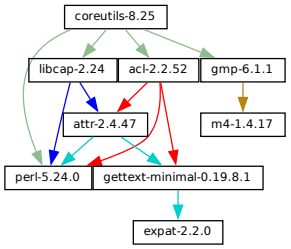
Packages and their dependencies form a *graph*, specifically a directed acyclic graph (DAG). It can quickly become difficult to have a mental model of the package DAG, so the `guix graph` command provides a visual representation of the DAG. By default, `guix graph` emits a DAG representation in the input format of Graphviz (`https://www.graphviz.org/`), so its output can be passed directly to the `dot` command of Graphviz. It can also emit an HTML page with embedded JavaScript code to display a "chord diagram" in a Web browser, using the d3.js (`https://d3js.org/`) library, or emit Cypher queries to construct a graph in a graph database supporting the openCypher (`https://www.opencypher.org/`) query language. With `--path`, it simply displays the shortest path between two packages. The general syntax is:

```
guix graph opciones paquete...
```

Por ejemplo, la siguiente orden genera un archivo PDF que representa el GAD para GNU Core Utilities, mostrando sus dependencias en tiempo de construcción:

```
guix graph coreutils | dot -Tpdf > gad.pdf
```

La salida es algo así:




Bonito y pequeño grafo, ¿no?

Puede encontrar más agradable la navegación interactiva del grafo con la orden `xdot` (del paquete `xdot`):

```
guix graph coreutils | xdot -
```

¡Pero hay más de un grafo! El grafo previo es conciso: es el grafo de los objetos package, omitiendo las entradas implícitas como GCC, libc, grep, etc. Es habitualmente útil tener un grafo conciso así, pero a veces una puede querer ver más detalles. `guix graph` implementa varios tipos de grafos, lo que le permite seleccionar el nivel de detalle:

`package` Este es el tipo por defecto usado en el ejemplo previo. Muestra el GAD de objetos package, excluyendo dependencias implícitas. Es conciso, pero deja fuera muchos detalles.

`reverse-package`
Esto muestra el GAD *inverso* de paquetes. Por ejemplo:

```
guix graph --type=reverse-package ocaml
```

... emite el grafo de paquetes que dependen *explícitamente* de OCaml (si también tiene interés en casos donde OCaml es una dependencia implícita, véase `reverse-bag` a continuación).

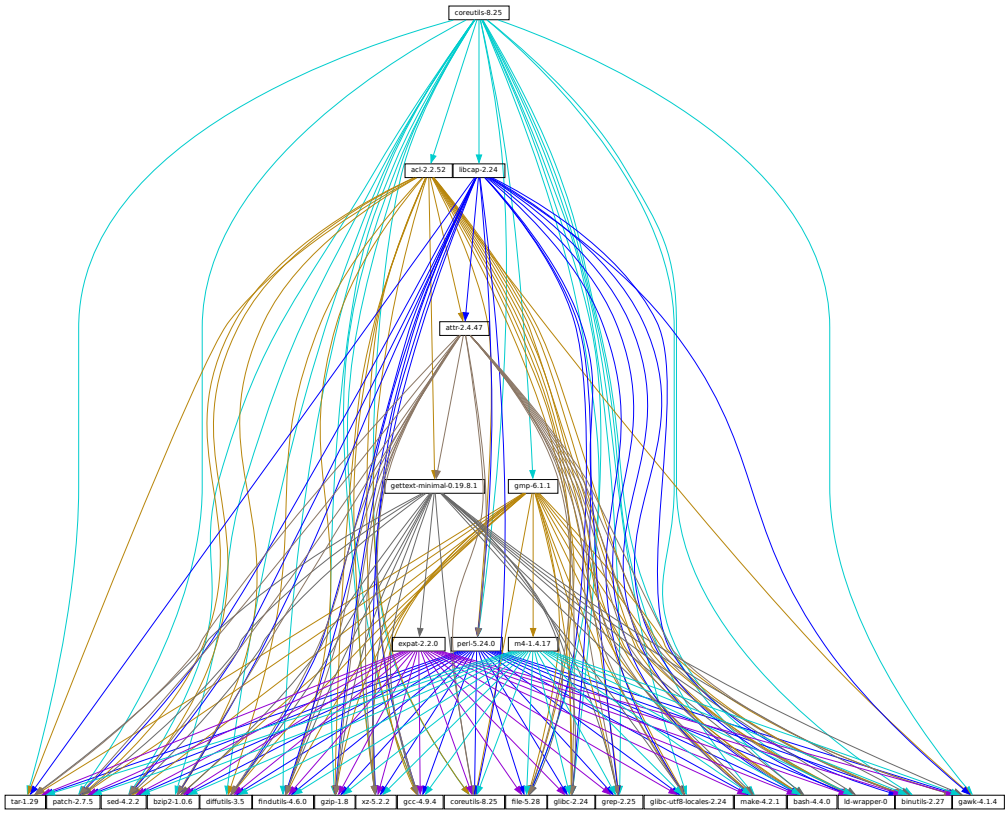
Fíjese que esto puede producir grafos inmensos para los paquetes básicos. Si todo lo que quiere saber es el número de paquetes que dependen de uno determinado, use `guix refresh --list-dependent` (see Section 9.6 [Invocación de guix refresh], page 207).

`bag-emerged`
Este es el GAD del paquete, *incluyendo* entradas implícitas.

Por ejemplo, la siguiente orden:

```
guix graph --type=bag-emerged coreutils
```

... emite este grafo más grande:




En la parte inferior del grafo, vemos todas las entradas implícitas de *gnu-build-system* (see Section 8.5 [Sistemas de construcción], page 123).

Ahora bien, fíjese que las dependencias de estas entradas implícitas—es decir, las *dependencias del lanzamiento inicial* (see Chapter 20 [Lanzamiento inicial], page 725)—no se muestran aquí para mantener una salida concisa.

`bag` Similar a `bag-emerged`, pero esta vez incluye todas las dependencias del lanzamiento inicial.

`bag-with-origins`
Similar a `bag`, pero también muestra los orígenes y sus dependencias.

`reverse-bag`
Muestra el GAD *inverso* de paquetes. Al contrario que `reverse-package`, también tiene en cuenta las dependencias implícitas. Por ejemplo:

```
guix graph -t reverse-bag dune
```

... emite el grafo de tosos los paquetes que dependen de Dune, directa o indirectamente. Ya que Dune es una dependencia *implícita* de muchos paquetes *vía* `dune-build-system`, esto mostrará un gran número de paquetes, mientras que `reverse-package` mostraría muy pocos si muestra alguno.

`derivation`
: Esta es la representación más detallada: muestra el GAD de derivaciones (see Section 8.10 [Derivaciones], page 159) y elementos simples del almacén. Comparada con las representaciones previas, muchos nodos adicionales son visibles, incluyendo los guiones de construcción, parches, módulos Guile, etc.

  Para este tipo de grafo, también es posible pasar un nombre de archivo `.drv` en vez del nombre del paquete, como en:

  ```
  guix graph -t derivation $(guix system build -d my-config.scm)
  ```

`module`
: Este es el grafo de los *módulos de paquete* (see Section 8.1 [Módulos de paquetes], page 101). Por ejemplo, la siguiente orden muestra el grafo para el módulo de paquetes que define el paquete `guile`:

  ```
  guix graph -t module guile | xdot -
  ```

Todos los tipos previos corresponden a las *dependencias durante la construcción*. El grafo siguiente representa las *dependencias en tiempo de ejecución*:

`references`
: Este es el grafo de *referencias* de la salida de un paquete, como lo devuelve `guix gc --references` (see Section 5.6 [Invocación de guix gc], page 53).

  Si la salida del paquete proporcionado no está disponible en el almacén, `guix graph` intenta obtener la información de dependencias desde las sustituciones.

  Aquí también puede proporcionar un nombre de archivo del almacén en vez de un nombre de paquete. Por ejemplo, la siguiente orden produce el grafo de referencias de su perfil (¡el cuál puede ser grande!):

  ```
  guix graph -t references $(readlink -f ~/.guix-profile)
  ```

`referrers`
: Este es el grafo de *referentes* de la salida de un paquete, como lo devuelve `guix gc --referrers` (see Section 5.6 [Invocación de guix gc], page 53).

  Depende exclusivamente de información en su almacén. Por ejemplo, supongamos que la versión actual de Inkscape está disponible en 10 perfiles en su máquina; `guix graph -t referrers inkscape` mostrará un grafo cuya raíz es Inkscape y con esos 10 perfiles enlazados a ella.

  Puede ayudar a determinar qué impide que un elemento del almacén sea recolectado.

Habitualmente el grafo del paquete por el que tiene interés no entrará en su pantall, y en cualquier caso todo lo que quiere saber es *por qué* dicho paquete depende de algún paquete que parece no tener relación. La opción `--path` le indica a `guix graph` que muestre la ruta más corta entre dos paquetes (o derivaciones, o elementos del almacén, etcétera):

```
$ guix graph --path emacs libunistring
emacs@26.3
mailutils@3.9
libunistring@0.9.10
$ guix graph --path -t derivation emacs libunistring
/gnu/store/...-emacs-26.3.drv
/gnu/store/...-mailutils-3.9.drv
```

```
/gnu/store/...-libunistring-0.9.10.drv
$ guix graph --path -t references emacs libunistring
/gnu/store/...-emacs-26.3
/gnu/store/...-libidn2-2.2.0
/gnu/store/...-libunistring-0.9.10
```

Sometimes you still want to visualize the graph but would like to trim it so it can actually be displayed. One way to do it is via the `--max-depth` (or `-M`) option, which lets you specify the maximum depth of the graph. In the example below, we visualize only `libreoffice` and the nodes whose distance to `libreoffice` is at most 2:

```
guix graph -M 2 libreoffice | xdot -f fdp -
```

Mind you, that's still a big ball of spaghetti, but at least `dot` can render it quickly and it can be browsed somewhat.

Las opciones disponibles son las siguientes:

`--type=`*tipo*
`-t` *tipo* Produce un grafo de salida de *tipo*, donde *tipo* debe ser uno de los valores enumerados previamente.

`--list-types`
Enumera los tipos de grafos implementados.

`--backend=`*motor*
`-b` *motor* Produce un grafo usando el *motor* seleccionado.

`--list-backends`
Enumera los motores de grafos implementados.

Actualmente, los motores disponibles son Graphviz y d3.js.

`--path` Muestra la ruta más corta entre dos nodos del tipo especificado por la opción `--type`. El ejemplo siguiente muestra la ruta más corta entre `libreoffice` y `llvm` de acuerdo con las referencias de `libreoffice`:

```
$ guix graph --path -t references libreoffice llvm
/gnu/store/...-libreoffice-6.4.2.2
/gnu/store/...-libepoxy-1.5.4
/gnu/store/...-mesa-19.3.4
/gnu/store/...-llvm-9.0.1
```

`--expression=`*expr*
`-e` *expr* Considera el paquete al que evalúa *expr*

Es útil para hacer una referencia precisa de un paquete concreto, como en este ejemplo:

```
guix graph -e '(@@ (gnu packages commencement) gnu-make-final)'
```

`--system=`*sistema*
`-s` *sistema*
Muestra el grafo para *sistema*—por ejemplo, `i686-linux`.

El grafo de dependencias del paquete es altamente independiente de la arquitectura, pero existen algunas partes dependientes de la arquitectura que esta opción le permite visualizar.

`--load-path=`*directorio*
`-L` *directorio*

Añade *directorio* al frente de la ruta de búsqueda de módulos de paquetes (see Section 8.1 [Módulos de paquetes], page 101).

Esto permite a las usuarias definir sus propios paquetes y hacerlos visibles a las herramientas de línea de órdenes.

Además de esto, `guix graph` permite todas las opciones habituales de transformación de paquetes (see Section 9.1.2 [Opciones de transformación de paquetes], page 184). Esto facilita la visualización del efecto de una transformación de reescritura de grafo como `--with-input`. Por ejemplo, la siguiente orden muestra el grafo de `git` una vez que `openssl` ha sido reemplazado por `libressl` en todos los nodos del grafo:

```
guix graph git --with-input=openssl=libressl
```

¡Tantas posibilidades, tanta diversión!

## 9.11 Invocación de `guix publish`

El propósito de `guix publish` es permitir a las usuarias compartir fácilmente su almacén con otras, quienes pueden usarlo como servidor de sustituciones (see Section 5.3 [Sustituciones], page 46).

When `guix publish` runs, it spawns an HTTP server which allows anyone with network access to obtain substitutes from it. This means that any machine running Guix can also act as if it were a build farm, since the HTTP interface is compatible with Cuirass, the software behind the `bordeaux.guix.gnu.org` build farm.

Por seguridad, cada sustitución se firma, permitiendo a las receptoras comprobar su autenticidad e integridad (see Section 5.3 [Sustituciones], page 46). Debido a que `guix publish` usa la clave de firma del sistema, que es únicamente legible por la administradora del sistema, debe iniciarse como root; la opción `--user` hace que renuncie a sus privilegios tan pronto como sea posible.

El par claves de firma debe generarse antes de ejecutar `guix publish`, usando `guix archive --generate-key` (see Section 5.11 [Invocación de guix archive], page 66).

When the `--advertise` option is passed, the server advertises its availability on the local network using multicast DNS (mDNS) and DNS service discovery (DNS-SD), currently *via* Guile-Avahi (see *Using Avahi in Guile Scheme Programs*).

La sintaxis general es:

```
guix publish opciones...
```

La ejecución de `guix publish` sin ningún parámetro adicional lanzará un servidor HTTP en el puerto 8080:

```
guix publish
```

`guix publish` can also be started following the systemd "socket activation" protocol (see Section "Service De- and Constructors" in *The GNU Shepherd Manual*).

Una vez el servidor de publicación ha sido autorizado, el daemon puede descargar sustituciones de él. See Section 5.3.3 [Obtención de sustiticiones desde otros servidores], page 48.

Por defecto, `guix publish` comprime los archivos al vuelo cuando es necesario. Este modo "al vuelo" es conveniente ya que no necesita configuración y está disponible inmediatamente. No obstante, cuando se proporciona servicio a muchos clientes, se recomienda usar la opción `--cache`, que activa el almacenamiento en caché de los archivos antes de enviarlos a los clientes—véase a continuación para más detalles. La orden `guix weather` proporciona una forma fácil de comprobar lo que proporciona un servidor (see Section 9.15 [Invocación de guix weather], page 234).

Además `guix publish` también sirve como un espejo de acceso por contenido a archivos de fuentes a los que los registros `origin` hacen referencia (see Section 8.2.2 [Referencia de origin], page 110). Por ejemplo, si asumimos que `guix publish` se ejecuta en `example.org`, la siguiente URL devuelve directamente el archivo `hello-2.10.tar.gz` con el hash SHA256 proporcionado (representado en formato `nix-base32`, see Section 9.4 [Invocación de guix hash], page 197).

```
http://example.org/file/hello-2.10.tar.gz/sha256/0ssi1...ndq1i
```

Obviamente estas URL funcionan solamente para archivos que se encuentran en el almacén; en otros casos devuelven un 404 ("No encontrado").

Los log de construcción están disponibles desde URL `/log` como:

```
http://example.org/log/gwspk...-guile-2.2.3
```

Cuando `guix-daemon` está configurado para almacenar comprimidos los log de construcción, como sucede de forma predeterminada (see Section 2.3 [Invocación de guix-daemon], page 12), las URL `/log` devuelven los log igualmente comprimidos, con un `Content-Type` adecuado y/o una cabecera `Content-Encoding`. Recomendamos ejecutar `guix-daemon` con `--log-compression=gzip` ya que los navegadores Web pueden extraer el contenido automáticamente, lo cual no es el caso con la compresión bzip2.

Las siguientes opciones están disponibles:

`--port=`*puerto*
`-p` *puerto* Escucha peticiones HTTP en *puerto*.

`--listen=`*dirección*
: Escucha en la interfaz de red de la *dirección*. El comportamiento predeterminado es aceptar conexiones de cualquier interfaz.

`--user=`*usuaria*
`-u` *usuaria*
: Cambia los privilegios a los de *usuaria* tan pronto como sea posible—es decir, una vez el socket del servidor esté abierto y la clave de firma haya sido leída.

`--compression[=`*método*`[:`*nivel*`]]`
`-C [`*método*`[:`*nivel*`]]`
: Compress data using the given *method* and *level*. *method* is one of `lzip`, `zstd`, and `gzip`; when *method* is omitted, `gzip` is used.

  Cuando el *nivel* es cero, desactiva la compresión. El rango 1 a 9 corresponde a distintos niveles de compresión gzip: 1 es el más rápido, y 9 es el mejor (intensivo a nivel de CPU). El valor predeterminado es 3.

  Usually, `lzip` compresses noticeably better than `gzip` for a small increase in CPU usage; see benchmarks on the lzip Web page (`https://nongnu.org/lzip/`

`lzip_benchmark.html`). However, `lzip` achieves low decompression throughput (on the order of 50 MiB/s on modern hardware), which can be a bottleneck for someone who downloads over a fast network connection.

The compression ratio of `zstd` is between that of `lzip` and that of `gzip`; its main advantage is a high decompression speed (`https://facebook.github.io/zstd/`).

A menos que se use `--cache`, la compresión ocurre al vuelo y los flujos comprimidos no se almacenan en caché. Por tanto, para reducir la carga en la máquina que ejecuta `guix publish`, puede ser una buena idea elegir un nivel de compresión bajo, ejecutar `guix publish` detrás de una pasarela con caché o usar `--cache`. El uso de `--cache` tiene la ventaja de que permite a `guix publish` añadir la cabecera HTTP `Content-Length` a sus respuestas.

Se puede repetir esta opción, en cuyo caso cada sustitución se comprime usando todos los métodos seleccionados, y todos son anunciados. Esto es útil cuando las usuarias pueden no implementar todos los métodos de compresión: pueden seleccionar el que implementan.

`--cache=`*directorio*
`-c` *directorio*

Almacena en caché los archivos y metadatos (URL `.narinfo`) en *directorio* y únicamente proporciona archivos que están en la caché.

Cuando se omite esta opción, los archivos y metadatos se crean al vuelo. Esto puede reducir el ancho de banda disponible, especialmente cuando la compresión está activa, ya que se puede llegar al límite de la CPU. Otra desventaja del modo predeterminado es que la longitud de los archivos no se conoce con anterioridad, por lo que `guix publish` no puede añadir la cabecera HTTP `Content-Length` a sus respuestas, lo que a su vez previene que los clientes conozcan la cantidad de datos a descargar.

De manera contraria, cuando se usa `--cache`, la primera petición de un elemento del almacén (a través de una URL `.narinfo`) inicia un proceso en segundo plano para *cocinar* el archivo—calcular su `.narinfo` y comprimirlo, en caso necesario. Una vez el archivo está alojado en la caché de *directorio*, las siguientes peticiones obtendrán un resultado satisfactorio y se ofrecerá el contenido directamente desde la caché, lo que garantiza que los clientes obtienen el mejor ancho de banda posible.

La primera petción de `.narinfo` devuelve no obstante el código 200, en el caso de que el elemento del almacén sea "lo suficientemente pequeño", es decir que su tamaño sea inferior al límite de bajo el que se ignora la caché—véase la opción `--cache-bypass-threshold` a continuación. De este modo, los clientes no deben esperar hasta que el archivo se haya cocinado. Con elementos del almacén de mayor tamaño la primera petición `.narinfo` devuelve el código 404, lo que significa que los clientes deben esperar hasta que el archivo se haya cocinado.

El proceso de "cocinado" se realiza por hilos de trabajo. Por defecto, se crea un hilo por núcleo de la CPU, pero puede ser personalizado. Véase `--workers` a continuación.

Cuando se usa `--ttl`, las entradas en caché se borran automáticamente cuando hayan expirado.

`--workers=`*N*
: Cuando se usa `--cache`, solicita la creación de *N* hilos de trabajo para "cocinar" archivos.

`--ttl=`*ttl*
: Produce cabeceras HTTP `Cache-Control` que anuncian un tiempo-de-vida (TTL) de *ttl*. *ttl* debe indicar una duración: `5d` significa 5 días, `1m` significa un mes, etc.

  Esto permite a la usuaria de Guix mantener información de sustituciones en la caché durante *ttl*. No obstante, fíjese que `guix publish` no garantiza en sí que los elementos del almacén que proporciona de hecho permanezcan disponibles hasta que *ttl* expire.

  Adicionalmente, cuando se usa `--cache`, las entradas en caché que no hayan sido accedidas en *ttl* y no tengan un elemento correspondiente en el almacén pueden ser borradas.

`--negative-ttl=`*ttl*
: Similarly produce `Cache-Control` HTTP headers to advertise the time-to-live (TTL) of *negative* lookups—missing store items, for which the HTTP 404 code is returned. By default, no negative TTL is advertised.

  This parameter can help adjust server load and substitute latency by instructing cooperating clients to be more or less patient when a store item is missing.

`--cache-bypass-threshold=`*tamaño*
: Cuando se usa en conjunto con la opción `--cache`, los elementos del almacén cuyo tamaño sea inferior a *tamaño* están disponibles de manera inmediata, incluso cuando no están todavía en la caché. *tamaño* es el número de bytes, o se puedem usar sufijos como `M` para megabytes, etcétera. El valor predeterminado es `10M`.

  La opción de omisión de la cache le permite reducir la latencia de publicación a los clientes a expensas de un posible incremento en el uso de E/S y procesador en el lado del servidor: dependiendo de los patrones de acceso de los clientes, dichos elementos del almacén pueden ser cocinados varias veces hasta que una copia se encuentre disponible en la caché.

  Incrementar el valor límite puede ser útil para servidores que tengan pocas usuarias, o para garantizar que dichas usuarias obtienen sustituciones incluso con elementos del almacén que no son populares.

`--nar-path=`*ruta*
: Usa *ruta* como el prefijo para las URL de los archivos "nar" (see Section 5.11 [Invocación de guix archive], page 66).

  Por defecto, los archivos nar se proporcionan en una URL como `/nar/gzip/...-coreutils-8.25`. Esta opción le permite cambiar la parte `/nar` por *ruta*.

`--public-key=`*archivo*
`--private-key=`*archivo*
: Usa los *archivo*s específicos como el par de claves pública y privada usadas para firmar los elementos del almacén publicados.

  Los archivos deben corresponder al mismo par de claves (la clave privada se usa para la firma y la clave pública simplemente se anuncia en los metadatos de la firma). Deben contener claves en el formato canónico de expresiones-S como el producido por `guix archive --generate-key` (see Section 5.11 [Invocación de guix archive], page 66). Por defecto, se usan `/etc/guix/signing-key.pub` y `/etc/guix/signing-key.sec`.

`--repl[=`*puerto*`]`
`-r [`*puerto*`]`
: Lanza un servidor REPL Guile (see Section "REPL Servers" in *GNU Guile Reference Manual*) en *puerto* (37146 por defecto). Esto se usa principalmente para la depuración de un servidor `guix publish` en ejecución.

Activar `guix publish` en el sistema Guix consiste en solo una línea: simplemente instancie un servicio `guix-publish-service-type` en el campo `services` de su declaración del sistema operativo `operating-system` (see [guix-publish-service-type], page 291)

Si en vez de eso ejecuta Guix en una distribución distinta, siga estas instrucciones:

- Si su distribución anfitriona usa el sistema de inicio systemd:

```
# ln -s ~root/.guix-profile/lib/systemd/system/guix-publish.service \
        /etc/systemd/system/
# systemctl start guix-publish && systemctl enable guix-publish
```

- Si su distribución anfitriona usa el sistema de inicio Upstart:

```
# ln -s ~root/.guix-profile/lib/upstart/system/guix-publish.conf /etc/init/
# start guix-publish
```

- En otro caso, proceda de forma similar con el sistema de inicio de su distribución.

## 9.12 Invocación de `guix challenge`

¿Los binarios que proporciona este servidor realmente corresponden al código fuente que dice construir? ¿Es determinista el proceso de construcción de un paquete? Estas son las preguntas que la orden `guix challenge` intenta responder.

La primera es obviamente una cuestión importante: antes de usar un servidor de sustituciones (see Section 5.3 [Sustituciones], page 46), es importante haber *verificado* que proporciona los binarios correctos, y por tanto *ponerlo a prueba*[3]. La segunda es lo que permite la primera: si las construcciones de los paquetes son deterministas, construcciones independientes deberían emitir el mismo resultado, bit a bit; si el servidor proporciona un binario diferente al obtenido localmente, o bien está corrupto o bien tiene intenciones perniciosas.

Sabemos que el hash que se muestra en los nombres de archivo en `/gnu/store` es el hash de todas las entradas del proceso que construyó el archivo o directorio—compiladores, bibliotecas, guiones de construcción, etc. (see Chapter 1 [Introducción], page 1). Asumiendo

[3] NdT: challenge en inglés.

procesos de construcción deterministas, un nombre de archivo del almacén debe corresponder exactamente a una salida de construcción. `guix challenge` comprueba si existe, realmente, una asociación unívoca comparando la salida de la construcción de varias construcciones independientes de cualquier elemento del almacén proporcionado.

La salida de la orden muestra algo así:

```
$ guix challenge \
  --substitute-urls="https://bordeaux.guix.gnu.org https://guix.example.org" \
  openssl git pius coreutils grep
updating substitutes from 'https://bordeaux.guix.gnu.org'... 100.0%
updating substitutes from 'https://guix.example.org'... 100.0%
/gnu/store/...-openssl-1.0.2d contents differ:
  local hash: 0725l22r5jnzazaacncwsvp9kgf42266ayyp814v7djxs7nk963q
  https://bordeaux.guix.gnu.org/nar/...-openssl-1.0.2d: 0725l22r5jnzazaacncwsvp9kgf42266ayyp814v7djxs7nk9
  https://guix.example.org/nar/...-openssl-1.0.2d: 1zy4fmaaqcnjrzzajkdn3f5gmjk754b43qkq47llbyak9z0qjyim
  differing files:
    /lib/libcrypto.so.1.1
    /lib/libssl.so.1.1

/gnu/store/...-git-2.5.0 contents differ:
  local hash: 00p3bmryhjxrhpn2gxs2fy0a15lnip05l97205pgbk5ra395hyha
  https://bordeaux.guix.gnu.org/nar/...-git-2.5.0: 069nb85bv4d4a6slrwjdy8v1cn4cwspm3kdbmyb81d6zckj3nq9f
  https://guix.example.org/nar/...-git-2.5.0: 0mdqa9w1p6cmli6976v4wi0sw9r4p5prkj7lzfd1877wk11c9c73
  differing file:
    /libexec/git-core/git-fsck

/gnu/store/...-pius-2.1.1 contents differ:
  local hash: 0k4v3m9z1zp8xzzizb7d8kjj72f9172xv078sq4wl73vnq9ig3ax
  https://bordeaux.guix.gnu.org/nar/...-pius-2.1.1: 0k4v3m9z1zp8xzzizb7d8kjj72f9172xv078sq4wl73vnq9ig3ax
  https://guix.example.org/nar/...-pius-2.1.1: 1cy25x1a4fzq5rk0pmvc8xhwyffnqz95h2bpvqsz2mpvlbccy0gs
  differing file:
    /share/man/man1/pius.1.gz

...

5 store items were analyzed:
  - 2 (40.0%) were identical
  - 3 (60.0%) differed
  - 0 (0.0%) were inconclusive
```

In this example, `guix challenge` queries all the substitute servers for each of the fives packages specified on the command line. It then reports those store items for which the servers obtained a result different from the local build (if it exists) and/or different from one another; here, the '`local hash`' lines indicate that a local build result was available for each of these packages and shows its hash.

As an example, `guix.example.org` always gets a different answer. Conversely, `bordeaux.guix.gnu.org` agrees with local builds, except in the case of Git. This might indicate that the build process of Git is non-deterministic, meaning that its output varies as a function of various things that Guix does not fully control, in spite of building packages in isolated environments (see Section 5.1 [Características], page 35). Most common sources of non-determinism include the addition of timestamps in build results, the inclusion of random numbers, and directory listings sorted by inode number. See `https://reproducible-builds.org/docs/`, for more information.

Para encontrar cuál es el problema con este binario Git, la aproximación más fácil es ejecutar:

```
guix challenge git \
  --diff=diffoscope \
  --substitute-urls="https://bordeaux.guix.gnu.org https://guix.example.org"
```

Esto invoca automáticamente `diffoscope`, que muestra información detallada sobre los archivos que son diferentes.

De manera alternativa, se puede hacer algo parecido a esto (see Section 5.11 [Invocación de guix archive], page 66):

```
$ wget -q -O - https://bordeaux.guix.gnu.org/nar/lzip/...-git-2.5.0 \
   | lzip -d | guix archive -x /tmp/git
$ diff -ur --no-dereference /gnu/store/...-git.2.5.0 /tmp/git
```

This command shows the difference between the files resulting from the local build, and the files resulting from the build on `bordeaux.guix.gnu.org` (see Section "Overview" in *Comparing and Merging Files*). The `diff` command works great for text files. When binary files differ, a better option is Diffoscope (`https://diffoscope.org/`), a tool that helps visualize differences for all kinds of files.

Una vez haya realizado este trabajo, puede determinar si las diferencias son debidas a un procedimiento de construcción no-determinista o a un servidor con intenciones ocultas. Intentamos duramente eliminar las fuentes de indeterminismo en los paquetes para facilitar la verificación de sustituciones, pero por supuesto es un proceso que implica no solo a Guix, sino a una gran parte de la comunidad del software libre. Entre tanto, `guix challenge` es una herramienta para ayudar a afrontar el problema.

If you are writing packages for Guix, you are encouraged to check whether `bordeaux.guix.gnu.org` and other substitute servers obtain the same build result as you did with:

```
guix challenge package
```

La sintaxis general es:

```
guix challenge options argument...
```

where *argument* is a package specification such as `guile@2.0` or `glibc:debug` or, alternatively, a store file name as returned, for example, by `guix build` or `guix gc --list-live`.

Cuando se encuentra una diferencia entre el hash de un elemento construido localmente y el proporcionado por un servidor de sustituciones; o entre las sustituciones proporcionadas por distintos servidores, esto es mostrado como en el ejemplo previo y el código de salida es 2 (otros valores código de salida distintos de cero denominan otros tipos de error).

La única opción de importancia es:

`--substitute-urls=`*urls*
: Considera *urls* la lista separada por espacios de URL de fuentes de sustituciones con las que realizar la comparación.

`--diff=`*modo*
: Muestra las diferencias encontradas de acuerdo con *modo*, uno de los siguientes:

  `simple` (el predeterminado)
  : Muestra la lista de archivos que son diferentes.

`diffoscope`

*orden* Invoca Diffoscope (`https://diffoscope.org/`) y le proporciona los dos directorios cuyo contenido es diferente.

Cuando *orden* es una ruta absoluta, ejecuta *orden* en vez de Diffoscope.

`none` No muestra más detalles sobre las diferencias.

Por tanto, a menos que se proporcione `--diff=none`, `guix challenge` descarga los contenidos de los elementos del almacén de los servidores de sustituciones proporcionados para poder compararlos.

`--verbose`
`-v` Muestra detalles sobre coincidencias (contenidos idénticos) además de información sobre las discrepancias.

## 9.13 Invocación de `guix copy`

La orden `guix copy` copia elementos del almacén de una máquina al de otra a través de una conexión de shell seguro (SSH)[4]. Por ejemplo, la siguiente orden copia el paquete `coreutils`, el perfil de la usuaria y todas sus dependencias a *dirección*, ingresando en el sistema como *usuaria*:

```
guix copy --to=user@host \
          coreutils $(readlink -f ~/.guix-profile)
```

Si alguno de los elementos del almacén a copiar ya están presentes en *dirección*, no se envían realmente.

La siguiente orden obtiene `libreoffice` y `gimp` de *dirección*, asumiendo que estén disponibles allí:

```
guix copy --from=dirección libreoffice gimp
```

La conexión SSH se establece usando el cliente Guile-SSH, que es compatible con OpenSSH: tiene en cuenta `~/.ssh/known_hosts` y `~/.ssh/config`, y usa el agente SSH para la identificación.

La clave usada para firmar los elementos enviados debe estar aceptada por la máquina remota. Del mismo modo, la clave usada por la máquina remota para firmar los elementos recibidos debe estar en `/etc/guix/acl` de modo que sea aceptada por su propio daemon. See Section 5.11 [Invocación de guix archive], page 66, para más información sobre la verificación de elementos del almacén.

La sintaxis general es:

```
guix copy [--to=spec|--from=spec] elementos...
```

Siempre debe especificar una de las siguientes opciones:

`--to=spec`
`--from=spec`

Especifica la máquina a la que mandar o desde la que recibir. *spec* debe ser una especificación SSH como `example.org`, `carlos@example.org`, or `carlos@example.org:2222`.

[4] Esta orden únicamente está disponible cuando ha encontrado Guile-SSH. See Section 22.1 [Requisitos], page 734, para detalles.

Los *elementos* pueden ser tanto nombres de paquetes, como `gimp`, como elementos del almacén, como `/gnu/store/...-idutils-4.6`.

Cuando se especifica el nombre del paquete a enviar, primero se construye si es necesario, a menos que se use `--dry-run`. Se aceptan las opciones comunes de construcción (see Section 9.1.1 [Opciones comunes de construcción], page 181).

## 9.14 Invocación de `guix container`

> **Nota:** En la versión c7888f5, esta herramienta es experimental. La interfaz está sujeta a cambios radicales en el futuro.

The purpose of `guix container` is to manipulate processes running within an isolated environment, commonly known as a "container", typically created by the `guix shell` (see Section 7.1 [Invoking guix shell], page 79) and `guix system container` (see Section 11.16 [Invocación de guix system], page 634) commands.

La sintaxis general es:

```
guix container acción opciones...
```

*acción* especifica la operación a realizar con el contenedor, y *opcines* especifica los parámetros específicos del contexto para la acción.

Las siguientes acciones están disponibles:

`exec` Ejecute una orden en el contexto de un contenedor en ejecución.

La sintaxis es:

```
guix container exec pid programa parámetros...
```

*pid* especifica el ID del proceso del contenedor en ejecución. *programa* especifica el nombre del archivo ejecutable dentro del sistema de archivos raíz del contenedor. *parámetros* son opciones adicionales que se pasarán a *programa*.

La siguiente orden lanza un shell interactivo de ingreso al sistema dentro de un contenedor del sistema, iniciado por `guix system container`, y cuyo ID de proceso es 9001:

```
guix container exec 9001 /run/current-system/profile/bin/bash --login
```

Fíjese que el *pid* no puede ser el proceso creador del contenedor. Debe ser el PID 1 del contenedor o uno de sus procesos hijos.

## 9.15 Invocación de `guix weather`

Occasionally you're grumpy because substitutes are lacking and you end up building packages by yourself (see Section 5.3 [Sustituciones], page 46). The `guix weather` command reports on substitute availability on the specified servers so you can have an idea of whether you'll be grumpy today. It can sometimes be useful info as a user, but it is primarily useful to people running `guix publish` (see Section 9.11 [Invocación de guix publish], page 226). Sometimes substitutes *are* available but they are not authorized on your system; `guix weather` reports it so you can authorize them if you want (see Section 5.3.3 [Obtención de sustiticiones desde otros servidores], page 48).

Esta es una ejecución de ejemplo:

```
$ guix weather --substitute-urls=https://guix.example.org
```

```
computing 5,872 package derivations for x86_64-linux...
looking for 6,128 store items on https://guix.example.org..
updating substitutes from 'https://guix.example.org'... 100.0%
https://guix.example.org
  43.4% substitutes available (2,658 out of 6,128)
  7,032.5 MiB of nars (compressed)
  19,824.2 MiB on disk (uncompressed)
  0.030 seconds per request (182.9 seconds in total)
  33.5 requests per second

  9.8% (342 out of 3,470) of the missing items are queued
  867 queued builds
      x86_64-linux: 518 (59.7%)
      i686-linux: 221 (25.5%)
      aarch64-linux: 128 (14.8%)
  build rate: 23.41 builds per hour
      x86_64-linux: 11.16 builds per hour
      i686-linux: 6.03 builds per hour
      aarch64-linux: 6.41 builds per hour
```

Como puede ver, informa de la fracción de todos los paquetes para los cuales hay sustituciones en el servidor—independientemente de que las sustituciones estén activadas, e independientemente de si la clave de firma del servidor está autorizada. También informa del tamaño de los archivos comprimidos ("nar") proporcionados por el servidor, el tamaño que los elementos correspondientes del almacén ocupan en el almacén (asumiendo que la deduplicación está apagada) y el caudal de proceso del servidor. La segunda parte proporciona estadísticas de integración continua (CI), si el servidor lo permite. Además, mediante el uso de la opción `--coverage`, `guix weather` puede enumerar sustituciones de paquetes "importantes" que no se encuentren en el servidor (véase más adelante).

Para conseguirlo, `guix weather` consulta los metadatos HTTP(S) (*narinfo*s) de todos los elementos relevantes del almacén. Como `guix challenge`, ignora las firmas en esas sustituciones, lo cual es inocuo puesto que la orden únicamente obtiene estadísticas y no puede instalar esas sustituciones.

La sintaxis general es:

```
guix weather opciones... [paquetes...]
```

Cuando se omite *paquetes*, `guix weather` comprueba la disponibilidad de sustituciones para *todos* los paquetes, o para aquellos especificados con la opción `--manifest`; en otro caso considera únicamente los paquetes especificados. También es posible consultar tipos de sistema específicos con `--system`. `guix weather` termina con un código de salida distinto a cero cuando la fracción de sustituciones disponibles se encuentra por debajo del 100%.

Las opciones disponibles se enumeran a continuación.

`--substitute-urls=urls`
: *urls* is the space-separated list of substitute server URLs to query. When this option is omitted, the URLs specified with the `--substitute-urls` option of `guix-daemon` are used or, as a last resort, the default set of substitute URLs.

`--system=`*sistema*
`-s` *sistema*

Consulta sustituciones para *sistema*—por ejemplo, `aarch64-linux`. Esta opción se puede repetir, en cuyo caso `guix weather` consultará las sustituciones para varios tipos de sistema.

`--manifest=`*archivo*

En vez de consultar las sustituciones de todos los paquetes, consulta únicamente los especificados en *archivo*. *archivo* debe contener un *manifiesto*, como el usado en la opción `-m` de `guix package` (see Section 5.2 [Invocación de guix package], page 36).

Esta opción puede repetirse varias veces, en cuyo caso los manifiestos se concatenan.

`--expression=`*expr*
`-e` *expr* Considera el paquete al que evalúa *expr*

A typical use case for this option is specifying a package that is hidden and thus cannot be referred to in the usual way, as in this example:

```
guix weather -e '(@@ (gnu packages rust) rust-bootstrap)'
```

This option can be repeated.

`--coverage[=`*numero*`]`
`-c [`*numero*`]`

Informa de la cobertura de sustituciones para paquetes: enumera paquetes con al menos *número* dependientes (cero por omisión) para los cuales no haya sustituciones disponibles. Los paquetes dependientes en sí no se enumeran: si *b* depende de *a* y *a* no tiene sustituciones disponibles, únicamente se muestra *a*, aunque *b* normalmente no tenga sustituciones tampoco. El resultado es más o menos así:

```
$ guix weather --substitute-urls=https://bordeaux.guix.gnu.org
https://ci.guix.gnu.org -c 10
computing 8,983 package derivations for x86_64-linux...
looking for 9,343 store items on https://bordeaux.guix.gnu.org
https://ci.guix.gnu.org...
updating substitutes from 'https://bordeaux.guix.gnu.org https://ci.guix.gnu
https://bordeaux.guix.gnu.org https://ci.guix.gnu.org
  64.7% substitutes available (6,047 out of 9,343)
...
2502 packages are missing from 'https://bordeaux.guix.gnu.org
https://ci.guix.gnu.org' for 'x86_64-linux', among which:
    58  kcoreaddons@5.49.0      /gnu/store/...-kcoreaddons-5.49.0
    46  qgpgme@1.11.1           /gnu/store/...-qgpgme-1.11.1
    37  perl-http-cookiejar@0.008  /gnu/store/...-perl-http-cookiejar-0.008
    ...
```

What this example shows is that `kcoreaddons` and presumably the 58 packages that depend on it have no substitutes at `bordeaux.guix.gnu.org`; likewise for `qgpgme` and the 46 packages that depend on it.

Si es una desarrolladora Guix, o si se encuentra a cargo de esta granja de construcción, probablemente quiera inspeccionar estos paquetes con más detalle: simplemente puede que su construcción falle.

`--display-missing`
: Muestra los elementos del almacén para los que faltan las sustituciones.

## 9.16 Invocación de `guix processes`

La orden `guix processes` puede ser útil a desarrolladoras y administradoras de sistemas, especialmente en máquinas multiusuaria y en granjas de construcción: enumera las sesiones actuales (conexiones al daemon), así como información sobre los procesos envueltos[5]. A continuación puede verse un ejemplo de la información que devuelve:

```
$ sudo guix processes
SessionPID: 19002
ClientPID: 19090
ClientCommand: guix shell python

SessionPID: 19402
ClientPID: 19367
ClientCommand: guix publish -u guix-publish -p 3000 -C 9 ...

SessionPID: 19444
ClientPID: 19419
ClientCommand: cuirass --cache-directory /var/cache/cuirass ...
LockHeld: /gnu/store/...-perl-ipc-cmd-0.96.lock
LockHeld: /gnu/store/...-python-six-bootstrap-1.11.0.lock
LockHeld: /gnu/store/...-libjpeg-turbo-2.0.0.lock
ChildPID: 20495
ChildCommand: guix offload x86_64-linux 7200 1 28800
ChildPID: 27733
ChildCommand: guix offload x86_64-linux 7200 1 28800
ChildPID: 27793
ChildCommand: guix offload x86_64-linux 7200 1 28800
```

En este ejemplo vemos que `guix-daemon` tiene tres clientes: `guix shell`, `guix publish` y la herramienta de integración continua Cuirass; sus identificadores de proceso (PID) se muestran en el campo `ClientPID`. El campo `SessionPID` proporciona el PID del subproceso de `guix-daemon` de cada sesión en particular.

The `LockHeld` fields show which store items are currently locked by this session, which corresponds to store items being built or substituted (the `LockHeld` field is not displayed when `guix processes` is not running as root). Last, by looking at the `ChildPID` and `ChildCommand` fields, we understand that these three builds are being offloaded (see Section 2.2.2 [Configuración de delegación del daemon], page 7).

La salida está en formato Recutils por lo que podemos usar la útil orden `recsel` para seleccionar sesiones de interés (see Section "Selection Expressions" in *GNU recutils manual*).

[5] Las sesiones remotas, cuando `guix-daemon` se ha iniciado con `--listen` especificando un punto de conexión TCP, *no* son enumeradas.

Como un ejemplo, la siguiente orden muestra la línea de órdenes y el PID del cliente que inició la construcción de un paquete Perl:

```
$ sudo guix processes | \
    recsel -p ClientPID,ClientCommand -e 'LockHeld ~ "perl"'
ClientPID: 19419
ClientCommand: cuirass --cache-directory /var/cache/cuirass ...
```

Additional options are listed below.

`--format=`*formato*
`-f` *formato*

Produce salida en el *formato* especificado, uno de:

`recutils` The default option. It outputs a set of Session recutils records that include each `ChildProcess` as a field.

`normalized`

Normalize the output records into record sets (see Section "Record Sets" in *GNU recutils manual*). Normalizing into record sets allows joins across record types. The example below lists the PID of each `ChildProcess` and the associated PID for `Session` that spawned the `ChildProcess` where the `Session` was started using `guix build`.

```
$ guix processes --format=normalized | \
    recsel \
    -j Session \
    -t ChildProcess \
    -p Session.PID,PID \
    -e 'Session.ClientCommand ~ "guix build"'
PID: 4435
Session_PID: 4278

PID: 4554
Session_PID: 4278

PID: 4646
Session_PID: 4278
```

# 10 Foreign Architectures

You can target computers of different CPU architectures when producing packages (see Section 5.2 [Invocación de guix package], page 36), packs (see Section 7.3 [Invocación de guix pack], page 92) or full systems (see Section 11.16 [Invocación de guix system], page 634).

GNU Guix supports two distinct mechanisms to target foreign architectures:

1. The traditional cross-compilation (`https://en.wikipedia.org/wiki/Cross_compiler`) mechanism.
2. The native building mechanism which consists in building using the CPU instruction set of the foreign system you are targeting. It often requires emulation, using the QEMU program for instance.

## 10.1 Cross-Compilation

The commands supporting cross-compilation are proposing the `--list-targets` and `--target` options.

The `--list-targets` option lists all the supported targets that can be passed as an argument to `--target`.

```
$ guix build --list-targets
The available targets are:

   - aarch64-linux-gnu
   - arm-linux-gnueabihf
   - avr
   - i586-pc-gnu
   - i686-linux-gnu
   - i686-w64-mingw32
   - mips64el-linux-gnu
   - or1k-elf
   - powerpc-linux-gnu
   - powerpc64le-linux-gnu
   - riscv64-linux-gnu
   - x86_64-linux-gnu
   - x86_64-linux-gnux32
   - x86_64-w64-mingw32
   - xtensa-ath9k-elf
```

Targets are specified as GNU triplets (see Section "Specifying Target Triplets" in *Autoconf*).

Those triplets are passed to GCC and the other underlying compilers possibly involved when building a package, a system image or any other GNU Guix output.

```
$ guix build --target=aarch64-linux-gnu hello
/gnu/store/9926by9qrxa91ijkhw9ndgwp4bn24g9h-hello-2.12

$ file /gnu/store/9926by9qrxa91ijkhw9ndgwp4bn24g9h-hello-2.12/bin/hello
/gnu/store/9926by9qrxa91ijkhw9ndgwp4bn24g9h-hello-2.12/bin/hello: ELF
```

```
64-bit LSB executable, ARM aarch64 ...
```

The major benefit of cross-compilation is that there are no performance penalty compared to emulation using QEMU. There are however higher risks that some packages fail to cross-compile because fewer users are using this mechanism extensively.

## 10.2 Native Builds

The commands that support impersonating a specific system have the `--list-systems` and `--system` options.

The `--list-systems` option lists all the supported systems that can be passed as an argument to `--system`.

```
$ guix build --list-systems
The available systems are:

   - x86_64-linux [current]
   - aarch64-linux
   - armhf-linux
   - i586-gnu
   - i686-linux
   - mips64el-linux
   - powerpc-linux
   - powerpc64le-linux
   - riscv64-linux

$ guix build --system=i686-linux hello
/gnu/store/cc0km35s8x2z4pmwkrqqjx46i8b1i3gm-hello-2.12

$ file /gnu/store/cc0km35s8x2z4pmwkrqqjx46i8b1i3gm-hello-2.12/bin/hello
/gnu/store/cc0km35s8x2z4pmwkrqqjx46i8b1i3gm-hello-2.12/bin/hello: ELF
32-bit LSB executable, Intel 80386 ...
```

In the above example, the current system is *x86_64-linux*. The *hello* package is however built for the *i686-linux* system.

This is possible because the *i686* CPU instruction set is a subset of the *x86_64*, hence *i686* targeting binaries can be run on *x86_64*.

Still in the context of the previous example, if picking the *aarch64-linux* system and the `guix build --system=aarch64-linux hello` has to build some derivations, an extra step might be needed.

The *aarch64-linux* targeting binaries cannot directly be run on a *x86_64-linux* system. An emulation layer is requested. The GNU Guix daemon can take advantage of the Linux kernel binfmt_misc (`https://en.wikipedia.org/wiki/Binfmt_misc`) mechanism for that. In short, the Linux kernel can defer the execution of a binary targeting a foreign platform, here *aarch64-linux*, to a userspace program, usually an emulator.

There is a service that registers QEMU as a backend for the `binfmt_misc` mechanism (see Section 11.10.30 [Servicios de virtualización], page 538). On Debian based foreign distributions, the alternative would be the `qemu-user-static` package.

If the `binfmt_misc` mechanism is not setup correctly, the building will fail this way:

```
$ guix build --system=armhf-linux hello --check
...
 unsupported-platform /gnu/store/jjn969pijv7hff62025yxpfmc8zy0aq0-hello-2.12.drv aarch
while setting up the build environment: a `aarch64-linux' is required to
build `/gnu/store/jjn969pijv7hff62025yxpfmc8zy0aq0-hello-2.12.drv', but
I am a `x86_64-linux'...
```

whereas, with the `binfmt_misc` mechanism correctly linked with QEMU, one can expect to see:

```
$ guix build --system=armhf-linux hello --check
/gnu/store/13xz4nghg39wpymivlwghy08yzj97hlj-hello-2.12
```

The main advantage of native building compared to cross-compiling, is that more packages are likely to build correctly. However it comes at a price: compilation backed by QEMU is *way slower* than cross-compilation, because every instruction needs to be emulated.

The availability of substitutes for the architecture targeted by the `--system` option can mitigate this problem. An other way to work around it is to install GNU Guix on a machine whose CPU supports the targeted instruction set, and set it up as an offload machine (see Section 2.2.2 [Configuración de delegación del daemon], page 7).

# 11 Configuración del sistema

El sistema Guix permite un mecanismo de configuración del sistema completo consistente. Con esto queremos decir que todos los aspectos de la configuración global del sistema—como los servicios disponibles, la zona horaria y la configuración de localización, las cuentas de usuarias—se declaran en un lugar único. Dicha *configuración del sistema* puede ser *instanciada*—es decir, hecha efectiva.

Una de las ventajas de poner toda la configuración del sistema bajo el control de Guix es que permite actualizaciones transaccionales del sistema, y hace posible volver a una instanciación previa del sistema, en caso de que haya algún problema con la nueva (see Section 5.1 [Características], page 35). Otra ventaja es que hace fácil replicar exactamente la misma configuración entre máquinas diferentes, o en diferentes momentos, sin tener que utilizar herramientas de administración adicionales sobre las propias herramientas del sistema.

Esta sección describe este mecanismo. Primero nos enfocaremos en el punto de vista de la administradora del sistema—explicando cómo se configura e instancia el sistema. Después mostraremos cómo puede extenderse este mecanismo, por ejemplo para añadir nuevos servicios del sistema.

## 11.1 Empezando

You're reading this section probably because you have just installed Guix System (see Chapter 3 [Instalación del sistema], page 21) and would like to know where to go from here. If you're already familiar with GNU/Linux system administration, the way Guix System is configured is very different from what you're used to: you won't install a system service by running `guix install`, you won't configure services by modifying files under `/etc`, and you won't create user accounts by invoking `useradd`; instead, all these aspects are spelled out in a *system configuration file*.

The first step with Guix System is thus to write the *system configuration file*; luckily, system installation already generated one for you and stored it under `/etc/config.scm`.

> **Nota:** You can store your system configuration file anywhere you like—it doesn't have to be at `/etc/config.scm`. It's a good idea to keep it under version control, for instance in a Git repository (`https://git-scm.com/book/en/`).

The *entire* configuration of the system—user accounts, system services, timezone, locale settings—is declared in this file, which follows this template:

```
(use-modules (gnu))
(use-package-modules ...)
(use-service-modules ...)

(operating-system
  (host-name ...)
  (timezone ...)
  (locale ...)
  (bootloader ...)
  (file-systems ...)
  (users ...)
```

```
  (packages ...)
  (services ...))
```

This configuration file is in fact a Scheme program; the first lines pull in modules providing variables you might need in the rest of the file—e.g., packages, services, etc. The `operating-system` form declares the system configuration as a *record* with a number of *fields*. See Section 11.2 [Uso de la configuración del sistema], page 244, to view complete examples and learn what to put in there.

The second step, once you have this configuration file, is to test it. Of course, you can skip this step if you're feeling lucky—you choose! To do that, pass your configuration file to `guix system vm` (no need to be root, you can do that as a regular user):

```
guix system vm /etc/config.scm
```

This command returns the name of a shell script that starts a virtual machine (VM) running the system *as described in the configuration file*:

```
/gnu/store/...-run-vm.sh
```

In this VM, you can log in as `root` with no password. That's a good way to check that your configuration file is correct and that it gives the expected result, without touching your system. See Section 11.16 [Invocación de guix system], page 634, for more information.

> **Nota:** When using `guix system vm`, aspects tied to your hardware such as file systems and mapped devices are overridden because they cannot be meaningfully tested in the VM. Other aspects such as static network configuration (see Section 11.10.4 [Networking Setup], page 303) are *not* overridden but they may not work inside the VM.

The third step, once you're happy with your configuration, is to *instantiate* it—make this configuration effective on your system. To do that, run:

```
sudo guix system reconfigure /etc/config.scm
```

This operation is *transactional*: either it succeeds and you end up with an upgraded system, or it fails and nothing has changed. Note that it does *not* restart system services that were already running. Thus, to upgrade those services, you have to reboot or to explicitly restart them; for example, to restart the secure shell (SSH) daemon, you would run:

```
sudo herd restart sshd
```

> **Nota:** System services are managed by the Shepherd (see Section "Jump Start" in *The GNU Shepherd Manual*). The `herd` command lets you inspect, start, and stop services. To view the status of services, run:
>
> ```
> sudo herd status
> ```
>
> To view detailed information about a given service, add its name to the command:
>
> ```
> sudo herd status sshd
> ```
>
> See Section 11.10 [Servicios], page 275, for more information.

The system records its *provenance*—the configuration file and channels that were used to deploy it. You can view it like so:

```
guix system describe
```

Additionally, `guix system reconfigure` preserves previous system generations, which you can list:

```
guix system list-generations
```

Crucially, that means that you can always *roll back* to an earlier generation should something go wrong! When you eventually reboot, you'll notice a sub-menu in the bootloader that reads "Old system generations": it's what allows you to boot *an older generation of your system*, should the latest generation be "broken" or otherwise unsatisfying. You can also "permanently" roll back, like so:

```
sudo guix system roll-back
```

Alternatively, you can use `guix system switch-generation` to switch to a specific generation.

Once in a while, you'll want to delete old generations that you do not need anymore to allow *garbage collection* to free space (see Section 5.6 [Invocación de guix gc], page 53). For example, to remove generations older than 4 months, run:

```
sudo guix system delete-generations 4m
```

From there on, anytime you want to change something in the system configuration, be it adding a user account or changing parameters of a service, you will first update your configuration file and then run `guix system reconfigure` as shown above.

Likewise, to *upgrade* system software, you first fetch an up-to-date Guix and then reconfigure your system with that new Guix:

```
guix pull
sudo guix system reconfigure /etc/config.scm
```

We recommend doing that regularly so that your system includes the latest security updates (see Chapter 19 [Actualizaciones de seguridad], page 723).

**Nota:**

`sudo guix` runs your user's `guix` command and *not* root's, because `sudo` leaves `PATH` unchanged.

The difference matters here, because `guix pull` updates the `guix` command and package definitions only for the user it is run as. This means that if you choose to use `guix system reconfigure` in root's login shell, you'll need to `guix pull` separately.

That's it! If you're getting started with Guix entirely, see Chapter 4 [Empezando], page 32. The next sections dive in more detail into the crux of the matter: system configuration.

## 11.2 Uso de la configuración del sistema

The previous section showed the overall workflow you would follow when administering a Guix System machine (see Section 11.1 [Getting Started with the System], page 242). Let's now see in more detail what goes into the system configuration file.

The operating system is configured by providing an `operating-system` declaration in a file that can then be passed to the `guix system` command (see Section 11.16 [Invocación de guix system], page 634), as we've seen before. A simple setup, with the default Linux-Libre

kernel, initial RAM disk, and a couple of system services added to those provided by default looks like this:

```
;; -*- mode: scheme; -*-
;; This is an operating system configuration template
;; for a "bare bones" setup, with no X11 display server.

(use-modules (gnu))
(use-service-modules networking ssh)
(use-package-modules screen ssh)

(operating-system
  (host-name "komputilo")
  (timezone "Europe/Berlin")
  (locale "en_US.utf8")

  ;; Boot in "legacy" BIOS mode, assuming /dev/sdX is the
  ;; target hard disk, and "my-root" is the label of the target
  ;; root file system.
  (bootloader (bootloader-configuration
               (bootloader grub-bootloader)
               (targets '("/dev/sdX"))))
  ;; It's fitting to support the equally bare bones ‘-nographic’
  ;; QEMU option, which also nicely sidesteps forcing QWERTY.
  (kernel-arguments (list "console=ttyS0,115200"))
  (file-systems (cons (file-system
                        (device (file-system-label "my-root"))
                        (mount-point "/")
                        (type "ext4"))
                      %base-file-systems))

  ;; This is where user accounts are specified.  The "root"
  ;; account is implicit, and is initially created with the
  ;; empty password.
  (users (cons (user-account
                (name "alice")
                (comment "Bob's sister")
                (group "users")

                ;; Adding the account to the "wheel" group
                ;; makes it a sudoer.  Adding it to "audio"
                ;; and "video" allows the user to play sound
                ;; and access the webcam.
                (supplementary-groups '("wheel"
                                        "audio" "video")))
               %base-user-accounts))
```

```
;; Globally-installed packages.
(packages (cons screen %base-packages))

;; Add services to the baseline: a DHCP client and an SSH
;; server.  You may wish to add an NTP service here.
(services (append (list (service dhcp-client-service-type)
                        (service openssh-service-type
                                 (openssh-configuration
                                  (openssh openssh-sans-x)
                                  (port-number 2222))))
                  %base-services)))
```

The configuration is declarative. It is code in the Scheme programming language; the whole `(operating-system ...)` expression produces a *record* with a number of *fields*. Some of the fields defined above, such as `host-name` and `bootloader`, are mandatory. Others, such as `packages` and `services`, can be omitted, in which case they get a default value. See Section 11.3 [Referencia de operating-system], page 253, for details about all the available fields.

Below we discuss the meaning of some of the most important fields.

> **Troubleshooting:** The configuration file is a Scheme program and you might get the syntax or semantics wrong as you get started. Syntactic issues such as misplaced parentheses can often be identified by reformatting your file:
>
> ```
> guix style -f config.scm
> ```
>
> The Cookbook has a short section to get started with the Scheme programming language that explains the fundamentals, which you will find helpful when hacking your configuration. See Section "A Scheme Crash Course" in *GNU Guix Cookbook*.

## Cargador de arranque

El campo `bootloader` describe el método que será usado para arrancar su sistema. Las máquinas basadas en procesadores Intel pueden arrancar en el "obsoleto" modo BIOS, como en el ejemplo previo. No obstante, máquinas más recientes usan la *Interfaz Unificada Extensible de Firmware* (UEFI) para arrancar. En ese caso, el capo `bootloader` debe contener algo parecido a esto:

```
(bootloader-configuration
  (bootloader grub-efi-bootloader)
  (targets '("/boot/efi")))
```

See Section 11.15 [Configuración del gestor de arranque], page 627, para más información sobre las opciones de configuración disponibles.

## Paquetes visibles globalmente

The `packages` field lists packages that will be globally visible on the system, for all user accounts—i.e., in every user's `PATH` environment variable—in addition to the per-user profiles (see Section 5.2 [Invocación de guix package], page 36). The `%base-packages` variable provides all the tools one would expect for basic user and administrator tasks—including the GNU Core Utilities, the GNU Networking Utilities, the `mg` lightweight text editor, `find`,

`grep`, etc. The example above adds GNU Screen to those, taken from the `(gnu packages screen)` module (see Section 8.1 [Módulos de paquetes], page 101). The `(list package output)` syntax can be used to add a specific output of a package:

```
(use-modules (gnu packages))
(use-modules (gnu packages dns))

(operating-system
  ;; ...
  (packages (cons (list isc-bind "utils")
                  %base-packages)))
```

Referring to packages by variable name, like `isc-bind` above, has the advantage of being unambiguous; it also allows typos and such to be diagnosed right away as "unbound variables". The downside is that one needs to know which module defines which package, and to augment the `use-package-modules` line accordingly. To avoid that, one can use the `specification->package` procedure of the `(gnu packages)` module, which returns the best package for a given name or name and version:

```
(use-modules (gnu packages))

(operating-system
  ;; ...
  (packages (append (map specification->package
                         '("tcpdump" "htop" "gnupg@2.0"))
                    %base-packages)))
```

When a package has more than one output it can be a challenge to refer to a specific output instead of just to the standard `out` output. For these situations one can use the `specifications->packages` procedure from the `(gnu packages)` module. For example:

```
(use-modules (gnu packages))

(operating-system
  ;; ...
  (packages (append (specifications->packages
                      '("git" "git:send-email"))
                    %base-packages)))
```

## Servicios del sistema

El campo `services` enumera los *servicios del sistema* disponibles cuando el sistema arranque (see Section 11.10 [Servicios], page 275). La declaración `operating-system` previa especifica que, además de los servicios básicos, queremos que el daemon de shell seguro OpenSSH espere conexiones por el puerto 2222 (see Section 11.10.5 [Servicios de red], page 314). En su implementación, `openssh-service-type` prepara todo para que `sshd` se inicie con las opciones de la línea de órdenes adecuadas, posiblemente generando bajo demanda los archivos de configuración necesarios (see Section 11.19 [Definición de servicios], page 649).

De manera ocasional, en vez de usar los servicios básicos tal y como vienen, puede querer personalizarlos. Para hacerlo, use `modify-services` (see Section 11.19.3 [Referencia de servicios], page 652) para modificar la lista.

Por ejemplo, supongamos que quiere modificar `guix-daemon` y Mingetty (el punto de acceso al sistema por consola) en la lista `%base-services` (see Section 11.10.1 [Servicios base], page 276). Para hacerlo, puede escribir lo siguiente en su declaración de sistema operativo:

```
(define %my-services
  ;; My very own list of services.
  (modify-services %base-services
    (guix-service-type config =>
                       (guix-configuration
                        (inherit config)
                        ;; Fetch substitutes from example.org.
                        (substitute-urls
                          (list "https://example.org/guix"
                                "https://ci.guix.gnu.org"))))
    (mingetty-service-type config =>
                           (mingetty-configuration
                            (inherit config)
                            ;; Automatically log in as "guest".
                            (auto-login "guest")))))

(operating-system
  ;; ...
  (services %mis-servicios))
```

This changes the configuration—i.e., the service parameters—of the `guix-service-type` instance, and that of all the `mingetty-service-type` instances in the `%base-services` list (see Section "Auto-Login to a Specific TTY" in *GNU Guix Cookbook*). Observe how this is accomplished: first, we arrange for the original configuration to be bound to the identifier `config` in the *body*, and then we write the *body* so that it evaluates to the desired configuration. In particular, notice how we use `inherit` to create a new configuration which has the same values as the old configuration, but with a few modifications.

The configuration for a typical "desktop" usage, with an encrypted root partition, a swap file on the root partition, the X11 display server, GNOME and Xfce (users can choose which of these desktop environments to use at the log-in screen by pressing *F1*), network management, power management, and more, would look like this:

```
;; -*- mode: scheme; -*-
;; This is an operating system configuration template
;; for a "desktop" setup with GNOME and Xfce where the
;; root partition is encrypted with LUKS, and a swap file.

(use-modules (gnu) (gnu system nss) (guix utils))
(use-service-modules desktop sddm xorg)
(use-package-modules gnome)

(operating-system
  (host-name "antelope")
  (timezone "Europe/Paris")
```

```
(locale "en_US.utf8")

;; Choose US English keyboard layout.  The "altgr-intl"
;; variant provides dead keys for accented characters.
(keyboard-layout (keyboard-layout "us" "altgr-intl"))

;; Use the UEFI variant of GRUB with the EFI System
;; Partition mounted on /boot/efi.
(bootloader (bootloader-configuration
              (bootloader grub-efi-bootloader)
              (targets '("/boot/efi"))
              (keyboard-layout keyboard-layout)))

;; Specify a mapped device for the encrypted root partition.
;; The UUID is that returned by 'cryptsetup luksUUID'.
(mapped-devices
 (list (mapped-device
        (source (uuid "12345678-1234-1234-1234-123456789abc"))
        (target "my-root")
        (type luks-device-mapping))))

(file-systems (append
               (list (file-system
                       (device (file-system-label "my-root"))
                       (mount-point "/")
                       (type "ext4")
                       (dependencies mapped-devices))
                     (file-system
                       (device (uuid "1234-ABCD" 'fat))
                       (mount-point "/boot/efi")
                       (type "vfat")))
               %base-file-systems))

;; Specify a swap file for the system, which resides on the
;; root file system.
(swap-devices (list (swap-space
                     (target "/swapfile"))))

;; Create user `bob' with `alice' as its initial password.
(users (cons (user-account
              (name "bob")
              (comment "Alice's brother")
              (password (crypt "alice" "$6$abc"))
              (group "students")
              (supplementary-groups '("wheel" "netdev"
                                      "audio" "video")))
             %base-user-accounts))
```

```
;; Add the `students' group
(groups (cons* (user-group
                (name "students"))
               %base-groups))

;; This is where we specify system-wide packages.
(packages (append (list
                   ;; for user mounts
                   gvfs)
                  %base-packages))

;; Add GNOME and Xfce---we can choose at the log-in screen
;; by clicking the gear.  Use the "desktop" services, which
;; include the X11 log-in service, networking with
;; NetworkManager, and more.
(services (if (target-x86-64?)
              (append (list (service gnome-desktop-service-type)
                            (service xfce-desktop-service-type)
                            (set-xorg-configuration
                             (xorg-configuration
                              (keyboard-layout keyboard-layout))))
                      %desktop-services)

              ;; FIXME: Since GDM depends on Rust (gdm -> gnome-shell -> gjs
              ;; -> mozjs -> rust) and Rust is currently unavailable on
              ;; non-x86_64 platforms, we use SDDM and Mate here instead of
              ;; GNOME and GDM.
              (append (list (service mate-desktop-service-type)
                            (service xfce-desktop-service-type)
                            (set-xorg-configuration
                             (xorg-configuration
                              (keyboard-layout keyboard-layout))
                             sddm-service-type))
                      %desktop-services)))

;; Allow resolution of '.local' host names with mDNS.
(name-service-switch %mdns-host-lookup-nss))
```

Un sistema gráfico con una selección de gestores de ventanas ligeros en vez de entornos de escritorio completos podría ser así:

```
;; -*- mode: scheme; -*-
;; This is an operating system configuration template
;; for a "desktop" setup without full-blown desktop
;; environments.

(use-modules (gnu) (gnu system nss))
```

```
(use-service-modules desktop)
(use-package-modules bootloaders emacs emacs-xyz ratpoison suckless wm
                     xorg)

(operating-system
  (host-name "antelope")
  (timezone "Europe/Paris")
  (locale "en_US.utf8")

  ;; Use the UEFI variant of GRUB with the EFI System
  ;; Partition mounted on /boot/efi.
  (bootloader (bootloader-configuration
               (bootloader grub-efi-bootloader)
               (targets '("/boot/efi"))))

  ;; Assume the target root file system is labelled "my-root",
  ;; and the EFI System Partition has UUID 1234-ABCD.
  (file-systems (append
                 (list (file-system
                         (device (file-system-label "my-root"))
                         (mount-point "/")
                         (type "ext4"))
                       (file-system
                         (device (uuid "1234-ABCD" 'fat))
                         (mount-point "/boot/efi")
                         (type "vfat")))
                 %base-file-systems))

  (users (cons (user-account
                (name "alice")
                (comment "Bob's sister")
                (group "users")
                (supplementary-groups '("wheel" "netdev"
                                        "audio" "video")))
               %base-user-accounts))

  ;; Add a bunch of window managers; we can choose one at
  ;; the log-in screen with F1.
  (packages (append (list
                     ;; window managers
                     ratpoison i3-wm i3status dmenu
                     emacs emacs-exwm emacs-desktop-environment
                     ;; terminal emulator
                     xterm)
                    %base-packages))

  ;; Use the "desktop" services, which include the X11
```

```
;; log-in service, networking with NetworkManager, and more.
(services %desktop-services)

;; Allow resolution of '.local' host names with mDNS.
(name-service-switch %mdns-host-lookup-nss))
```

Este ejemplo se refiere al sistema de archivos `/boot/efi` por su UUID `1234-ABCD`. Substituya este UUID con el UUID correcto en su sistema, como el devuelto por la orden `blkid`.

See Section 11.10.9 [Servicios de escritorio], page 363, for the exact list of services provided by `%desktop-services`.

De nuevo, `%desktop-services` es simplemente una lista de objetos de servicios. Si desea borrar servicios de aquí, puede hacerlo usando procedimientos de filtrado de listas (see Section "SRFI-1 Filtering and Partitioning" in *GNU Guile Reference Manual*). Por ejemplo, la siguiente expresión devuelve una lista que contiene todos los servicios en `%desktop-services` excepto el servicio Avahi:

```
(remove (lambda (service)
          (eq? (service-kind service) avahi-service-type))
        %desktop-services)
```

Alternatively, the `modify-services` macro can be used:

```
(modify-services %desktop-services
  (delete avahi-service-type))
```

## Inspecting Services

As you work on your system configuration, you might wonder why some system service doesn't show up or why the system is not as you expected. There are several ways to inspect and troubleshoot problems.

First, you can inspect the dependency graph of Shepherd services like so:

```
guix system shepherd-graph /etc/config.scm | \
  guix shell xdot -- xdot -
```

This lets you visualize the Shepherd services as defined in `/etc/config.scm`. Each box is a service as would be shown by `sudo herd status` on the running system, and each arrow denotes a dependency (in the sense that if service *A* depends on *B*, then *B* must be started before *A*).

Not all "services" are Shepherd services though, since Guix System uses a broader definition of the term (see Section 11.10 [Servicios], page 275). To visualize system services and their relations at a higher level, run:

```
guix system extension-graph /etc/config.scm | \
  guix shell xdot -- xdot -
```

This lets you view the *service extension graph*: how services "extend" each other, for instance by contributing to their configuration. See Section 11.19.1 [Composición de servicios], page 649, to understand the meaning of this graph.

Last, you may also find it useful to inspect your system configuration at the REPL (see Section 8.14 [Using Guix Interactively], page 178). Here is an example session:

```
$ guix repl
```

```
scheme@(guix-user)> ,use (gnu)
scheme@(guix-user)> (define os (load "config.scm"))
scheme@(guix-user)> ,pp (map service-kind (operating-system-services os))
$1 = (#<service-type localed cabba93>
      ...)
```

See Section 11.19.3 [Referencia de servicios], page 652, to learn about the Scheme interface to manipulate and inspect services.

### Instanciación del sistema

Assuming the `operating-system` declaration is stored in the `config.scm` file, the `sudo guix system reconfigure config.scm` command instantiates that configuration, and makes it the default boot entry. See Section 11.1 [Getting Started with the System], page 242, for an overview.

La manera habitual de cambiar la configuración del sistema es actualizar este archivo y volver a ejecutar `guix system reconfigure`. Nunca se deberían tocar los archivos en `/etc` o ejecutar órdenes que modifiquen el estado del sistema como `useradd` o `grub-install`. De hecho, debe evitarlo ya que no únicamente anularía su garantía sino que también le impediría volver a una versión previa de su sistema, en caso de necesitarlo.

### La interfaz programática

A nivel Scheme, el grueso de una declaración `operating-system` se instancia con el siguiente procedimiento monádico (see Section 8.11 [La mónada del almacén], page 162):

`operating-system-derivation` *so* [Procedimiento monádico]

Devuelve una derivación que construye *so*, un objeto `operating-system` (see Section 8.10 [Derivaciones], page 159).

La salida de la derivación es un único directorio que hace referencia a todos los paquetes, archivos de configuración y otros archivos auxiliares necesarios para instanciar *so*.

Este procedimiento se proporciona por el módulo `(gnu system)`. Junto con `(gnu services)` (see Section 11.10 [Servicios], page 275), este módulo contiene los entresijos del sistema Guix. ¡Asegúrese de echarle un vistazo!

## 11.3 Referencia de `operating-system`

Esta sección resume todas las opciones disponibles en las declaraciones de `operating-system` (see Section 11.2 [Uso de la configuración del sistema], page 244).

`operating-system` [Tipo de datos]

Este es el tipo de datos que representa la configuración del sistema operativo. Con ello queremos decir toda la configuración global del sistema, no la configuración específica de las usuarias (see Section 11.2 [Uso de la configuración del sistema], page 244).

`kernel` (predeterminado: `linux-libre`)

El objeto del paquete del núcleo del sistema operativo usado[1].

[1] Actualmente únicamente está completamente implementado el núcleo Linux-libre. El uso de GNU mach con GNU Hurd es experimental y únicamente está disponible cuando se construye una imagen de disco para máquina virtual.

`hurd` (predeterminado: `#f`)
: El objeto del paquete de Hurd iniciado por el núcleo. Cuando se proporciona este campo, produce un sistema operativo GNU/Hurd. En ese caso, `kernel` también debe contener el paquete `gnumach`—el micronúcleo sobre el que se ejecuta Hurd.

  **Aviso:** Esta característica es experimental y únicamente está implementada para imágenes de disco.

`kernel-loadable-modules` (predeterminados: `'()`)
: Una lista de objetos (habitualmente paquetes) desde los que se obtendrán los módulos del núcleo–por ejemplo `(list ddcci-driver-linux)`.

`kernel-arguments` (predeterminados: `%default-kernel-arguments`)
: Lista de cadenas o expresiones-G que representan parámetros adicionales a pasar en la línea de órdenes del núcleo—por ejemplo, `("console=ttyS0")`.

`bootloader`
: El objeto de configuración del cargador de arranque del sistema. See Section 11.15 [Configuración del gestor de arranque], page 627.

`label` Es una etiqueta (una cadena) con la que aparecerá en el menú del cargador de arranque. La etiqueta predeterminada incluye el nombre y la versión del núcleo.

`keyboard-layout` (predeterminada: `#f`)
: This field specifies the keyboard layout to use in the console. It can be either `#f`, in which case the default keyboard layout is used (usually US English), or a `<keyboard-layout>` record. See Section 11.8 [Distribución de teclado], page 271, for more information.

  Esta distribución de teclado se hace efectiva tan pronto el núcleo haya arrancado. Por ejemplo, la distribución de teclado está en efecto cuando introduzca una contraseña si su sistema de archivos raíz se encuentra en un dispositivo traducido `luks-device-mapping` (see Section 11.5 [Dispositivos traducidos], page 263).

  **Nota:** Esto *no* especifica la distribución de teclado usada por el cargador de arranque, ni tampoco la usada por el servidor gráfico. See Section 11.15 [Configuración del gestor de arranque], page 627, para información sobre cómo especificar la distribución de teclado del cargador de arranque. See Section 11.10.7 [Sistema X Window], page 340, para información sobre cómo especificar la distribución de teclado usada por el sistema de ventanas X.

`initrd-modules` (predeterminados: `%base-initrd-modules`)
: La lista de módulos del núcleo Linux que deben estar disponibles en el disco inicial en RAM. See Section 11.14 [Disco en RAM inicial], page 624.

`initrd` (predeterminado: `base-initrd`)
: Un procedimiento que devuelve un disco inicial en RAM para el núcleo Linux. Este campo se proporciona para permitir personalizaciones de

bajo nivel y no debería ser necesario para un uso habitual. See Section 11.14 [Disco en RAM inicial], page 624.

`firmware` (predeterminado: `%base-firmware`)
: Lista de paquetes de firmware que pueden ser cargados por el núcleo del sistema operativo.

  El valor predeterminado incluye el firmware necesario para dispositivos WiFi basados en Atheros y Broadcom (módulos Linux-libre `ath9k` y `b43-open`, respectivamente). See Section 3.2 [Consideraciones sobre el hardware], page 21, para más información sobre hardware soportado.

`host-name`
: El nombre de la máquina.

`mapped-devices` (predeterminados: `'()`)
: Una lista de dispositivos traducidos. See Section 11.5 [Dispositivos traducidos], page 263.

`file-systems`
: Una lista de sistemas de archivos. See Section 11.4 [Sistemas de archivos], page 257.

`swap-devices` (predeterminados: `'()`)
: A list of swap spaces. See Section 11.6 [Swap Space], page 265.

`users` (predeterminadas: `%base-user-accounts`)
`groups` (predeterminados: `%base-groups`)
: Lista de cuentas de usuaria y grupos. See Section 11.7 [Cuentas de usuaria], page 268.

  Si la lista de `usuarias` carece de una cuenta de usuaria con UID 0, una cuenta "root" con UID 0 se añade automáticamente.

`skeletons` (predeterminados: `(default-skeletons)`)
: Una lista de tuplas de nombre de archivo de destino/objeto tipo-archivo (see Section 8.12 [Expresiones-G], page 167). Estos son los archivos de esqueleto que se añadirán al directorio de las cuentas de usuaria que se creen.

  Por ejemplo, un valor válido puede parecer algo así:

```
`((".bashrc" ,(plain-file "bashrc" "echo Hola\n"))
  (".guile" ,(plain-file "guile"
                         "(use-modules (ice-9 readline))
                          (activate-readline)")))
```

`issue` (predeterminado: `%default-issue`)
: Una cadena que denota el contenido del archivo `/etc/issue`, que se muestra cuando las usuarias ingresan al sistema en una consola de texto.

`packages` (predeterminados: `%base-packages`)
: Una lista de paquetes instalados en el perfil global, que es accesible en `/run/current-system/profile`. Cada elemento debe ser una variable

de paquete o una tupla paquete/salida. A continuación se muestra un ejemplo de ambos tipos:

```
(cons* git                      ; la salida predeterminada "out"
       (list git "send-email") ; otra salida de git
       %base-packages)         ; el conjunto predeterminado
```

El conjunto predeterminado incluye utilidades básicas y es una buena práctica instalar utilidades no-básicas en los perfiles de las usuarias (see Section 5.2 [Invocación de guix package], page 36).

`timezone` (default: `"Etc/UTC"`)
: Una cadena que identifica la zona horaria—por ejemplo, `"Europe/Paris"`.

  Puede ejecutar la orden `tzselect` para encontrar qué cadena de zona horaria corresponde con su región. Elegir una zona horaria no válida provoca un fallo en `guix system`.

`locale` (predeterminado: `"en_US.utf8"`)
: El nombre de la localización predeterminada (see Section "Locale Names" in *The GNU C Library Reference Manual*). See Section 11.9 [Localizaciones], page 273, para más información.

`locale-definitions` (predeterminadas: `%default-locale-definitions`)
: La lista de definiciones de localizaciones a compilar y que puede ser usada en tiempo de ejecución. See Section 11.9 [Localizaciones], page 273.

`locale-libcs` (predeterminadas: `(list` *`glibc`*`)`)
: La lista de paquetes GNU libc cuyos datos de localización y herramientas son usadas para las definiciones de localizaciones. See Section 11.9 [Localizaciones], page 273, para consideraciones de compatibilidad que justifican esta opción.

`name-service-switch` (predeterminado: `%default-nss`)
: Configuración del selector de servicios de nombres de libc (NSS)—un objeto `<name-service-switch>`. See Section 11.13 [Selector de servicios de nombres], page 622, para detalles.

`services` (predeterminados: `%base-services`)
: Una lista de objetos service denotando los servicios del sistema. See Section 11.10 [Servicios], page 275.

`essential-services` (predeterminados: ...)
: The list of "essential services"—i.e., things like instances of `system-service-type` (see Section 11.19.3 [Referencia de servicios], page 652) and `host-name-service-type`, which are derived from the operating system definition itself. As a user you should *never* need to touch this field.

`pam-services` (predeterminados: `(base-pam-services)`)
: Servicios de los *módulos de identificación conectables* (PAM) de Linux.

`setuid-programs` (predeterminados: `%setuid-programs`)
: List of `<setuid-program>`. See Section 11.11 [Programas con setuid], page 620, for more information.

`sudoers-file` (predeterminado: `%sudoers-specification`)
: El contenido de `/etc/sudoers` como un objeto tipo-archivo (see Section 8.12 [Expresiones-G], page 167).

  Este archivo especifica qué usuarias pueden usar la orden `sudo`, lo que se les permite hacer y qué privilegios pueden obtener. El comportamiento predefinido es que únicamente `root` y los miembros del grupo `wheel` pueden usar `sudo`.

`this-operating-system` [Macro]
: Cuando se usa en el *ámbito léxico* de un campo de una definición de sistema operativo, este identificador está enlazado al sistema operativo en definición.

  El siguiente ejemplo muestra cómo hacer referencia al sistema operativo en definición en la definición del campo `label`:

```
(use-modules (gnu) (guix))

(operating-system
  ;; ...
  (label (package-full-name
          (operating-system-kernel this-operating-system))))
```

  Es un error hacer referencia a `this-operating-system` fuera de una definición de sistema operativo.

## 11.4 Sistemas de archivos

La lista de sistemas de archivos que deben montarse se especifica en el campo `file-systems` de la declaración del sistema operativo (see Section 11.2 [Uso de la configuración del sistema], page 244). Cada sistema de archivos se declara usando la forma `file-system`, como en el siguiente ejemplo:

```
(file-system
  (mount-point "/home")
  (device "/dev/sda3")
  (type "ext4"))
```

Como es habitual, algunos de los campos son obligatorios—aquellos mostrados en el ejemplo previo—mientras que otros pueden omitirse. Se describen a continuación.

`file-system` [Tipo de datos]
: Objetos de este tipo representan los sistemas de archivos a montar. Contienen los siguientes campos:

  `type` Este campo es una cadena que especifica el tipo de sistema de archivos—por ejemplo, `"ext4"`.

  `mount-point`
  : Designa la ruta donde el sistema de archivos debe montarse.

  `device` Nombra la "fuente" del sistema de archivos. Puede ser una de estas tres opciones: una etiqueta de sistema de archivos, un UUID de sistema de archivos o el nombre de un nodo `/dev`. Las etiquetas y UUID ofrecen una

forma de hacer referencia a sistemas de archivos sin codificar su nombre de dispositivo actual[2].

Las etiquetas del sistema de archivos se crean mediante el uso del procedimiento `file-system-label`, los UUID se crean mediante el uso de `uuid` y los nodos `/dev` son simples cadenas. A continuación se proporciona un ejemplo de un sistema de archivos al que se hace referencia mediante su etiqueta, como es mostrada por la orden `e2label`:

```
(file-system
  (mount-point "/home")
  (type "ext4")
  (device (file-system-label "mi-home")))
```

Los UUID se convierten dede su representación en forma de cadena (como se muestra con la orden `tune2fs -l`) mediante el uso de la forma `uuid`[3], como sigue:

```
(file-system
  (mount-point "/home")
  (type "ext4")
  (device (uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")))
```

Cuando la fuente de un sistema de archivos es un dispositivo traducido (see Section 11.5 [Dispositivos traducidos], page 263), su campo `device` *debe* hacer referencia al nombre del dispositivo traducido—por ejemplo, `"/dev/mapper/particion-raiz"`. Esto es necesario para que el sistema sepa que el montaje del sistema de archivos depende del establecimiento de la traducción de dispositivos correspondiente.

`flags` (predeterminadas: `'()`)
: This is a list of symbols denoting mount flags. Recognized flags include `read-only`, `bind-mount`, `no-dev` (disallow access to special files), `no-suid` (ignore setuid and setgid bits), `no-atime` (do not update file access times), `no-diratime` (likewise for directories only), `strict-atime` (update file access time), `lazy-time` (only update time on the in-memory version of the file inode), `no-exec` (disallow program execution), and `shared` (make the mount shared). See Section "Mount-Unmount-Remount" in *The GNU C Library Reference Manual*, for more information on these flags.

`options` (predeterminadas: `#f`)
: This is either `#f`, or a string denoting mount options passed to the file system driver. See Section "Mount-Unmount-Remount" in *The GNU C Library Reference Manual*, for details.

---

[2] Fíjese que, aunque es tentador usa `/dev/disk/by-uuid` y nombres de dispositivo similares para obtener el mismo resultado, no es lo recomendado: estos nodo especiales de dispositivos se crean por el daemon udev y puede no estar disponible cuando el dispositivo sea montado.

[3] La forma `uuid` espera un UUID de 16 bytes como se define en la RFC 4122 (`https://tools.ietf.org/html/rfc4122`). Este es el formato de UUID que usan la familia de sistemas de archivos ext2 y otros, pero es diferente de los "UUID" de los sistemas de archivos FAT, por ejemplo.

Run `man 8 mount` for options for various file systems, but beware that what it lists as file-system-independent "mount options" are in fact flags, and belong in the `flags` field described above.

The `file-system-options->alist` and `alist->file-system-options` procedures from `(gnu system file-systems)` can be used to convert file system options given as an association list to the string representation, and vice-versa.

`mount?` (predeterminado: `#t`)
Este valor indica si debe montarse el sistema de archivos automáticamente al iniciar el sistema. Cuando se establece como `#f`, el sistema de archivos tiene una entrada en `/etc/fstab` (el cual es leído por la orden `mount`) pero no se montará automáticamente.

`needed-for-boot?` (predeterminado: `#f`)
Este valor lógico indica si el sistema de archivos es necesario para el arranque. Si es verdadero, el sistema de archivos se monta al cargar el disco inicial en RAM (initrd). Este es siempre el caso, por ejemplo, para el sistema de archivos raíz.

`check?` (predeterminado: `#t`)
This Boolean indicates whether the file system should be checked for errors before being mounted. How and when this happens can be further adjusted with the following options.

`skip-check-if-clean?` (default: `#t`)
When true, this Boolean indicates that a file system check triggered by `check?` may exit early if the file system is marked as "clean", meaning that it was previously correctly unmounted and should not contain errors.

Setting this to false will always force a full consistency check when `check?` is true. This may take a very long time and is not recommended on healthy systems—in fact, it may reduce reliability!

Conversely, some primitive file systems like `fat` do not keep track of clean shutdowns and will perform a full scan regardless of the value of this option.

`repair` (default: `'preen`)
When `check?` finds errors, it can (try to) repair them and continue booting. This option controls when and how to do so.

If false, try not to modify the file system at all. Checking certain file systems like `jfs` may still write to the device to replay the journal. No repairs will be attempted.

If `#t`, try to repair any errors found and assume "yes" to all questions. This will fix the most errors, but may be risky.

If `'preen`, repair only errors that are safe to fix without human interaction. What that means is left up to the developers of each file system and may be equivalent to "none" or "all".

`create-mount-point?` (predeterminado: `#f`)
: Cuando es verdadero, el punto de montaje es creado si no existía previamente.

`mount-may-fail?` (predeterminado: `#f`)
: Cuando tiene valor verdadero indica que el montaje de este sistema de archivos puede fallar pero no debe considerarse un error. Es útil en casos poco habituales; un ejemplo de esto es `efivarfs`, un sistema de archivos que únicamente puede montarse en sistemas EFI/UEFI.

`dependencies` (predeterminadas: `'()`)
: Una lista de objetos `<file-system>` o `<mapped-device>` que representan sistemas de archivos que deben montarse o dispositivos traducidos que se deben abrir antes (y desmontar o cerrar después) que el declarado.

  Como ejemplo, considere la siguiente jerarquía de montajes: `/sys/fs/cgroup` es una dependencia de `/sys/fs/cgroup/cpu` y `/sys/fs/cgroup/memory`.

  Otro ejemplo es un sistema de archivos que depende de un dispositivo traducido, por ejemplo una partición cifrada (see Section 11.5 [Dispositivos traducidos], page 263).

`file-system-label` *str* [Procedimiento]
Este procedimiento devuelve un objeto opaco de etiqueta del sistema de archivos a partir de *str*, una cadena:

```
(file-system-label "home")
⇒ #<file-system-label "home">
```

Las etiquetas del sistema de archivos se usan para hacer referencia a los sistemas de archivos por etiqueta en vez de por nombre de dispositivo. Puede haber encontrado previamente ejemplos en el texto.

El módulo `(gnu system file-systems)` exporta las siguientes variables útiles.

`%base-file-systems` [Variable]
Estos son los sistemas de archivos esenciales que se necesitan en sistemas normales, como `%pseudo-terminal-file-system` y `%immutable-store` (véase a continuación). Las declaraciones de sistemas operativos deben contener siempre estos al menos.

`%pseudo-terminal-file-system` [Variable]
El sistema de archivos que debe montarse como `/dev/pts`. Permite la creación de *pseudoterminales* a través de `openpty` y funciones similares (see Section "Pseudo-Terminals" in *The GNU C Library Reference Manual*). Los pseudoterminales son usados por emuladores de terminales como `xterm`.

`%shared-memory-file-system` [Variable]
Este sistema de archivos se monta como `/dev/shm` y se usa para permitir el uso de memoria compartida entre procesos (see Section "Memory-mapped I/O" in *The GNU C Library Reference Manual*).

`%immutable-store` [Variable]
Este sistema de archivos crea un montaje enlazado ("bind-mount") de `/gnu/store`, permitiendo solo el acceso de lectura para todas las usuarias incluyendo a `root`. Esto previene modificaciones accidentales por software que se ejecuta como `root` o por las administradoras del sistema.

El daemon sí es capaz de escribir en el almacén: vuelve a montar `/gnu/store` en modo lectura-escritura en su propio "espacio de nombres".

`%binary-format-file-system` [Variable]
El sistema de archivos `binfmt_misc`, que permite que el manejo de tipos de archivos ejecutables arbitrarios se delegue al espacio de usuaria. Necesita la carga del módulo del núcleo `binfmt.ko`.

`%fuse-control-file-system` [Variable]
El sistema de archivos `fusectl`, que permite a usuarias sin privilegios montar y desmontar sistemas de archivos de espacio de usuaria FUSE. Necesita la carga del módulo del núcleo `fuse.ko`.

El módulo `(gnu system uuid)` proporciona herramientas para tratar con "identificadores únicos" de sistemas de archivos (UUID).

`uuid` *str* [*tipo*] [Procedimiento]
Devuelve un objeto opaco de UUID (identificador único) del *tipo* (un símbolo) procesando *str* (una cadena):

```
(uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")
⇒ #<<uuid> type: dce bv: ...>

(uuid "1234-ABCD" 'fat)
⇒ #<<uuid> type: fat bv: ...>
```

*tipo* puede ser `dce`, `iso9660`, `fat`, `ntfs`, o uno de sus sinónimos habitualmente usados para estos tipos.

Los UUID son otra forma de hacer referencia de forma inequívoca a sistemas de archivos en la configuración de sistema operativo. Puede haber encontrado previamente ejemplos en el texto.

### 11.4.1 Sistema de archivos Btrfs

El sistema de archivos Btrfs tiene características especiales, como los subvolúmenes, que merecen una explicación más detallada. La siguiente sección intenta cubrir usos básicos así como usos complejos del sistema de archivos Btrfs con el sistema Guix.

Con el uso más simple se puede describir un sistema de archivos Btrfs puede describirse, por ejemplo, del siguiente modo:

```
(file-system
  (mount-point "/home")
  (type "btrfs")
  (device (file-system-label "mi-home")))
```

El ejemplo siguiente es más complejo, ya que usa un subvolumen de Btrfs, llamado `rootfs`. El sistema de archivos tiene la etiqueta `mi-btrfs`, y se encuentra en un dispositivo cifrado (de aquí la dependencia de `mapped-devices`):

```
(file-system
  (device (file-system-label "mi-btrfs"))
  (mount-point "/")
  (type "btrfs")
  (options "subvol=rootfs")
  (dependencies mapped-devices))
```

Algunos cargadores de arranque, por ejemplo GRUB, únicamente montan una partición Btrfs en su nivel superior durante los momentos iniciales del arranque, y dependen de que su configuración haga referencia a la ruta correcta del subvolumen dentro de dicho nivel superior. Los cargadores de arranque que operan de este modo producen habitualmente su configuración en un sistema en ejecución donde las particiones Btrfs ya se encuentran montadas y donde la información de subvolúmenes está disponible. Como un ejemplo, `grub-mkconfig`, la herramienta de generación de configuración que viene con GRUB, lee `/proc/self/mountinfo` para determinar la ruta desde el nivel superior de un subvolumen.

El sistema Guix produce una configuración para el cargador de arranque usando la configuración del sistema operativo como su única entrada; por lo tanto es necesario extraer la información del subvolumen en el que se encuentra `/gnu/store` (en caso de estar en alguno) de la configuración del sistema operativo. Para ilustrar esta situación mejor, considere un subvolumen que se llama 'rootfs' el cual contiene el sistema de archivos raiz. En esta situación, el cargador de arranque GRUB únicamente vería el nivel superior de la partición de raíz de Btrfs, por ejemplo:

```
/                   (nivel superior)
 rootfs           (directorio del subvolumen)
     gnu          (directorio normal)
         store    (directorio normal)
[...]
```

Por lo tanto, el nombre del subvolumen debe añadirse al inicio de la ruta al núcleo, los binarios de initrd y otros archivos a los que haga referencia la configuración de GRUB en `/gnu/store`, para que puedan encontrarse en los momentos iniciales del arranque.

El siguiente ejemplo muestra una jerarquía anidada de subvolúmenes y directorios:

```
/                   (nivel superior)
 rootfs           (subvolumen)
     gnu          (directorio normal)
         store    (subvolumen)
[...]
```

Este escenario funcionaría sin montar el subvolumen 'store'. Montar 'rootfs' es suficiente, puesto que el nombre del subvolumen corresponde con el punto de montaje deseado en la jerarquía del sistema de archivos. Alternativamente se puede hacer referencia el subvolumen 'store' proporcionando tanto el valor `/rootfs/gnu/store` como el valor `rootfs/gnu/store` a la opción `subvol`.

Por último, un ejemplo más elaborado de subvolúmenes anidados:

```
/                           (nivel superior)
```

```
 root-snapshots          (subvolumen)
     root-current        (subvolumen)
         guix-store      (subvolumen)
[...]
```

Aquí, el subvolumen 'guix-store' no corresponde con el punto de montaje deseado, por lo que es necesario montarlo. El subvolumen debe ser especificado completamente proporcionando su nombre de archivo a la opción `subvol`. Para ilustrar este ejemplo, el subvolumen 'guix-store' puede montarse en `/gnu/store` usando una declaración de sistema de archivos como la siguiente:

```
(file-system
  (device (file-system-label "mi-otro-btrfs"))
  (mount-point "/gnu/store")
  (type "btrfs")
  (options "subvol=root-snapshots/root-current/guix-store,\
compress-force=zstd,space_cache=v2"))
```

## 11.5 Dispositivos traducidos

The Linux kernel has a notion of *device mapping*: a block device, such as a hard disk partition, can be *mapped* into another device, usually in `/dev/mapper/`, with additional processing over the data that flows through it[4]. A typical example is encryption device mapping: all writes to the mapped device are encrypted, and all reads are deciphered, transparently. Guix extends this notion by considering any device or set of devices that are *transformed* in some way to create a new device; for instance, RAID devices are obtained by *assembling* several other devices, such as hard disks or partitions, into a new one that behaves as one partition.

Los dispositivos traducidos se declaran mediante el uso de la forma `mapped-device`, definida a continuación; ejemplos más adelante.

`mapped-device` [Tipo de datos]

Objetos de este tipo representan traducciones de dispositivo que se llevarán a cabo cuando el sistema arranque.

`source` This is either a string specifying the name of the block device to be mapped, such as `"/dev/sda3"`, or a list of such strings when several devices need to be assembled for creating a new one. In case of LVM this is a string specifying name of the volume group to be mapped.

`target` This string specifies the name of the resulting mapped device. For kernel mappers such as encrypted devices of type `luks-device-mapping`, specifying `"my-partition"` leads to the creation of the `"/dev/mapper/my-partition"` device. For RAID devices of type `raid-device-mapping`, the full device name such as `"/dev/md0"` needs to be given. LVM logical

[4] Note that the GNU Hurd makes no difference between the concept of a "mapped device" and that of a file system: both boil down to *translating* input/output operations made on a file to operations on its backing store. Thus, the Hurd implements mapped devices, like file systems, using the generic *translator* mechanism (see Section "Translators" in *The GNU Hurd Reference Manual*).

volumes of type `lvm-device-mapping` need to be specified as `"VGNAME-LVNAME"`.

`targets` This list of strings specifies names of the resulting mapped devices in case there are several. The format is identical to *target*.

`type` Debe ser un objeto `mapped-device-kind`, que especifica cómo *source* se traduce a *target*.

`luks-device-mapping` [Variable]
Define el cifrado de bloques LUKS mediante el uso de la orden `cryptsetup` del paquete del mismo nombre. Depende del módulo `dm-crypt` del núcleo Linux.

`luks-device-mapping-with-options` [*#:key-file*] [Procedure]
Return a `luks-device-mapping` object, which defines LUKS block device encryption using the `cryptsetup` command from the package with the same name. It relies on the `dm-crypt` Linux kernel module.

If `key-file` is provided, unlocking is first attempted using that key file. This has an advantage of not requiring a password entry, so it can be used (for example) to unlock RAID arrays automatically on boot. If key file unlock fails, password unlock is attempted as well. Key file is not stored in the store and needs to be available at the given location at the time of the unlock attempt.

```
;; Following definition would be equivalent to running:
;;   cryptsetup open --key-file /crypto.key /dev/sdb1 data
(mapped-device
 (source "/dev/sdb1)
 (target "data)
 (type (luks-device-mapping-with-options
        #:key-file "/crypto.key")))
```

`raid-device-mapping` [Variable]
Define un dispositivo RAID, el cual se ensambla mediante el uso de la orden `mdadm` del paquete del mismo nombre. Requiere la carga del módulo del núcleo Linux para el nivel RAID apropiado, como `raid456` para RAID-4, RAID-5 o RAID-6, o `raid10` para RAID-10.

`lvm-device-mapping` [Variable]
This defines one or more logical volumes for the Linux Logical Volume Manager (LVM) (`https://www.sourceware.org/lvm2/`). The volume group is activated by the `vgchange` command from the `lvm2` package.

El siguiente ejemplo especifica una traducción de `/dev/sda3` a `/dev/mapper/home` mediante el uso de LUKS—la configuración de claves unificada de Linux (`https://gitlab.com/cryptsetup/cryptsetup`), un mecanismo estándar para cifrado de disco. El dispositivo `/dev/mapper/home` puede usarse entonces como el campo `device` de una declaración `file-system` (see Section 11.4 [Sistemas de archivos], page 257).

```
(mapped-device
  (source "/dev/sda3")
  (target "home")
```

```
  (type luks-device-mapping))
```

De manera alternativa, para independizarse de la numeración de dispositivos, puede obtenerse el UUID LUKS (*identificador único*) del dispositivo fuente con una orden así:

```
cryptsetup luksUUID /dev/sda3
```

y usarlo como sigue:

```
(mapped-device
  (source (uuid "cb67fc72-0d54-4c88-9d4b-b225f30b0f44"))
  (target "home")
  (type luks-device-mapping))
```

It is also desirable to encrypt swap space, since swap space may contain sensitive data. One way to accomplish that is to use a swap file in a file system on a device mapped via LUKS encryption. In this way, the swap file is encrypted because the entire device is encrypted. See Section 11.6 [Swap Space], page 265, or See Section 3.4 [Disk Partitioning], page 23, for an example.

Un dispositivo RAID formado por las particiones `/dev/sda1` y `/dev/sdb1` puede declararse como se muestra a continuación:

```
(mapped-device
  (source (list "/dev/sda1" "/dev/sdb1"))
  (target "/dev/md0")
  (type raid-device-mapping))
```

El dispositivo `/dev/md0` puede usarse entonces como el campo `device` de una declaración `file-system` (see Section 11.4 [Sistemas de archivos], page 257). Fíjese que no necesita proporcionar el nivel RAID; se selecciona durante la creación inicial y formato del dispositivo RAID y después se determina automáticamente.

LVM logical volumes “alpha” and “beta” from volume group “vg0” can be declared as follows:

```
(mapped-device
  (source "vg0")
  (targets (list "vg0-alpha" "vg0-beta"))
  (type lvm-device-mapping))
```

Devices `/dev/mapper/vg0-alpha` and `/dev/mapper/vg0-beta` can then be used as the `device` of a `file-system` declaration (see Section 11.4 [Sistemas de archivos], page 257).

## 11.6 Swap Space

Swap space, as it is commonly called, is a disk area specifically designated for paging: the process in charge of memory management (the Linux kernel or Hurd's default pager) can decide that some memory pages stored in RAM which belong to a running program but are unused should be stored on disk instead. It unloads those from the RAM, freeing up precious fast memory, and writes them to the swap space. If the program tries to access that very page, the memory management process loads it back into memory for the program to use.

A common misconception about swap is that it is only useful when small amounts of RAM are available to the system. However, it should be noted that kernels often use all

available RAM for disk access caching to make I/O faster, and thus paging out unused portions of program memory will expand the RAM available for such caching.

For a more detailed description of how memory is managed from the viewpoint of a monolithic kernel, see Section "Memory Concepts" in *The GNU C Library Reference Manual*.

The Linux kernel has support for swap partitions and swap files: the former uses a whole disk partition for paging, whereas the second uses a file on a file system for that (the file system driver needs to support it). On a comparable setup, both have the same performance, so one should consider ease of use when deciding between them. Partitions are "simpler" and do not need file system support, but need to be allocated at disk formatting time (logical volumes notwithstanding), whereas files can be allocated and deallocated at any time.

Swap space is also required to put the system into *hibernation* (also called *suspend to disk*), whereby memory is dumped to swap before shutdown so it can be restored when the machine is eventually restarted. Hibernation uses at most half the size of the RAM in the configured swap space. The Linux kernel needs to know about the swap space to be used to resume from hibernation on boot (*via* a kernel argument). When using a swap file, its offset in the device holding it also needs to be given to the kernel; that value has to be updated if the file is initialized again as swap—e.g., because its size was changed.

Note that swap space is not zeroed on shutdown, so sensitive data (such as passwords) may linger on it if it was paged out. As such, you should consider having your swap reside on an encrypted device (see Section 11.5 [Dispositivos traducidos], page 263).

`swap-space` [Data Type]

Objects of this type represent swap spaces. They contain the following members:

`target` The device or file to use, either a UUID, a `file-system-label` or a string, as in the definition of a `file-system` (see Section 11.4 [Sistemas de archivos], page 257).

`dependencies` (predeterminadas: `'()`)
A list of `file-system` or `mapped-device` objects, upon which the availability of the space depends. Note that just like for `file-system` objects, dependencies which are needed for boot and mounted in early userspace are not managed by the Shepherd, and so automatically filtered out for you.

`priority` (default: `#f`)
Only supported by the Linux kernel. Either `#f` to disable swap priority, or an integer between 0 and 32767. The kernel will first use swap spaces of higher priority when paging, and use same priority spaces on a round-robin basis. The kernel will use swap spaces without a set priority after prioritized spaces, and in the order that they appeared in (not round-robin).

`discard?` (default: `#f`)
Only supported by the Linux kernel. When true, the kernel will notify the disk controller of discarded pages, for example with the TRIM operation on Solid State Drives.

Here are some examples:

```
(swap-space (target (uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")))
```

Use the swap partition with the given UUID. You can learn the UUID of a Linux swap partition by running `swaplabel` *`device`*, where *device* is the `/dev` file name of that partition.

```
(swap-space
  (target (file-system-label "swap"))
  (dependencies mapped-devices))
```

Use the partition with label `swap`, which can be found after all the *mapped-devices* mapped devices have been opened. Again, the `swaplabel` command allows you to view and change the label of a Linux swap partition.

Here's a more involved example with the corresponding `file-systems` part of an `operating-system` declaration.

```
(file-systems
  (list (file-system
          (device (file-system-label "root"))
          (mount-point "/")
          (type "ext4"))
        (file-system
          (device (file-system-label "btrfs"))
          (mount-point "/btrfs")
          (type "btrfs"))))

(swap-devices
  (list
    (swap-space
      (target "/btrfs/swapfile")
      (dependencies (filter (file-system-mount-point-predicate "/btrfs")
                            file-systems)))))
```

Use the file `/btrfs/swapfile` as swap space, which depends on the file system mounted at `/btrfs`. Note how we use Guile's filter to select the file system in an elegant fashion!

```
(swap-devices
  (list
    (swap-space
      (target "/dev/mapper/my-swap")
      (dependencies mapped-devices))))

(kernel-arguments
  (cons* "resume=/dev/mapper/my-swap"
         %default-kernel-arguments))
```

The above snippet of an `operating-system` declaration enables the mapped device `/dev/mapper/my-swap` (which may be part of an encrypted device) as swap space, and tells the kernel to use it for hibernation via the `resume` kernel argument (see Section 11.3 [Referencia de operating-system], page 253, `kernel-arguments`).

```
(swap-devices
```

```
  (list
    (swap-space
      (target "/swapfile")
      (dependencies (filter (file-system-mount-point-predicate "/")
                            file-systems)))))

(kernel-arguments
  (cons* "resume=/dev/sda3"        ;device that holds /swapfile
         "resume_offset=92514304"  ;offset of /swapfile on device
         %default-kernel-arguments))
```

This other snippet of `operating-system` enables the swap file `/swapfile` for hibernation by telling the kernel about the partition containing it (`resume` argument) and its offset on that partition (`resume_offset` argument). The latter value can be found in the output of the command `filefrag -e` as the first number right under the `physical_offset` column header (the second command extracts its value directly):

```
$ sudo filefrag -e /swapfile
Filesystem type is: ef53
File size of /swapfile is 2463842304 (601524 blocks of 4096 bytes)
 ext:     logical_offset:        physical_offset: length:   expected: flags:
   0:        0..    2047:   92514304..  92516351:   2048:
...
$ sudo filefrag -e /swapfile | grep '^ *0:' | cut -d: -f3 | cut -d. -f1
   92514304
```

## 11.7 Cuentas de usuaria

Los grupos y cuentas de usuaria se gestionan completamente a través de la declaración `operating-system`. Se especifican con las formas `user-account` y `user-group`:

```
(user-account
  (name "alicia")
  (group "users")
  (supplementary-groups '("wheel"   ;permite usar sudo, etc.
                          "audio"   ;tarjeta de sonido
                          "video"   ;dispositivos audivisuales como cámaras
                          "cdrom")) ;el veterano CD-ROM
  (comment "hermana de Roberto"))
```

Esta es una cuenta que usa un shell diferente y un directorio personalizado (el predeterminado sería `"/home/rober"`):

```
(user-account
  (name "rober")
  (group "users")
  (comment "hermano de Alicia")
  (shell (file-append zsh "/bin/zsh"))
  (home-directory "/home/roberto"))
```

Durante el arranque o tras la finalización de `guix system reconfigure`, el sistema se asegura de que únicamente las cuentas de usuaria y grupos especificados en la declaración `operating-system` existen, y con las propiedades especificadas. Por tanto, la creación o

modificación de cuentas o grupos realizadas directamente invocando órdenes como `useradd` se pierden al reconfigurar o reiniciar el sistema. Esto asegura que el sistema permanece exactamente como se declaró.

`user-account` [Tipo de datos]

Objetos de este tipo representan cuentas de usuaria. Los siguientes miembros pueden ser especificados:

`name` El nombre de la cuenta de usuaria.

`group` Este es el nombre (una cadena) o identificador (un número) del grupo de usuarias al que esta cuenta pertenece.

`supplementary-groups` (predeterminados: `'()`)
Opcionalmente, esto puede definirse como una lista de nombres de grupo a los que esta cuenta pertenece.

`uid` (predeterminado: `#f`)
Este es el ID de usuaria para esta cuenta (un número), o `#f`. En el último caso, un número es seleccionado automáticamente por el sistema cuando la cuenta es creada.

`comment` (predeterminado: `""`)
Un comentario sobre la cuenta, como el nombre completo de la propietaria.

Note that, for non-system accounts, users are free to change their real name as it appears in `/etc/passwd` using the `chfn` command. When they do, their choice prevails over the system administrator's choice; reconfiguring does *not* change their name.

`home-directory`
Este es el nombre del directorio de usuaria de la cuenta.

`create-home-directory?` (predeterminado: `#t`)
Indica si el directorio de usuaria de esta cuenta debe ser creado si no existe todavía.

`shell` (predeterminado: Bash)
Esto es una expresión-G denotando el nombre de archivo de un programa que será usado como shell (see Section 8.12 [Expresiones-G], page 167). Por ejemplo, podría hacer referencia al ejecutable de Bash de este modo:

```
(file-append bash "/bin/bash")
```

... y al ejecutable de Zsh de este otro:

```
(file-append zsh "/bin/zsh")
```

`system?` (predeterminado: `#f`)
Este valor lógico indica si la cuenta es una cuenta "del sistema". Las cuentas del sistema se tratan a veces de forma especial; por ejemplo, los gestores gráficos de inicio no las enumeran.

`password` (predeterminada: `#f`)
Normalmente debería dejar este campo a `#f`, inicializar la contraseña de usuaria como `root` con la orden `passwd`, y entonces dejar a las usuarias

cambiarla con `passwd`. Las contraseñas establecidas con `passwd` son, por supuesto, preservadas entre reinicio y reinicio, y entre reconfiguraciones.

Si usted *realmente quiere* tener una contraseña prefijada para una cuenta, entonces este campo debe contener la contraseña cifrada, como una cadena. Puede usar el procedimiento `crypt` para este fin:

```
(user-account
  (name "carlos")
  (group "users")

  ;; Especifica una contraseña inicial mediante un hash SHA-512.
  (password (crypt "ContraseñaInicial!" "$6$abc")))
```

**Nota:** El hash de esta contraseña inicial estará disponible en un archivo en `/gnu/store`, legible por todas las usuarias, por lo que este método debe usarse con precaución.

See Section "Passphrase Storage" in *The GNU C Library Reference Manual*, para más información sobre el cifrado de contraseñas, y Section "Encryption" in *GNU Guile Reference Manual*, para información sobre el procedimiento de Guile `crypt`.

Las declaraciones de grupos incluso son más simples:

```
(user-group (name "estudiantes"))
```

`user-group` [Tipo de datos]

Este tipo es para grupos de usuarias. Hay únicamente unos pocos campos:

`name` El nombre del grupo.

`id` (predeterminado: `#f`)
El identificador del grupo (un número). Si es `#f`, un nuevo número es reservado automáticamente cuando se crea el grupo.

`system?` (predeterminado: `#f`)
Este valor booleano indica si el grupo es un grupo "del sistema". Los grupos del sistema tienen identificadores numéricos bajos.

`password` (predeterminada: `#f`)
¿Qué? ¿Los grupos de usuarias pueden tener una contraseña? Bueno, aparentemente sí. A menos que sea `#f`, este campo especifica la contraseña del grupo.

Por conveniencia, una variable contiene una lista con todos los grupos de usuarias básicos que se puede esperar:

`%base-groups` [Variable]

Esta es la lista de grupos de usuarias básicos que las usuarias y/o los paquetes esperan que estén presentes en el sistema. Esto incluye grupos como "root", "wheel" y "users", así como grupos usados para controlar el acceso a dispositivos específicos como "audio", "disk" y "cdrom".

`%base-user-accounts` [Variable]

Esta es la lista de cuentas de usuaria básicas que los programas pueden esperar encontrar en un sistema GNU/Linux, como la cuenta "nobody".

Fíjese que la cuenta de "root" no se incluye aquí. Es un caso especial y se añade automáticamente esté o no especificada.

## 11.8 Distribución de teclado

Para especificar qué hace cada tecla de su teclado, necesita decirle al sistema operativo qué *distribución de teclado* desea usar. La predeterminada, cuando no se especifica ninguna, es la distribución QWERTY de 105 teclas para PC de teclado inglés estadounidense. No obstante, las personas germano-parlantes habitualmente prefieren la distribución QWERTZ alemana, las franco-parlantes desearán la distribución AZERTY, etcétera; las hackers pueden preferir Dvorak o bépo, y pueden incluso desear personalizar más aún el efecto de determinadas teclas. Esta sección explica cómo hacerlo.

Hay tres componentes que desearán conocer la distribución de su teclado:

- El *cargador de arranque* puede desear conocer cual es la distribución de teclado que desea usar (see Section 11.15 [Configuración del gestor de arranque], page 627). Esto es útil si desea, por ejemplo, asegurarse de que puede introducir la contraseña de cifrado de su partición raíz usando la distribución correcta.
- El *núcleo del sistema operativo*, Linux, la necesitará de manera que la consola se configure de manera adecuada (see Section 11.3 [Referencia de operating-system], page 253).
- El *servidor gráfico*, habitualmente Xorg, también tiene su propia idea de distribución de teclado (see Section 11.10.7 [Sistema X Window], page 340).

Guix le permite configurar las tres distribuciones por separado pero, afortunadamente, también le permite compartir la misma distribución de teclado para los tres componentes.

Las distribuciones de teclado se representan mediante registros creados con el procedimiento `keyboard-layout` de `(gnu system keyboard)`. A imagen de la extensión de teclado de X (XKB), cada distribución tiene cuatro atributos: un nombre (habitualmente un código de idioma como "fi" para finés o "jp" para japonés), un nombre opcional de variante, un nombre opcional de modelo de teclado y una lista, puede que vacía, de opciones adicionales. En la mayor parte de los casos el nombre de la distribución es lo único que le interesará.

`keyboard-layout` *nombre* [*variante*] [*#:model*] [*#:options '()*] [Procedimiento]

Devuelve una distribución de teclado para el *nombre* y la *variante* que se proporcionan.

*nombre* debe ser una cadena como `"fr"`; *variante* debe ser una cadena como `"bepo"` o `"nodeadkeys"`. Véase el paquete `xkeyboard-config` para las opciones válidas.

Estos son algunos ejemplos:

```
;; La distribución QWERTZ alemana. Se asume un modelo de
;; teclado "pc105" estándar.
(keyboard-layout "de")

;; La variante bépo de la distribución francesa.
(keyboard-layout "fr" "bepo")
```

```
;; La distribución de teclado catalana.
(keyboard-layout "es" "cat")

;; Distribución de teclado árabe con "Alt-Shift" para cambiar
;; a la distribución de teclado de EEUU.
(keyboard-layout "ar,us" #:options '("grp:alt_shift_toggle"))

;; La distribución de teclado de latinoamérica. Además,
;; la tecla "Bloq Mayús" se usa como una tecla "Ctrl"
;; adicional, y la tecla "Menú" se usa como una tecla
;; "Componer/Compose" para introducir letras acentuadas.
(keyboard-layout "latam"
                 #:options '("ctrl:nocaps" "compose:menu"))

;; La distribución rusa para un teclado ThinkPad.
(keyboard-layout "ru" #:model "thinkpad")

;; La distribución estadounidense internacional, la cual es
;; la distribución estadounidense junto a teclas muertas para
;; introducir caracteres acentuados. Esta es para un teclado
;; Apple MackBook.
(keyboard-layout "us" "intl" #:model "macbook78")
```

Véase el directorio `share/X11/xkb` del paquete `xkeyboard-config` para una lista completa de implementaciones de distribuciones, variantes y modelos.

Digamos que desea que su sistema use la distribución de teclado turca a lo largo de todo su sistema—cargador de arranque, consola y Xorg. Así es como sería su configuración del sistema:

```
;; Usando la distribución turca para el cargador de
;; arranque, la consola y Xorg.

(operating-system
  ;; ...
  (keyboard-layout (keyboard-layout "tr"))  ;for the console
  (bootloader (bootloader-configuration
               (bootloader grub-efi-bootloader)
               (targets '("/boot/efi"))
               (keyboard-layout keyboard-layout))) ;for GRUB
  (services (cons (set-xorg-configuration
                   (xorg-configuration             ;for Xorg
                     (keyboard-layout keyboard-layout)))
                  %desktop-services)))
```

En el ejemplo previo, para GRUB y para Xorg, simplemente hemos hecho referencia al campo `keyboard-layout` definido previamente, pero también podíamos haber hecho referencia a una distribución diferente. El procedimiento `set-xorg-configuration` comunica la configuración de Xorg deseada al gestor gráfico de ingreso en el sistema, GDM por omisión.

Hemos tratado cómo especificar la distribución *predeterminada* del teclado de su sistema cuando arranca, pero también la puede modificar en tiempo de ejecución:

- Si usa GNOME, su panel de configuración tiene una entrada de "Región e Idioma" donde puede seleccionar una o más distribuciones de teclado.
- En Xorg, la orden `setxkbmap` (del paquete con el mismo nombre) le permite cambiar la distribución en uso actualmente. Por ejemplo, así es como cambiaría a la distribución Dvorak estadounidense:

  ```
  setxkbmap us dvorak
  ```

- La orden `loadkeys` cambia la distribución de teclado en efecto en la consola Linux. No obstante, tenga en cuenta que `loadkeys` *no* usa la categorización de distribuciones de XKB descrita previamente. La orden a continuación carga la distribución francesa bépo:

  ```
  loadkeys fr-bepo
  ```

## 11.9 Localizaciones

Una *localización* define convenciones culturales para una lengua y región del mundo particular (see Section "Locales" in *The GNU C Library Reference Manual*). Cada localización tiene un nombre que típicamente tiene la forma de *`lengua_territorio.codificación`*—por ejemplo, `fr_LU.utf8` designa la localización para la lengua francesa, con las convenciones culturales de Luxemburgo, usando la codificación UTF-8.

Normalmente deseará especificar la localización predeterminada para la máquina usando el campo `locale` de la declaración `operating-system` (see Section 11.3 [Referencia de operating-system], page 253).

La localización seleccionada es automáticamente añadida a las *definiciones de localización* conocidas en el sistema si es necesario, con su codificación inferida de su nombre—por ejemplo, se asume que `bo_CN.utf8` usa la codificación `UTF-8`. Definiciones de localización adicionales pueden ser especificadas en el campo `locale-definitions` de `operating-system`—esto es util, por ejemplo, si la codificación no puede ser inferida del nombre de la localización. El conjunto predeterminado de definiciones de localización incluye algunas localizaciones ampliamente usadas, pero no todas las disponibles, para ahorrar espacio.

Por ejemplo, para añadir la localización del frisio del norte para Alemania, el valor de dicho campo puede ser:

```
(cons (locale-definition
        (name "fy_DE.utf8") (source "fy_DE"))
      %default-locale-definitions)
```

De mismo modo, para ahorrar espacio, se puede desear que `locale-definitions` contenga únicamente las localizaciones que son realmente usadas, como en:

```
(list (locale-definition
        (name "ja_JP.eucjp") (source "ja_JP")
        (charset "EUC-JP")))
```

Las definiciones de localización compiladas están disponibles en `/run/current-system/locale/X.Y`, donde `X.Y` es la versión de libc, que es la ruta donde la GNU libc contenida en Guix buscará los datos de localización. Esto puede ser sobreescrito usando la variable de entorno `LOCPATH` (see [locales-and-locpath], page 17).

La forma `locale-definition` es proporcionada por el módulo `(gnu system locale)`. Los detalles se proporcionan a continuación.

`locale-definition` [Tipo de datos]

Este es el tipo de datos de una definición de localización.

`name` El nombre de la localización. See Section "Locale Names" in *The GNU C Library Reference Manual*, para más información sobre nombres de localizaciones.

`source` El nombre de la fuente para dicha localización. Habitualmente es la parte *`idioma_territorio`* del nombre de localización.

`charset` (predeterminado: `"UTF-8"`)
La "codificación de caracteres" o "conjunto de caracteres" para dicha localización, como lo define IANA (`https://www.iana.org/assignments/character-sets`).

`%default-locale-definitions` [Variable]

Una lista de localizaciones UTF-8 usadas de forma común, usada como valor predeterminado del campo `locale-definitions` en las declaraciones `operating-system`.

Estas definiciones de localizaciones usan la *codificación normalizada* para el fragmento tras el punto en el nombre (see Section "Using gettextized software" in *The GNU C Library Reference Manual*). Por lo que por ejemplo es válido `uk_UA.utf8` pero *no*, digamos, `uk_UA.UTF-8`.

### 11.9.1 Consideraciones sobre la compatibilidad de datos de localización

Las declaraciones `operating-system` proporcionan un campo `locale-libcs` para especificar los paquetes GNU libc que se usarán para compilar las declaraciones de localizaciones (see Section 11.3 [Referencia de operating-system], page 253). "¿Por qué debo preocuparme?", puede preguntarse. Bueno, sucede que el formato binario de los datos de localización es ocasionalmente incompatible de una versión de libc a otra.

Por ejemplo, un programa enlazado con la versión 2.21 de libc no puede leer datos de localización producidos con libc 2.22; peor aún, ese programa *aborta* en vez de simplemente ignorar los datos de localización incompatibles[5]. De manera similar, un programa enlazado con libc 2.22 puede leer la mayor parte, pero no todo, de los datos de localización de libc 2.21 (específicamente, los datos `LC_COLLATE` son incompatibles); por tanto las llamadas a `setlocale` pueden fallar, pero los programas no abortarán.

El "problema" con Guix es que las usuarias tienen mucha libertad: pueden elegir cuando e incluso si actualizar el software en sus perfiles, y pueden estar usando una versión de libc diferente de la que la administradora del sistema usó para construir los datos de localización comunes a todo el sistema.

Por suerte, las usuarias sin privilegios también pueden instalar sus propios datos de localización y definir `GUIX_LOCPATH` de manera adecuada (see [locales-and-locpath], page 17).

[5] Las versiones 2.23 y posteriores de GNU libc simplemente ignorarán los datos de localización incompatibles, lo cual ya es un avance.

No obstante, es mejor si los datos de localización globales del sistema en `/run/current-system/locale` se construyen para todas las versiones de libc realmente en uso en el sistema, de manera que todos los programas puedan acceder a ellos—esto es especialmente crucial en un sistema multiusuaria. Para hacerlo, la administradora puede especificar varios paquetes libc en el campo `locale-libcs` de `operating-system`:

```
(use-package-modules base)

(operating-system
  ;; ...
  (locale-libcs (list glibc-2.21 (canonical-package glibc))))
```

Este ejemplo llevaría a un sistema que contiene definiciones de localización tanto para libc 2.21 como para la versión actual de libc en `/run/current-system/locale`.

## 11.10 Servicios

Una parte importante de la preparación de una declaración `operating-system` es listar los *servicios del sistema* y su configuración (see Section 11.2 [Uso de la configuración del sistema], page 244). Los servicios del sistema típicamente son daemon lanzados cuando el sistema arrancha, u otras acciones necesarias en ese momento—por ejemplo, configurar el acceso de red.

Guix tiene una definición amplia de "servicio" (see Section 11.19.1 [Composición de servicios], page 649), pero muchos servicios se gestionan por GNU Shepherd (see Section 11.19.4 [Servicios de Shepherd], page 657). En un sistema en ejecución, la orden `herd` le permite enumerar los servicios disponibles, mostrar su estado, arrancarlos y pararlos, o realizar otras acciones específicas (see Section "Jump Start" in *The GNU Shepherd Manual*). Por ejemplo:

```
# herd status
```

La orden previa, ejecutada como `root`, enumera los servicios actualmente definidos. La orden `herd doc` muestra una sinopsis del servicio proporcionado y sus acciones asociadas:

```
# herd doc nscd
Run libc's name service cache daemon (nscd).

# herd doc nscd action invalidate
invalidate: Invalidate the given cache--e.g., 'hosts' for host name lookups.
```

Las ordenes internas `start`, `stop` y `restart` tienen el efecto de arrancar, parar y reiniciar el servicio, respectivamente. Por ejemplo, las siguientes órdenes paran el servicio nscd y reinician el servidor gráfico Xorg:

```
# herd stop nscd
Service nscd has been stopped.
# herd restart xorg-server
Service xorg-server has been stopped.
Service xorg-server has been started.
```

For some services, `herd configuration` returns the name of the service's configuration file, which can be handy to inspect its configuration:

```
# herd configuration sshd
```

```
/gnu/store/...-sshd_config
```

Las siguientes secciones documentan los servicios disponibles, comenzando con los servicios básicos, que pueden ser usados en una declaración `operating-system`.

### 11.10.1 Servicios base

El módulo `(gnu services base)` proporciona definiciones para los servicios básicos que se esperan en el sistema. Los servicios exportados por este módulo se enumeran a continuación.

`%base-services` [Variable]

Esta variable contiene una lista de servicios básicos (see Section 11.19.2 [Tipos de servicios y servicios], page 650, para más información sobre los objetos servicio) que se pueden esperar en el sistema: un servicio de ingreso al sistema (mingetty) en cada tty, syslogd, el daemon de la caché del servicio de nombres (nscd), el gestor de dispositivos udev, y más.

Este es el valor predeterminado del campo `services` de las declaraciones `operating-system`. De manera habitual, cuando se personaliza el sistema, es deseable agregar servicios a `%base-services`, de esta forma:

```
(append (list (service avahi-service-type)
              (service openssh-service-type))
        %base-services)
```

`special-files-service-type` [Variable]

El servicio que establece "archivos especiales" como `/bin/sh`; una instancia suya es parte de `%base-services`.

The value associated with `special-files-service-type` services must be a list of two-element lists where the first element is the "special file" and the second element is its target. By default it is:

```
`(("/bin/sh" ,(file-append bash "/bin/sh"))
  ("/usr/bin/env" ,(file-append coreutils "/bin/env")))
```

If you want to add, say, `/bin/bash` to your system, you can change it to:

```
`(("/bin/sh" ,(file-append bash "/bin/sh"))
  ("/usr/bin/env" ,(file-append coreutils "/bin/env"))
  ("/bin/bash" ,(file-append bash "/bin/bash")))
```

Ya que es parte de `%base-services`, puede usar `modify-services` para personalizar el conjunto de archivos especiales (see Section 11.19.3 [Referencia de servicios], page 652). Pero una forma simple de añadir un archivo especial es usar el procedimiento `extra-special-file` (véase a continuación).

`extra-special-file` *archivo destino* [Procedimiento]

Usa *destino* como el "archivo especial" *archivo*.

Por ejemplo, la adición de las siguientes líneas al campo `services` de su declaración de sistema operativo genera `/usr/bin/env` como un enlace simbólico:

```
(extra-special-file "/usr/bin/env"
                    (file-append coreutils "/bin/env"))
```

`host-name-service-type` [Variable]

Type of the service that sets the system host name, whose value is a string. This service is included in `operating-system` by default (see [operating-system-essential-services], page 256).

`console-font-service-type` [Variable]

Instala las tipografías proporcionadas en las consolas virtuales (tty) especificados (las tipografías se asocian a cada consola virtual con el núcleo Linux). El valor de este servicio es una lista de pares tty/tipografía. La tipografía puede ser el nombre de alguna de las proporcionadas por el paquete `kbd` o cualquier parámetro válido para la orden `setfont`, como en este ejemplo:

```
`(("tty1" . "LatGrkCyr-8x16")
  ("tty2" . ,(file-append
              font-tamzen
              "/share/kbd/consolefonts/TamzenForPowerline10x20.psf"))
  ("tty3" . ,(file-append
              font-terminus
              "/share/consolefonts/ter-132n"))) ; para HDPI
```

`hosts-service-type` [Variable]

Type of the service that populates the entries for (`/etc/hosts`). This service type can be *extended* by passing it a list of `host` records.

The example below shows how to add two entries to `/etc/hosts`:

```
(simple-service 'add-extra-hosts
                hosts-service-type
                (list (host "192.0.2.1" "example.com"
                            '("example.net" "example.org"))
                      (host "2001:db8::1" "example.com"
                            '("example.net" "example.org"))))
```

**Nota:**

By default `/etc/hosts` comes with the following entries:

```
127.0.0.1 localhost host-name
::1       localhost host-name
```

For most setups this is what you want though if you find yourself in the situation where you want to change the default entries, you can do so in `operating-system` via `modify-services` (see Section 11.19.3 [Referencia de servicios], page 652).

The following example shows how to unset *host-name* from being an alias of `localhost`.

```
(operating-system
  ;; ...

  (essential-services
   (modify-services
     (operating-system-default-essential-services this-operating-system)
```

```
(hosts-service-type config => (list
                                (host "127.0.0.1" "localhost")
                                (host "::1"       "localhost"))))))
```

`host` *address canonical-name* [*aliases*] [Procedure]
Return a new record for the host at *address* with the given *canonical-name* and possibly *aliases*.

*address* must be a string denoting a valid IPv4 or IPv6 address, and *canonical-name* and the strings listed in *aliases* must be valid host names.

`login-service-type` [Variable]
Type of the service that provides a console login service, whose value is a `<login-configuration>` object.

`login-configuration` [Tipo de datos]
Data type representing the configuration of login, which specifies the MOTD (message of the day), among other things.

`motd` Un objeto tipo-archivo que contiene el "mensaje del día".

`allow-empty-passwords?` (predeterminado: `#t`)
Permite contraseñas vacías por defecto para que las primeras usuarias puedan ingresar en el sistema cuando la cuenta de "root" está recién creada.

`mingetty-service-type` [Variable]
Type of the service that runs Mingetty, an implementation of the virtual console log-in. The value for this service is a `<mingetty-configuration>` object.

`mingetty-configuration` [Tipo de datos]
Data type representing the configuration of Mingetty, which specifies the tty to run, among other things.

`tty` El nombre de la consola en la que se ejecuta este Mingetty—por ejemplo, `"tty1"`.

`auto-login` (predeterminado: `#f`)
Cuando sea verdadero, este campo debe ser una cadena que denote el nombre de usuaria bajo el cual el sistema ingresa automáticamente. Cuando es `#f`, se deben proporcionar un nombre de usuaria y una contraseña para ingresar en el sistema.

`login-program` (predeterminado: `#f`)
Debe ser `#f`, en cuyo caso se usa el programa predeterminado de ingreso al sistema (`login` de las herramientas Shadow), o una expresión-G que determine el nombre del programa de ingreso al sistema.

`login-pause?` (predeterminado: `#f`)
Cuando es `#t` en conjunción con *auto-login*, la usuaria deberá presionar una tecla para lanzar el shell de ingreso al sistema.

`clear-on-logout?` (default: `#t`)
When set to `#t`, the screen will be cleared after logout.

`mingetty` (predeterminado: *mingetty*)
: El paquete Mingetty usado.

`agetty-service-type` [Variable]
: Type of the service that runs agetty, which implements virtual and serial console log-in. The value for this service is a `<agetty-configuration>` object.

`agetty-configuration` [Tipo de datos]
: Data type representing the configuration of agetty, which specifies the tty to run, among other things[6].

`tty`
: El nombre de la consola en la que se ejecuta este agetty, como una cadena—por ejemplo, `"ttyS0"`. Este parámetro es opcional, su valor predeterminado es un puerto serie razonable usado por el núcleo Linux.

  Para ello, si hay un valor para una opción `agetty.tty` en la línea de órdenes del núcleo, agetty extraerá el nombre del dispositivo del puerto serie de allí y usará dicho valor.

  Si no y hay un valor para la opción `console` con un tty en la línea de órdenes de Linux, agetty extraerá el nombre del dispositivo del puerto serie de allí y usará dicho valor.

  En ambos casos, agetty dejará el resto de configuración de dispositivos serie (tasa de transmisión, etc.) sin modificar—con la esperanza de que Linux haya proporcionado ya los valores correctos.

`baud-rate` (predeterminado: `#f`)
: Una cadena que contenga una lista separada por comas de una o más tasas de transmisión, en orden descendiente.

`term` (predeterminado: `#f`)
: Una cadena que contiene el valor usado para la variable de entorno `TERM`.

`eight-bits?` (predeterminado: `#f`)
: En caso de ser `#t`, se asume que el tty permite el paso de 8 bits, y la detección de paridad está desactivada.

`auto-login` (predeterminado: `#f`)
: Cuando se proporciona un nombre de ingreso al sistema, como una cadena, la usuaria especificada ingresará automáticamente sin solicitar su nombre de ingreso ni su contraseña.

`no-reset?` (predeterminado: `#f`)
: En caso de ser `#t`, no reinicia los modos de control del terminal (cflags).

`host` (predeterminado: `#f`)
: Acepta una cadena que contenga el "nombre_de_máquina_de_ingreso", que será escrito en el archivo `/var/run/utmpx`.

`remote?` (predeterminado: `#f`)
: Cuando se fija a `#t` en conjunción con *host*, se añadirá una opción `-r "fakehost"` a la línea de órdenes del programa de ingreso al sistema especificado en *login-program*.

[6] See the `agetty(8)` man page for more information.

`flow-control?` (predeterminado: `#f`)
Cuando es `#t`, activa el control de flujo hardware (RTS/CTS).

`no-issue?` (predeterminado: `#f`)
Cuando es `#t`, el contenido del archivo `/etc/issue` no se mostrará antes de presentar el mensaje de ingreso al sistema.

`init-string` (predeterminada: `#f`)
Esto acepta una cadena que se enviará al tty o módem antes de mandar nada más. Puede usarse para inicializar un modem.

`no-clear?` (predeterminado: `#f`)
Cuando es `#t`, agetty no limpiará la pantalla antes de mostrar el mensaje de ingreso al sistema.

`login-program` (predeterminado: (file-append shadow "/bin/login"))
Esto debe ser o bien una expresión-g que denote el nombre del programa de ingreso al sistema, o no debe proporcionarse, en cuyo caso el valor predeterminado es `login` del conjunto de herramientas Shadow.

`local-line` (predeterminado: `#f`)
Control the CLOCAL line flag. This accepts one of three symbols as arguments, `'auto`, `'always`, or `'never`. If `#f`, the default value chosen by agetty is `'auto`.

`extract-baud?` (predeterminado: `#f`)
Cuando es `#t`, instruye a agetty para extraer la tasa de transmisión de los mensajes de estado producidos por ciertos tipos de módem.

`skip-login?` (predeterminado: `#f`)
Cuando es `#t`, no solicita el nombre de la usuaria para el ingreso al sistema. Puede usarse con el campo *login-program* para usar sistemas de ingreso no estándar.

`no-newline?` (predeterminado: `#f`)
Cuando es `#t`, no imprime una nueva línea antes de imprimir el archivo `/etc/issue`.

`login-options` (predeterminadas: `#f`)
Esta opción acepta una cadena que contenga opciones para proporcionar al programa de ingreso al sistema. Cuando se use con *login-program*, sea consciente de que una usuaria con malas intenciones podría intentar introducir un nombre que contuviese opciones embebidas que serían procesadas por el programa de ingreso.

`login-pause` (predeterminada: `#f`)
Cuando es `#t`, espera la pulsación de cualquier tecla antes de mostrar el mensaje de ingreso al sistema. Esto puede usarse en conjunción con *auto-login* para ahorrar memoria lanzando cada shell cuando sea necesario.

`chroot` (predeterminado: `#f`)
Cambia la raíz al directorio especificado. Esta opción acepta una ruta de directorio como una cadena.

`hangup?` (predeterminado: `#f`)
: Usa la llamada del sistema Linux `vhangup` para colgar de forma virtual el terminal especificado.

`keep-baud?` (predeterminado: `#f`)
: Cuando es `#t`, prueba a mantener la tasa de transmisión existente. Las tasas de transmisión de *baud-rate* se usan cuando agetty recibe un carácter `BREAK`.

`timeout` (predeterminado: `#f`)
: Cuando sea un valor entero, termina si no se pudo leer ningún nombre de usuaria en *timeout* segundos.

`detect-case?` (predeterminado: `#f`)
: Cuando es `#t`, activa la detección de terminales únicamente con mayúsculas. ESta configuración detectará un nombre de ingreso que contenga únicamente letras mayúsculas como un indicativo de un terminal con letras únicamente mayúsculas y activará las conversiones de mayúscula a minúscula. Tenga en cuenta que esto no permitirá caracteres Unicode.

`wait-cr?` (predeterminado: `#f`)
: Cuando es `#t`, espera hasta que la usuaria o el modem envíen un carácter de retorno de carro o de salto de línea antes de mostrar `/etc/issue` o el mensaje de ingreso. Se usa de forma típica junto a la opción *init-string*.

`no-hints?` (predeterminado: `#f`)
: Cuando es `#t`, no imprime avisos sobre el bloqueo numérico, las mayúsculas o el bloqueo del desplazamiento.

`no-hostname?` (predeterminado: `#f`)
: El nombre de la máquina se imprime de forma predeterminada. Cuando esta opción es `#t`, no se mostrará ningún nombre de máquina.

`long-hostname?` (predeterminado: `#f`)
: El nombre de máquina se imprime de forma predeterminada únicamente hasta el primer punto. Cuando esta opción es `#t`, se muestra el nombre completamente cualificado de la máquina mostrado por `gethostname` o `getaddrinfo`.

`erase-characters` (predeterminado: `#f`)
: Esta opción acepta una cadena de caracteres adicionales que deben interpretarse como borrado del carácter anterior cuando la usuaria introduce su nombre de ingreso.

`kill-characters` (predeterminado: `#f`)
: Esta opción acepta una cadena de que debe ser interpretada como “ignora todos los caracteres anteriores” (también llamado carácter “kill”) cuando la usuaria introduce su nombre de ingreso.

`chdir` (predeterminado: `#f`)
: Esta opción acepta, como una cadena, una ruta de directorio que a la que se cambiará antes del ingreso al sistema.

`delay` (predeterminado: `#f`)
: Esta opción acepta, como un entero, el número de segundos a esperar antes de abrir el tty y mostrar el mensaje de ingreso al sistema.

`nice` (predeterminado: `#f`)
: Esta opción acepta, como un entero, el valor "nice" con el que se ejecutará el programa `login`.

`extra-options` (predeterminadas: `'()`)
: Esta opción proporciona una "trampilla de escape" para que la usuaria proporcione parámetros de línea de órdenes adicionales a `agetty` como una lista de cadenas.

`shepherd-requirement` (default: `'()`)
: The option can be used to provides extra shepherd requirements (for example `'syslogd`) to the respective `'term-`* shepherd service.

`kmscon-service-type` [Variable]
: Type of the service that runs kmscon (`https://www.freedesktop.org/wiki/Software/kmscon`), which implements virtual console log-in. The value for this service is a `<kmscon-configuration>` object.

`kmscon-configuration` [Tipo de datos]
: Data type representing the configuration of Kmscon, which specifies the tty to run, among other things.

`virtual-terminal`
: El nombre de la consola en la que se ejecuta este Kmscon—por ejemplo, `"tty1"`.

`login-program` (predeterminado: `#~(string-append #$shadow "/bin/login")`)
: A gexp denoting the name of the log-in program. The default log-in program is `login` from the Shadow tool suite.

`login-arguments` (predeterminados: `'("-p")`)
: Una lista de parámetros para proporcionar a `login`.

`auto-login` (predeterminado: `#f`)
: Cuando se proporciona un nombre de ingreso al sistema, como una cadena, la usuaria especificada ingresará automáticamente sin solicitar su nombre de ingreso ni su contraseña.

`hardware-acceleration?` (predeterminado: #f)
: Determina si se usará aceleración hardware.

`font-engine` (default: `"pango"`)
: Font engine used in Kmscon.

`font-size` (default: `12`)
: Font size used in Kmscon.

`keyboard-layout` (predeterminada: `#f`)
: If this is `#f`, Kmscon uses the default keyboard layout—usually US English ("qwerty") for a 105-key PC keyboard.

Otherwise this must be a `keyboard-layout` object specifying the keyboard layout. See Section 11.8 [Distribución de teclado], page 271, for more information on how to specify the keyboard layout.

`kmscon` (predeterminado: *kmscon*)
El paquete Kmscon usado.

`nscd-service-type` [Variable]
Type of the service that runs the libc nscd (name service cache daemon), whose value is an `<nscd-configuration>` object.

Por conveniencia, el servicio ncsd de Shepherd proporciona las siguientes acciones:

`invalidate`
Esto invalida la caché dada. Por ejemplo, ejecutar:

```
herd invalidate nscd hosts
```

invalida la caché de búsqueda de nombres de máquinas de nscd.

`statistics`
Ejecutar `herd statistics nscd` muestra información del uso nscd y la caché.

`nscd-configuration` [Tipo de datos]
Data type representing the nscd (name service cache daemon) configuration.

`name-services` (predeterminados: `'()`)
Lista de paquetes que indican los *servicios de nombres* que serán visibles al nscd—por ejemplo, `(list nss-mdns)`.

`glibc` (predeterminada: *glibc*)
Paquete que denota la biblioteca C de GNU que proporciona la orden `nscd`.

`log-file` (predeterminado: `"/var/log/nscd.log"`)
Nombre del archivo de registro de nscd. Aquí es donde se almacena la salida de depuración cuando `debug-level` es estrictamente positivo.

`debug-level` (predeterminado: `0`)
Entero que indica el nivel de depuración. Números mayores significan que se registra más salida de depuración.

`caches` (predeterminado: `%nscd-default-caches`)
Lista de objetos `<nscd-cache>` que indican cosas a mantener en caché; véase a continuación.

`nscd-cache` [Tipo de datos]
Tipo de datos que representa una base de datos de caché de nscd y sus parámetros.

`base de datos`
Es un símbolo que representa el nombre de la base de datos de la que se actúa como caché. Se aceptan los valores `passwd`, `group`, `hosts` y `services`, que designan las bases de datos NSS correspondientes (see Section “NSS Basics” in *The GNU C Library Reference Manual*).

`positive-time-to-live`
`negative-time-to-live` (predeterminado: `20`)
: Un número que representa el número de segundos durante los que una búsqueda positiva o negativa permanece en la caché.

`check-files?` (predeterminado: `#t`)
: Si se comprobará en busca de actualizaciones los archivos que correspondan con *database*.

  Por ejemplo, cuando *database* es `hosts`, la activación de esta opción instruye a nscd para comprobar actualizaciones en `/etc/hosts` y tenerlas en cuenta.

`persistent?` (predeterminada: `#t`)
: Determina si la caché debe almacenarse de manera persistente en disco.

`shared?` (predeterminado: `#t`)
: Determina si la caché debe compartirse entre las usuarias.

`max-database-size` (predeterminado: 32 MiB)
: Tamaño máximo en bytes de la caché de la base de datos.

`%nscd-default-caches` [Variable]
: Lista de objetos `<nscd-cache>` usados por omisión por `nscd-configuration` (véase en la sección previa)

  Activa el almacenamiento en caché persistente y agresivo de búsquedas de servicios y nombres de máquina. La última proporciona un mejor rendimiento en la búsqueda de nombres de máquina, resilencia en caso de nombres de servidor no confiables y también mejor privacidad—a menudo el resultado de las búsquedas de nombres de máquina está en la caché local, por lo que incluso ni es necesario consultar servidores de nombres externos.

`syslog-service-type` [Variable]
: Type of the service that runs the syslog daemon, whose value is a `<syslog-configuration>` object.

To have a modified `syslog-configuration` come into effect after reconfiguring your system, the '`reload`' action should be preferred to restarting the service, as many services such as the login manager depend on it and would be restarted as well:

```
# herd reload syslog
```

which will cause the running `syslogd` process to reload its configuration.

`syslog-configuration` [Tipo de datos]
: Data type representing the configuration of the syslog daemon.

  `syslogd` (predeterminado: `#~(string-append #$inetutils "/libexec/syslogd")`)
  : El daemon syslog usado.

  `config-file` (predeterminado: `%default-syslog.conf`)
  : The syslog configuration file to use. See Section "syslogd invocation" in *GNU Inetutils*, for more information on the configuration file syntax.

`guix-service-type` [Variable]
El tipo de servicio que ejecuta el daemon de construcción, `guix-daemon` (see Section 2.3 [Invocación de guix-daemon], page 12). Su valor debe ser un registro `guix-configuration` como se describe a continuación.

`guix-configuration` [Tipo de datos]
Este tipo de datos representa la configuración del daemon de construcción de Guix. See Section 2.3 [Invocación de guix-daemon], page 12, para más información.

`guix` (predeterminado: *guix*)
The Guix package to use. See Section 6.4 [Customizing the System-Wide Guix], page 71, to learn how to provide a package with a pre-configured set of channels.

`build-group` (predeterminado: `"guixbuild"`)
El nombre del grupo de las cuentas de usuarias de construcción.

`build-accounts` (predeterminadas: `10`)
Número de cuentas de usuarias de construcción a crear.

`authorize-key?` (predeterminado: `#t`)
Whether to authorize the substitute keys listed in `authorized-keys`—by default that of `bordeaux.guix.gnu.org` and `ci.guix.gnu.org` (see Section 5.3 [Sustituciones], page 46).

Cuando `authorize-key?` es verdadero, `/etc/guix/acl` no se puede cambiar a través de `guix archive --authorize`. En vez de eso debe ajustar `guix-configuration` como desee y reconfigurar el sistema. Esto asegura que la configuración de su sistema operativo es auto-contenida.

> **Nota:** Cuando arranque o reconfigure a un sistema donde `authorize-key?` sea verdadero, se crea una copia de seguridad del archivo `/etc/guix/acl` existente como `/etc/guix/acl.bak` si se determina que el archivo se ha modificado de manera manual. Esto facilita la migración desde versiones anteriores, en las que se permitían las modificaciones directas del archivo `/etc/guix/acl`.

`authorized-keys` (predeterminadas: `%default-authorized-guix-keys`)
The list of authorized key files for archive imports, as a list of string-valued gexps (see Section 5.11 [Invocación de guix archive], page 66). By default, it contains that of `bordeaux.guix.gnu.org` and `ci.guix.gnu.org` (see Section 5.3 [Sustituciones], page 46). See `substitute-urls` below for an example on how to change it.

`use-substitutes?` (predeterminado: `#t`)
Determina si se usarán sustituciones.

`substitute-urls` (predeterminado: `%default-substitute-urls`)
La lista de URLs donde se buscarán sustituciones por defecto.

Suppose you would like to fetch substitutes from `guix.example.org` in addition to `bordeaux.guix.gnu.org`. You will need to do two things: (1)

add `guix.example.org` to `substitute-urls`, and (2) authorize its signing key, having done appropriate checks (see Section 5.3.2 [Autorización de servidores de sustituciones], page 47). The configuration below does exactly that:

```
(guix-configuration
  (substitute-urls
   (append (list "https://guix.example.org")
           %default-substitute-urls))
  (authorized-keys
   (append (list (local-file "./guix.example.org-clave.pub"))
           %default-authorized-guix-keys)))
```

Este ejemplo asume que el archivo `./guix.example.org-clave.pub` contiene la clave pública que `guix.example.org` usa para firmar las sustituciones.

`generate-substitute-key?` (default: `#t`)
: Whether to generate a *substitute key pair* under `/etc/guix/signing-key.pub` and `/etc/guix/signing-key.sec` if there is not already one.

  This key pair is used when exporting store items, for instance with `guix publish` (see Section 9.11 [Invocación de guix publish], page 226) or `guix archive` (see Section 5.11 [Invocación de guix archive], page 66). Generating a key pair takes a few seconds when enough entropy is available and is only done once; you might want to turn it off for instance in a virtual machine that does not need it and where the extra boot time is a problem.

`channels` (predeterminados: `%default-channels`)
: List of channels to be specified in `/etc/guix/channels.scm`, which is what `guix pull` uses by default (see Section 5.7 [Invocación de guix pull], page 57).

  > **Nota:** When reconfiguring a system, the existing `/etc/guix/channels.scm` file is backed up as `/etc/guix/channels.scm.bak` if it was determined to be a manually modified file. This is to facilitate migration from earlier versions, which allowed for in-place modifications to `/etc/guix/channels.scm`.

`max-silent-time` (default: `3600`)
`timeout` (default: `(* 3600 24)`)
: El número de segundos de silencio y el número de segundos de actividad respectivamente, tras los cuales el proceso de construcción supera el plazo. Un valor de cero proporciona plazos ilimitados.

`log-compression` (default: `'gzip`)
: El tipo de compresión usado en los log de construcción—o bien `gzip`, o bien `bzip2` o `none`.

`discover?` (default: `#f`)
: Whether to discover substitute servers on the local network using mDNS and DNS-SD.

`build-machines` (default: `#f`)
: This field must be either `#f` or a list of gexps evaluating to a `build-machine` record or to a list of `build-machine` records (see Section 2.2.2 [Configuración de delegación del daemon], page 7).

  When it is `#f`, the `/etc/guix/machines.scm` file is left untouched. Otherwise, the list of of gexps is written to `/etc/guix/machines.scm`; if a previously-existing file is found, it is backed up as `/etc/guix/machines.scm.bak`. This allows you to declare build machines for offloading directly in the operating system declaration, like so:

```
(guix-configuration
  (build-machines
    (list #~(build-machine (name "foo.example.org") ...)
          #~(build-machine (name "bar.example.org") ...))))
```

  Additional build machines may be added *via* the `guix-extension` mechanism (see below).

`extra-options` (predeterminadas: `'()`)
: Lista de opciones de línea de órdenes adicionales para `guix-daemon`.

`log-file` (predeterminado: `"/var/log/guix-daemon.log"`)
: Archivo al que se escriben la salida estándar y la salida estándar de error de `guix-daemon`.

`http-proxy` (predeterminado: `#f`)
: La URL de los proxy HTTP y HTTPS que se usa para la descarga de derivaciones de salida fija y sustituciones.

  También es posible cambiar la pasarela del daemon en tiempo te ejecución con la acción `set-http-proxy`, la cual lo reinicia:

```
herd set-http-proxy guix-daemon http://localhost:8118
```

  Para desactivar el uso actual de una pasarela ejecute:

```
herd set-http-proxy guix-daemon
```

`tmpdir` (predeterminado: `#f`)
: Una ruta de directorio donde `guix-daemon` realiza las construcciones.

`environment` (default: `'()`)
: Environment variables to be set before starting the daemon, as a list of `key=value` strings.

`guix-extension` [Data Type]

This data type represents the parameters of the Guix build daemon that are extendable. This is the type of the object that must be used within a guix service extension. See Section 11.19.1 [Composición de servicios], page 649, for more information.

`authorized-keys` (predeterminadas: `'()`)
: A list of file-like objects where each element contains a public key.

`substitute-urls` (default: `'()`)
: A list of strings where each element is a substitute URL.

`build-machines` (default: `'()`)
: A list of gexps that evaluate to `build-machine` records or to a list of `build-machine` records. (see Section 2.2.2 [Configuración de delegación del daemon], page 7).

  Using this field, a service may add new build machines to receive builds offloaded by the daemon. This is useful for a service such as `hurd-vm-service-type`, which can make a GNU/Hurd virtual machine directly usable for offloading (see [hurd-vm], page 550).

`chroot-directories` (default: `'()`)
: A list of file-like objects or strings pointing to additional directories the build daemon can use.

`udev-service-type` [Variable]

Type of the service that runs udev, a service which populates the `/dev` directory dynamically, whose value is a `<udev-configuration>` object.

Since the file names for udev rules and hardware description files matter, the configuration items for rules and hardware cannot simply be plain file-like objects with the rules content, because the name would be ignored. Instead, they are directory file-like objects that contain optional rules in `lib/udev/rules.d` and optional hardware files in `lib/udev/hwdb.d`. This way, the service can be configured with whole packages from which to take rules and hwdb files.

The `udev-service-type` can be *extended* with file-like directories that respect this hierarchy. For convenience, the `udev-rule` and `file->udev-rule` can be used to construct udev rules, while `udev-hardware` and `file->udev-hardware` can be used to construct hardware description files.

In an `operating-system` declaration, this service type can be *extended* using procedures `udev-rules-service` and `udev-hardware-service`.

`udev-configuration` [Data Type]

Data type representing the configuration of udev.

`udev` (default: `eudev`) (type: file-like)
: Package object of the udev service. This package is used at run-time, when compiled for the target system. In order to generate the `hwdb.bin` hardware index, it is also used when generating the system definition, compiled for the current system.

`rules` (default: *'()*) (type: list-of-file-like)
: List of file-like objects denoting udev rule files under a sub-directory.

`hardware` (default: *'()*) (type: list-of-file-like)
: List of file-like objects denoting udev hardware description files under a sub-directory.

`udev-rule` *nombre-archivo contenido* [Procedimiento]

Devuelve un archivo de reglas de udev con nombre *nombre-archivo* que contiene las reglas definidas en el literal *contenido*.

En el ejemplo siguiente se define una regla para un dispositivo USB que será almacenada en el archivo `90-usb-cosa.rules`. Esta regla ejecuta un script cuando se detecta un dispositivo USB con un identificador de producto dado.

```
(define %regla-ejemplo-udev
  (udev-rule
    "90-usb-cosa.rules"
    (string-append "ACTION==\"add\", SUBSYSTEM==\"usb\", "
                   "ATTR{product}==\"Ejemplo\", "
                   "RUN+=\"/ruta/al/ejecutable\"")))
```

`udev-hardware` *file-name contents* [Procedure]
Return a udev hardware description file named *file-name* containing the hardware information *contents*.

`udev-rules-service` *name rules* [*#:groups '*()] [Procedure]
Return a service that extends `udev-service-type` with *rules* and `account-service-type` with *groups* as system groups. This works by creating a singleton service type *name*`-udev-rules`, of which the returned service is an instance.

A continuación se muestra cómo se puede usar para extender `udev-service-type` con la regla `%regla-ejemplo-udev` definida previamente.

```
(operating-system
 ;; ...
 (services
   (cons (udev-rules-service 'usb-thing %regla-ejemplo-udev)
         %desktop-services)))
```

`udev-hardware-service` *name hardware* [Procedure]
Return a service that extends `udev-service-type` with *hardware*. The service name is *name*`-udev-hardware`.

`file->udev-rule` *nombre-archivo archivo* [Procedimiento]
Return a udev-rule file named *file-name* containing the rules defined within *file*, a file-like object.

El ejemplo siguiente muestra cómo podemos usar un archivo de reglas existente.

```
(use-modules (guix download)     ;para url-fetch
             (guix packages)     ;para origin
             ...)

(define %reglas-android-udev
  (file->udev-rule
    "51-android-udev.rules"
    (let ((version "20170910"))
      (origin
       (method url-fetch)
       (uri (string-append "https://raw.githubusercontent.com/M0Rf30/"
                           "android-udev-rules/" version "/51-android.rules"))
       (sha256
        (base32 "0lmmagpyb6xsq6zcr2w1cyx9qmjqmajkvrdbhjx32gqf1d9is003"))))))
```

Since guix package definitions can be included in *rules* in order to use all their rules under the `lib/udev/rules.d` sub-directory, then in lieu of the previous *file->udev-rule* example, we could have used the *android-udev-rules* package which exists in Guix in the `(gnu packages android)` module.

`file->udev-hardware` *file-name file* [Procedure]
Return a udev hardware description file named *file-name* containing the rules defined within *file*, a file-like object.

El siguiente ejemplo muestra cómo usar el paquete *android-udev-rules* para que la herramienta de Android `adb` pueda detectar dispositivos sin privilegios de "root". También detalla como crear el grupo `adbusers`, el cual se requiere para el funcionamiento correcto de las reglas definidas dentro del paquete `android-udev-rules`. Para crear tal grupo, debemos definirlo tanto como parte de *supplementary-groups* de la declaración de nuestra cuenta de usuaria en *user-account*, así como en el parámetro *groups* del procedimiento `udev-rules-service`.

```
(use-modules (gnu packages android)  ;para android-udev-rules
             (gnu system shadow)     ;para user-group
             ...)

(operating-system
  ;; ...
  (users (cons (user-account
                ;; ...
                (supplementary-groups
                 '("adbusers"   ;for adb
                   "wheel" "netdev" "audio" "video")))))
  ;; ...
  (services
    (cons (udev-rules-service 'android android-udev-rules
                              #:groups '("adbusers"))
          %desktop-services)))
```

`urandom-seed-service-type` [Variable]
Almacena alguna entropía en `%random-seed-file` para alimentar `/dev/urandom` cuando se reinicia. También intenta alimentar `/dev/urandom` con `/dev/hwrng` durante el arranque, si `/dev/hwrng` existe y se tienen permisos de lectura.

`%random-seed-file` [Variable]
Es el nombre del archivo donde algunos bytes aleatorios son almacenados por el servicio *urandom-seed-service* para alimentar `/dev/urandom` durante el reinicio. Su valor predeterminado es `/var/lib/random-seed`.

`gpm-service-type` [Variable]
Este es el tipo de servicio que ejecuta GPM, el *daemon de ratón de propósito general*, que permite el uso del ratón en la consola Linux. GPM permite a las usuarias el uso del ratón en la consola, notablemente la selección, copia y pegado de texto.

El valor para servicios de este tipo debe ser un objeto `gpm-configuration` (véase a continuación). Este servicio no es parte de `%base-services`.

`gpm-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de GPM.

`opciones` (predeterminadas: `%default-gpm-options`)
: Opciones de línea de órdenes proporcionadas a `gpm`. El conjunto predeterminado de opciones instruye a `gpm` para esperar eventos de ratón en `/dev/input/mice`. See Section "Command Line" in *gpm manual*, para más información.

`gpm` (predeterminado: `gpm`)
: El paquete GPM usado.

`guix-publish-service-type` [Variable]

Este es el tipo de servicio para `guix publish` (see Section 9.11 [Invocación de guix publish], page 226). Su valor debe ser un objeto `guix-publish-configuration`, como se describe a continuación.

Se asume que `/etc/guix` ya contiene el par de claves de firma como `guix archive --generate-key` lo crea (see Section 5.11 [Invocación de guix archive], page 66). Si no es el caso, el servicio fallará al arrancar.

`guix-publish-configuration` [Tipo de datos]

Tipo de datos que representa la configuración del servicio `guix publish`.

`guix` (predeterminado: `guix`)
: El paquete Guix usado.

`port` (predeterminado: `80`)
: El puerto TCP en el que se esperan conexiones.

`host` (predeterminado: `"localhost"`)
: La dirección de red (y, por tanto, la interfaz de red) en la que se esperarán conexiones. Use `"0.0.0.0"` para aceptar conexiones por todas las interfaces de red.

`advertise?` (default: `#f`)
: When true, advertise the service on the local network *via* the DNS-SD protocol, using Avahi.

  This allows neighboring Guix devices with discovery on (see `guix-configuration` above) to discover this `guix publish` instance and to automatically download substitutes from it.

`compression` (default: `'(("gzip" 3) ("zstd" 3))`)
: Es una lista de tuplas método de compresión/nivel usadas para la compresión de sustituciones. Por ejemplo, para comprimir todas las sustituciones *tanto con* lzip a nivel 8 *como con* gzip a nivel 9, escriba:

  ```
  '(("lzip" 7) ("gzip" 9))
  ```

  Level 9 achieves the best compression ratio at the expense of increased CPU usage, whereas level 1 achieves fast compression. See Section 9.11 [Invocación de guix publish], page 226, for more information on the available compression methods and the tradeoffs involved.

  Una lista vacía desactiva completamente la compresión.

`nar-path` (predeterminado: `"nar"`)
: La ruta URL de la que se pueden obtener "nars". See Section 9.11 [Invocación de guix publish], page 226, para más detalles.

`cache` (predeterminado: `#f`)
: Cuando es `#f`, desactiva la caché y genera los archivos bajo demanda. De otro modo, debería ser el nombre de un directorio—por ejemplo, `"/var/cache/guix/publish"`—donde `guix pubish` almacena los archivos y metadatos en caché listos para ser enviados. See Section 9.11 [Invocación de guix publish], page 226, para más información sobre sus ventajas e inconvenientes.

`workers` (predeterminado: `#f`)
: Cuando es un entero, es el número de hilos de trabajo usados para la caché; cuando es `#f`, se usa el número de procesadores. See Section 9.11 [Invocación de guix publish], page 226, para más información.

`cache-bypass-threshold` (predeterminado: 10 MiB)
: Cuando `cache` es verdadero, su valor indica el tamaño máximo en bytes de un elemento del almacén hasta el cual `guix publish` puede ignorar un fallo de caché y realizar la petición directamente. See Section 9.11 [Invocación de guix publish], page 226, para obtener más información.

`ttl` (predeterminado: `#f`)
: Cuando es un entero, denota el *tiempo de vida* en segundos de los archivos publicados. See Section 9.11 [Invocación de guix publish], page 226, para más información.

`negative-ttl` (default: `#f`)
: When it is an integer, this denotes the *time-to-live* in seconds for the negative lookups. See Section 9.11 [Invocación de guix publish], page 226, for more information.

`rngd-service-type` [Variable]
: Type of the service that runs rng-tools rngd, whose value is an `<rngd-configuration>` object.

`rngd-configuration` [Data Type]
: Data type representing the configuration of rngd.

  `rng-tools` (default: `rng-tools`) (type: file-like)
  : Package object of the rng-tools rngd.

  `device` (default: `"/dev/hwrng"`) (type: string)
  : Path of the device to add to the kernel's entropy pool. The service will fail if *device* does not exist.

`pam-limits-service-type` [Variable]
: Type of the service that installs a configuration file for the `pam_limits` module (`http://linux-pam.org/Linux-PAM-html/sag-pam_limits.html`). The value for this service type is a list of `pam-limits-entry` values, which can be used to specify

`ulimit` limits and `nice` priority limits to user sessions. By default, the value is the empty list.

Las siguientes definiciones de límites establecen dos límites "hard" y "soft" para todas las sesiones de ingreso al sistema de usuarias pertenecientes al grupo `realtime`:

```
(service pam-limits-service-type
         (list
          (pam-limits-entry "@realtime" 'both 'rtprio 99)
          (pam-limits-entry "@realtime" 'both 'memlock 'unlimited)))
```

La primera entrada incrementa la prioridad máxima de tiempo real para procesos sin privilegios; la segunda entrada elimina cualquier restricción sobre el espacio de direcciones que puede bloquearse en memoria. Estas configuraciones se usan habitualmente para sistemas de sonido en tiempo real.

Another useful example is raising the maximum number of open file descriptors that can be used:

```
(service pam-limits-service-type
         (list
          (pam-limits-entry "*" 'both 'nofile 100000)))
```

In the above example, the asterisk means the limit should apply to any user. It is important to ensure the chosen value doesn't exceed the maximum system value visible in the `/proc/sys/fs/file-max` file, else the users would be prevented from login in. For more information about the Pluggable Authentication Module (PAM) limits, refer to the '`pam_limits`' man page from the `linux-pam` package.

`greetd-service-type` [Variable]

`greetd` (`https://git.sr.ht/~kennylevinsen/greetd`) is a minimal and flexible login manager daemon, that makes no assumptions about what you want to launch.

If you can run it from your shell in a TTY, greetd can start it. If it can be taught to speak a simple JSON-based IPC protocol, then it can be a geeter.

`greetd-service-type` provides necessary infrastructure for logging in users, including:

- `greetd` PAM service
- Special variation of `pam-mount` to mount `XDG_RUNTIME_DIR`

Here is an example of switching from `mingetty-service-type` to `greetd-service-type`, and how different terminals could be:

```
(append
 (modify-services %base-services
   ;; greetd-service-type provides "greetd" PAM service
   (delete login-service-type)
   ;; and can be used in place of mingetty-service-type
   (delete mingetty-service-type))
 (list
  (service greetd-service-type
           (greetd-configuration
            (terminals
```

```
                        (list
                         ;; we can make any terminal active by default
                         (greetd-terminal-configuration (terminal-vt "1") (terminal-switch
                         ;; we can make environment without XDG_RUNTIME_DIR set
                         ;; even provide our own environment variables
                         (greetd-terminal-configuration
                          (terminal-vt "2")
                          (default-session-command
                            (greetd-agreety-session
                             (extra-env '(("MY_VAR" . "1")))
                             (xdg-env? #f))))
                         ;; we can use different shell instead of default bash
                         (greetd-terminal-configuration
                          (terminal-vt "3")
                          (default-session-command
                            (greetd-agreety-session (command (file-append zsh "/bin/zsh"))
                         ;; we can use any other executable command as greeter
                         (greetd-terminal-configuration
                          (terminal-vt "4")
                          (default-session-command (program-file "my-noop-greeter" #~(exit
                         (greetd-terminal-configuration (terminal-vt "5"))
                         (greetd-terminal-configuration (terminal-vt "6"))))))
                ;; mingetty-service-type can be used in parallel
                ;; if needed to do so, do not (delete login-service-type)
                ;; as illustrated above
                #| (service mingetty-service-type (mingetty-configuration (tty "tty8"))) |#))
```

`greetd-configuration` [Data Type]

Configuration record for the `greetd-service-type`.

`motd` Un objeto tipo-archivo que contiene el "mensaje del día".

`allow-empty-passwords?` (predeterminado: `#t`)
Permite contraseñas vacías por defecto para que las primeras usuarias puedan ingresar en el sistema cuando la cuenta de "root" está recién creada.

`terminals` (default: `'()`)
List of `greetd-terminal-configuration` per terminal for which `greetd` should be started.

`greeter-supplementary-groups` (default: `'()`)
List of groups which should be added to `greeter` user. For instance:

```
(greeter-supplementary-groups '("seat" "video"))
```

Note that this example will fail if `seat` group does not exist.

`greetd-terminal-configuration` [Data Type]

Configuration record for per terminal greetd daemon service.

`greetd` (default: `greetd`)
The greetd package to use.

`config-file-name`
: Configuration file name to use for greetd daemon. Generally, autogenerated derivation based on `terminal-vt` value.

`log-file-name`
: Log file name to use for greetd daemon. Generally, autogenerated name based on `terminal-vt` value.

`terminal-vt` (default: ‘`"7"`’)
: The VT to run on. Use of a specific VT with appropriate conflict avoidance is recommended.

`terminal-switch` (default: `#f`)
: Make this terminal active on start of `greetd`.

`source-profile?` (default: `#t`)
: Whether to source `/etc/profile` and `~/.profile`, when they exist.

`default-session-user` (default: ‘`"greeter"`’)
: The user to use for running the greeter.

`default-session-command` (default: `(greetd-agreety-session)`)
: Can be either instance of `greetd-agreety-session` configuration or `gexp->script` like object to use as greeter.

`greetd-agreety-session` [Data Type]

Configuration record for the agreety greetd greeter.

`agreety` (default: `greetd`)
: The package with `/bin/agreety` command.

`command` (default: `(file-append bash "/bin/bash")`)
: Command to be started by `/bin/agreety` on successful login.

`command-args` (default: `'("-l")`)
: Command arguments to pass to command.

`extra-env` (default: `'()`)
: Extra environment variables to set on login.

`xdg-env?` (default: `#t`)
: If true `XDG_RUNTIME_DIR` and `XDG_SESSION_TYPE` will be set before starting command. One should note that, `extra-env` variables are set right after mentioned variables, so that they can be overridden.

`greetd-wlgreet-session` [Data Type]

Generic configuration record for the wlgreet greetd greeter.

`wlgreet` (default: `wlgreet`)
: The package with the `/bin/wlgreet` command.

`command` (default: `(file-append sway "/bin/sway")`)
: Command to be started by `/bin/wlgreet` on successful login.

`command-args` (default: `'()`)
: Command arguments to pass to command.

`output-mode` (default: `"all"`)
: Option to use for `outputMode` in the TOML configuration file.

`scale` (default: `1`)
: Option to use for `scale` in the TOML configuration file.

`background` (default: `'(0 0 0 0.9)`)
: RGBA list to use as the background colour of the login prompt.

`headline` (default: `'(1 1 1 1)`)
: RGBA list to use as the headline colour of the UI popup.

`prompt` (default: `'(1 1 1 1)`)
: RGBA list to use as the prompt colour of the UI popup.

`prompt-error` (default: `'(1 1 1 1)`)
: RGBA list to use as the error colour of the UI popup.

`border` (default: `'(1 1 1 1)`)
: RGBA list to use as the border colour of the UI popup.

`extra-env` (default: `'()`)
: Extra environment variables to set on login.

`greetd-wlgreet-sway-session` [Data Type]

Sway-specific configuration record for the wlgreet greetd greeter.

`wlgreet-session` (default: `(greetd-wlgreet-session)`)
: A `greetd-wlgreet-session` record for generic wlgreet configuration, on top of the Sway-specific `greetd-wlgreet-sway-session`.

`sway` (default: `sway`)
: The package providing the `/bin/sway` command.

`sway-configuration` (default: #f)
: File-like object providing an additional Sway configuration file to be prepended to the mandatory part of the configuration.

Here is an example of a greetd configuration that uses wlgreet and Sway:

```
(greetd-configuration
 ;; We need to give the greeter user these permissions, otherwise
 ;; Sway will crash on launch.
 (greeter-supplementary-groups (list "video" "input" "seat"))
 (terminals
  (list (greetd-terminal-configuration
         (terminal-vt "1")
         (terminal-switch #t)
         (default-session-command
          (greetd-wlgreet-sway-session
           (sway-configuration
            (local-file "sway-greetd.conf")))))))))
```

### 11.10.2 Ejecución de tareas programadas

El módulo `(gnu services mcron)` proporciona una interfaz a GNU mcron, un daemon para ejecutar trabajos planificados de antemano (see *GNU mcron*). GNU mcron es similar al daemon tradicional de Unix `cron`; la principal diferencia es que está implementado en Scheme Guile, que proporciona mucha flexibilidad cuando se especifica la planificación de trabajos y sus acciones.

El siguiente ejemplo define un sistema operativo que ejecuta las órdenes `updatedb` (see Section "Invoking updatedb" in *Finding Files*) y `guix gc` (see Section 5.6 [Invocación de guix gc], page 53) de manera diaria, así como la orden `mkid` como una usuaria sin privilegios (see Section "mkid invocation" in *ID Database Utilitites*). Usa expresiones-g para introducir definiciones de trabajos que serán proporcionados a mcron (see Section 8.12 [Expresiones-G], page 167).

```
(use-modules (guix) (gnu) (gnu services mcron))
(use-package-modules base idutils)

(define updatedb-job
  ;; Run 'updatedb' at 3AM every day.  Here we write the
  ;; job's action as a Scheme procedure.
  #~(job '(next-hour '(3))
         (lambda ()
           (system* (string-append #$findutils "/bin/updatedb")
                    "--prunepaths=/tmp /var/tmp /gnu/store"))
         "updatedb"))

(define trabajo-recolector-basura
  ;; Recolecta basura 5 minutos después de media noche,
  ;; todos los días. La acción del trabajo es una orden
  ;; del shell.
  #~(job "5 0 * * *"            ;sintaxis de Vixie cron
         "guix gc -F 1G"))

(define trabajo-idutils
  ;; Actualiza el índice de la base de datos como "carlos" a las
  ;; 12:15 y a las 19:15.  Esto se ejecuta desde su directorio.
  #~(job '(next-minute-from (next-hour '(12 19)) '(15))
         (string-append #$idutils "/bin/mkid src")
         #:user "carlos"))

(operating-system
  ;; ...

  ;; %BASE-SERVICES already includes an instance of
  ;; 'mcron-service-type', which we extend with additional
  ;; jobs using 'simple-service'.
  (services (cons (simple-service 'my-cron-jobs
                                  mcron-service-type
```

```
                                    (list garbage-collector-job
                                          updatedb-job
                                          idutils-job))
                    %base-services)))
```

**Tip:** When providing the action of a job specification as a procedure, you should provide an explicit name for the job via the optional 3rd argument as done in the `updatedb-job` example above. Otherwise, the job would appear as "Lambda function" in the output of `herd schedule mcron`, which is not nearly descriptive enough!

**Tip:** Avoid calling the Guile procedures `execl`, `execle` or `execlp` inside a job specification, else mcron won't be able to output the completion status of the job.

Para trabajos más complejos definidos en Scheme donde necesita control en el ámbito global, por ejemplo para introducir una forma `use-modules`, puede mover su código a un programa separado usando el procedimiento `program-file` del módulo `(guix gexp)` (see Section 8.12 [Expresiones-G], page 167). El siguiente ejemplo ilustra este caso.

```
(define %tarea-alerta-bateria
  ;; Pita cuando la carga de la batería es inferior a %CARGA-MIN
  #~(job
     '(next-minute (range 0 60 1))
     #$(program-file
        "alerta-batería.scm"
        (with-imported-modules (source-module-closure
                                '((guix build utils)))
          #~(begin
              (use-modules (guix build utils)
                           (ice-9 popen)
                           (ice-9 regex)
                           (ice-9 textual-ports)
                           (srfi srfi-2))
              (define %carga-min 20)

              (setenv "LC_ALL" "C")     ;Procesado de cadenas en inglés
              (and-let* ((entrada (open-pipe*
                                   OPEN_READ
                                   #$(file-append acpi "/bin/acpi")))
                         (salida (get-string-all entrada))
                         (m (string-match "Discharging, ([0-9]+)%" output))
                         (carga (string->number (match:substring m 1)))
                         ((< carga %carga-min)))
                (setenv "LC_ALL" "")     ;Mensaje de salida traducido
                (format #t "aviso: La carga de la batería es baja (~a%)~%"
                        carga)
                (invoke #$(file-append beep "/bin/beep") "-r5")))))))
```

See Section "Guile Syntax" in *GNU mcron*, para más información sobre las especificaciones de trabajos de mcron. A continuación se encuentra la referencia del servicio mcron.

En un sistema en ejecución puede usar la acción `schedule` del servicio para visualizar los siguientes trabajos mcron que se ejecutarán:

```
# herd schedule mcron
```

El ejemplo previo enumera las siguientes cinco tareas que se ejecutarán, pero también puede especificar el número de tareas a mostrar:

```
# herd schedule mcron 10
```

`mcron-service-type` [Variable]

Este es el tipo del servicio `mcron`, cuyo valor es un objeto `mcron-configuration`.

This service type can be the target of a service extension that provides additional job specifications (see Section 11.19.1 [Composición de servicios], page 649). In other words, it is possible to define services that provide additional mcron jobs to run.

`mcron-configuration` [Tipo de datos]

Available `mcron-configuration` fields are:

`mcron` (default: `mcron`) (type: file-like)
El paquete mcron usado.

`jobs` (default: `'()`) (type: list-of-gexps)
This is a list of gexps (see Section 8.12 [Expresiones-G], page 167), where each gexp corresponds to an mcron job specification (see Section "Syntax" in *GNU mcron*).

`log?` (default: `#t`) (type: boolean)
Log messages to standard output.

`log-file` (default: `"/var/log/mcron.log"`) (type: string)
Log file location.

`log-format` (default: `"~1@*~a ~a: ~a~%"`) (type: string)
`(ice-9 format)` format string for log messages. The default value produces messages like '*pid name*: *message*' (see Section "Invoking mcron" in *GNU mcron*). Each message is also prefixed by a timestamp by GNU Shepherd.

`date-format` (type: maybe-string)
`(srfi srfi-19)` format string for date.

### 11.10.3 Rotación del registro de mensajes

Los archivos de registro como los encontrados en `/var/log` tienden a crecer indefinidamente, de modo que es buena idea *llevar a cabo una rotación* de vez en cuando—es decir, archivar su contenido en archivos distintos, posiblemente comprimidos. El módulo `(gnu services admin)` proporciona una interfaz con GNU Rot[t]log, una herramienta de rotación de registros (see *GNU Rot[t]log Manual*).

Este servicio es parte de `%base-services`, y por lo tanto se activa de manera predeterminada, con la configuración predeterminada, para archivos de registro que se pueden encontrar habitualmente. El siguiente ejemplo muestra como extenderlo con una *rotación*

adicional, en caso de que deba hacerlo (habitualmente los servicios que producen archivos de registro ya lo hacen ellos mismos):

```
(use-modules (guix) (gnu))
(use-service-modules admin)

(define mis-archivos-de-registro
  ;; Archivos que deseo rotar.
  '("/var/log/un-archivo.log" "/var/log/otro.log"))

(operating-system
  ;; ...
  (services (cons (simple-service 'rota-mis-cosas
                                  rottlog-service-type
                                  (list (log-rotation
                                         (frequency 'daily)
                                         (files mis-archivos-de-registro))))
                  %base-services)))
```

`rottlog-service-type` [Variable]

Este es el tipo del servicio Rottlog, cuyo valor es un objeto `rottlog-configuration`.

Otros servicios pueden extenderlo con nuevos objetos `log-rotation` (véase a continuación), aumentando de dicho modo el conjunto de archivos a rotar.

Este servicio puede definir trabajos de mcron (see Section 11.10.2 [Ejecución de tareas programadas], page 297) para ejecutar el servicio rottlog.

`rottlog-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de rottlog.

`rottlog` (predeterminado: `rottlog`)
: El paquete Rottlog usado.

`rc-file` (predeterminado: `(file-append rottlog "/etc/rc")`)
: El archivo de configuración de Rottlog usado (see Section "Mandatory RC Variables" in *GNU Rot[t]log Manual*).

`rotations` (predeterminadas: `%default-rotations`)
: Una lista de objetos `log-rotation` como se define a continuación.

`jobs` Esta es una lista de expresiones-G donde cada expresión-G corresponde a una especificación de trabajo de mcron (see Section 11.10.2 [Ejecución de tareas programadas], page 297).

`log-rotation` [Tipo de datos]

Tipo de datos que representa la rotación de un grupo de archivos de log.

Tomando el ejemplo del manual de Rottlog (see Section "Period Related File Examples" in *GNU Rot[t]log Manual*), una rotación de registros se podría definir de esta manera:

```
(log-rotation
  (frequency 'daily)
```

```
        (files '("/var/log/apache/*"))
        (options '("storedir apache-archives"
                   "rotate 6"
                   "notifempty"
                   "nocompress")))
```

La lista de campos es como sigue:

`frequency` (predeterminada: `'weekly`)
: La frecuencia de rotación de logs, un símbolo.

`files`
: La lista de archivos o patrones extendidos de archivo a rotar.

`options` (default: `%default-log-rotation-options`)
: The list of rottlog options for this rotation (see Section "Configuration parameters" in *GNU Rot[t]log Manual*).

`post-rotate` (predeterminado: `#f`)
: O bien `#f`, o bien una expresión-G que se ejecutará una vez la rotación se haya completado.

`%default-rotations` [Variable]
Especifica la rotación semanal de `%rotated-files` y de `/var/log/guix-daemon.log`.

`%rotated-files` [Variable]
La lista de archivos controlados por syslog que deben ser rotados. De manera predeterminada es `'("/var/log/messages" "/var/log/secure" "/var/log/maillog")`.

Some log files just need to be deleted periodically once they are old, without any other criterion and without any archival step. This is the case of build logs stored by `guix-daemon` under `/var/log/guix/drvs` (see Section 2.3 [Invocación de guix-daemon], page 12). The `log-cleanup` service addresses this use case. For example, `%base-services` (see Section 11.10.1 [Servicios base], page 276) includes the following:

```
;; Periodically delete old build logs.
(service log-cleanup-service-type
         (log-cleanup-configuration
          (directory "/var/log/guix/drvs")))
```

That ensures build logs do not accumulate endlessly.

`log-cleanup-service-type` [Variable]
This is the type of the service to delete old logs. Its value must be a `log-cleanup-configuration` record as described below.

`log-cleanup-configuration` [Data Type]
Data type representing the log cleanup configuration

`directory`
: Name of the directory containing log files.

`expiry` (default: `(* 6 30 24 3600)`)
: Age in seconds after which a file is subject to deletion (six months by default).

`schedule` (default: `"30 12 01,08,15,22 * *"`)
: String or gexp denoting the corresponding mcron job schedule (see Section 11.10.2 [Ejecución de tareas programadas], page 297).

## Anonip Service

Anonip is a privacy filter that removes IP address from web server logs. This service creates a FIFO and filters any written lines with anonip before writing the filtered log to a target file.

The following example sets up the FIFO `/var/run/anonip/https.access.log` and writes the filtered log file `/var/log/anonip/https.access.log`.

```
(service anonip-service-type
         (anonip-configuration
           (input  "/var/run/anonip/https.access.log")
           (output "/var/log/anonip/https.access.log")))
```

Configure your web server to write its logs to the FIFO at `/var/run/anonip/https.access.log` and collect the anonymized log file at `/var/web-logs/https.access.log`.

`anonip-configuration` [Data Type]
This data type represents the configuration of anonip. It has the following parameters:

`anonip` (default: `anonip`)
: The anonip package to use.

`input`
: The file name of the input log file to process. The service creates a FIFO of this name. The web server should write its logs to this FIFO.

`output`
: The file name of the processed log file.

The following optional settings may be provided:

`skip-private?`
: When `#true` do not mask addresses in private ranges.

`column`
: A 1-based indexed column number. Assume IP address is in the specified column (default is 1).

`replacement`
: Replacement string in case address parsing fails, e.g. `"0.0.0.0"`.

`ipv4mask`
: Number of bits to mask in IPv4 addresses.

`ipv6mask`
: Number of bits to mask in IPv6 addresses.

`increment`
: Increment the IP address by the given number. By default this is zero.

`delimiter`
: Log delimiter string.

`regex`
: Regular expression for detecting IP addresses. Use this instead of `column`.

### 11.10.4 Networking Setup

The `(gnu services networking)` module provides services to configure network interfaces and set up networking on your machine. Those services provide different ways for you to set up your machine: by declaring a static network configuration, by running a Dynamic Host Configuration Protocol (DHCP) client, or by running daemons such as NetworkManager and Connman that automate the whole process, automatically adapt to connectivity changes, and provide a high-level user interface.

On a laptop, NetworkManager and Connman are by far the most convenient options, which is why the default desktop services include NetworkManager (see Section 11.10.9 [Servicios de escritorio], page 363). For a server, or for a virtual machine or a container, static network configuration or a simple DHCP client are often more appropriate.

This section describes the various network setup services available, starting with static network configuration.

`static-networking-service-type` [Variable]

This is the type for statically-configured network interfaces. Its value must be a list of `static-networking` records. Each of them declares a set of *addresses*, *routes*, and *links*, as shown below.

Here is the simplest configuration, with only one network interface controller (NIC) and only IPv4 connectivity:

```
;; Static networking for one NIC, IPv4-only.
(service static-networking-service-type
         (list (static-networking
                (addresses
                 (list (network-address
                        (device "eno1")
                        (value "10.0.2.15/24"))))
                (routes
                 (list (network-route
                        (destination "default")
                        (gateway "10.0.2.2"))))
                (name-servers '("10.0.2.3")))))
```

The snippet above can be added to the `services` field of your operating system configuration (see Section 11.2 [Uso de la configuración del sistema], page 244). It will configure your machine to have 10.0.2.15 as its IP address, with a 24-bit netmask for the local network—meaning that any 10.0.2.*x* address is on the local area network (LAN). Traffic to addresses outside the local network is routed *via* 10.0.2.2. Host names are resolved by sending domain name system (DNS) queries to 10.0.2.3.

`static-networking` [Data Type]

This is the data type representing a static network configuration.

As an example, here is how you would declare the configuration of a machine with a single network interface controller (NIC) available as `eno1`, and with one IPv4 and one IPv6 address:

```
;; Network configuration for one NIC, IPv4 + IPv6.
(static-networking
```

```
(addresses (list (network-address
                  (device "eno1")
                  (value "10.0.2.15/24"))
                 (network-address
                  (device "eno1")
                  (value "2001:123:4567:101::1/64"))))
(routes (list (network-route
               (destination "default")
               (gateway "10.0.2.2"))
              (network-route
               (destination "default")
               (gateway "2020:321:4567:42::1"))))
(name-servers '("10.0.2.3")))
```

If you are familiar with the `ip` command of the `iproute2` package (`https://wiki.linuxfoundation.org/networking/iproute2`) found on Linux-based systems, the declaration above is equivalent to typing:

```
ip address add 10.0.2.15/24 dev eno1
ip address add 2001:123:4567:101::1/64 dev eno1
ip route add default via inet 10.0.2.2
ip route add default via inet6 2020:321:4567:42::1
```

Run `man 8 ip` for more info. Venerable GNU/Linux users will certainly know how to do it with `ifconfig` and `route`, but we'll spare you that.

The available fields of this data type are as follows:

`addresses`
`links` (default: `'()`)
`routes` (default: `'()`)
: The list of `network-address`, `network-link`, and `network-route` records for this network (see below).

`name-servers` (default: `'()`)
: The list of IP addresses (strings) of domain name servers. These IP addresses go to `/etc/resolv.conf`.

`provision` (default: `'(networking)`)
: If true, this should be a list of symbols for the Shepherd service corresponding to this network configuration.

`requirement` (default `'()`)
: The list of Shepherd services depended on.

`network-address` [Data Type]

This is the data type representing the IP address of a network interface.

`device` The name of the network interface for this address—e.g., `"eno1"`.

`value` The actual IP address and network mask, in CIDR (Classless Inter-Domain Routing) notation (`https://en.wikipedia.org/wiki/CIDR#CIDR_notation`), as a string.

For example, `"10.0.2.15/24"` denotes IPv4 address 10.0.2.15 on a 24-bit sub-network—all 10.0.2.*x* addresses are on the same local network.

`ipv6?` Whether `value` denotes an IPv6 address. By default this is automatically determined.

`network-route` [Data Type]

This is the data type representing a network route.

`destination`
The route destination (a string), either an IP address and network mask or `"default"` to denote the default route.

`source` (default: `#f`)
The route source.

`device` (predeterminado: `#f`)
The device used for this route—e.g., `"eno2"`.

`ipv6?` (default: auto)
Whether this is an IPv6 route. By default this is automatically determined based on `destination` or `gateway`.

`gateway` (default: `#f`)
IP address (a string) through which traffic is routed.

`network-link` [Data Type]

Data type for a network link (see Section "Link" in *Guile-Netlink Manual*). During startup, network links are employed to construct or modify existing or virtual ethernet links. These ethernet links can be identified by their *name* or *mac-address*. If there is a need to create virtual interface, *name* and *type* fields are required.

`name` The name of the link—e.g., `"v0p0"` (default: `#f`).

`type` A symbol denoting the type of the link—e.g., `'veth` (default: `#f`).

`mac-address`
The mac-address of the link—e.g., `"98:11:22:33:44:55"` (default: `#f`).

`arguments`
List of arguments for this type of link.

Consider a scenario where a server equipped with a network interface which has multiple ports. These ports are connected to a switch, which supports link aggregation (`https://en.wikipedia.org/wiki/Link_aggregation`) (also known as bonding or NIC teaming). The switch uses port channels to consolidate multiple physical interfaces into one logical interface to provide higher bandwidth, load balancing, and link redundancy. When a port is added to a LAG (or link aggregation group), it inherits the properties of the port-channel. Some of these properties are VLAN membership, trunk status, and so on.

VLAN (`https://en.wikipedia.org/wiki/Virtual_LAN`) (or virtual local area network) is a logical network that is isolated from other VLANs on the same physical network. This can be used to segregate traffic, improve security, and simplify network management.

With all that in mind let's configure our static network for the server. We will bond two existing interfaces together using 802.3ad schema and on top of it, build a VLAN interface with id 1055. We assign a static ip to our new VLAN interface.

```
(static-networking
 (links (list (network-link
               (name "bond0")
               (type 'bond)
               (arguments '((mode . "802.3ad")
                            (miimon . 100)
                            (lacp-active . "on")
                            (lacp-rate . "fast"))))

              (network-link
               (mac-address "98:11:22:33:44:55")
               (arguments '((master . "bond0"))))

              (network-link
               (mac-address "98:11:22:33:44:56")
               (arguments '((master . "bond0"))))

              (network-link
               (name "bond0.1055")
               (type 'vlan)
               (arguments '((id . 1055)
                            (link . "bond0"))))))
 (addresses (list (network-address
                   (value "192.168.1.4/24")
                   (device "bond0.1055")))))
```

`%loopback-static-networking` [Variable]

This is the `static-networking` record representing the "loopback device", `lo`, for IP addresses 127.0.0.1 and ::1, and providing the `loopback` Shepherd service.

`%qemu-static-networking` [Variable]

This is the `static-networking` record representing network setup when using QEMU's user-mode network stack on `eth0` (see Section "Using the user mode network stack" in *QEMU Documentation*).

`dhcp-client-service-type` [Variable]

This is the type of services that run *dhcp*, a Dynamic Host Configuration Protocol (DHCP) client.

`dhcp-client-configuration` [Data Type]

Data type representing the configuration of the DHCP client service.

`package` (predeterminado: `isc-dhcp`)
DHCP client package to use.

`interfaces` (default: `'all`)
: Either `'all` or the list of interface names that the DHCP client should listen on—e.g., `'("eno1")`.

  When set to `'all`, the DHCP client listens on all the available non-loopback interfaces that can be activated. Otherwise the DHCP client listens only on the specified interfaces.

`shepherd-requirement` (default: `'()`)
`shepherd-provision` (default: `'(networking)`)
: This option can be used to provide a list of symbols naming Shepherd services that this service will depend on, such as `'wpa-supplicant` or `'iwd` if you require authenticated access for encrypted WiFi or Ethernet networks.

  Likewise, `shepherd-provision` is a list of Shepherd service names (symbols) provided by this service. You might want to change the default value if you intend to run several DHCP clients, only one of which provides the `networking` Shepherd service.

`network-manager-service-type` [Variable]
Este es el tipo de servicio para el servicio NetworkManager (`https://wiki.gnome.org/Projects/NetworkManager`). El valor para este tipo de servicio es un registro `network-manager-configuration`.

Este servicio es parte de `%desktop-services` (see Section 11.10.9 [Servicios de escritorio], page 363).

`network-manager-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de NetworkManager.

`network-manager` (predeterminado: `network-manager`)
: El paquete de NetworkManager usado.

`shepherd-requirement` (default: `'(wpa-supplicant)`)
: This option can be used to provide a list of symbols naming Shepherd services that this service will depend on, such as `'wpa-supplicant` or `'iwd` if you require authenticated access for encrypted WiFi or Ethernet networks.

`dns` (predeterminado: `"default"`)
: Modo de procesamiento para DNS, que afecta la manera en la que NetworkManager usa el archivo de configuración `resolv.conf`.

  ‘`default`’ NetworkManager actualizará `resolv.conf` para reflejar los servidores de nombres proporcionados por las conexiones activas actualmente.

  ‘`dnsmasq`’ NetworkManager ejecutará `dnsmasq` como una caché local del servicio de nombres, mediante un *reenvío condicional* si se encuentra conectada a una VPN, y actualiza posteriormente `resolv.conf` para apuntar al servidor de nombres local.

  Con esta configuración puede compartir su conexión de red. Por ejemplo, cuando desee compartir su conexión de red a

otro equipo a través de un cable Ethernet, puede abrir `nm-connection-editor` y configurar el método de la conexión cableada para IPv4 y IPv6 "Compartida con otros equipos" y restablecer la conexión (o reiniciar).

También puede configurar una *conexión anfitrión-invitado* a las máquinas virtuales de QEMU (see Section 3.8 [Instalación de Guix en una máquina virtual], page 30). Con una conexión anfitrión-invitado puede, por ejemplo, acceder a un servidor web que se ejecute en la máquina virtual (see Section 11.10.20 [Servicios Web], page 469) desde un navegador web en su sistema anfitrión, o conectarse a la máquina virtual a través de SSH (see Section 11.10.5 [Servicios de red], page 314). Para configurar una conexión anfitrión-invitado, ejecute esta orden una única vez:

```
nmcli connection add type tun \
 connection.interface-name tap0 \
 tun.mode tap tun.owner $(id -u) \
 ipv4.method shared \
 ipv4.addresses 172.28.112.1/24
```

Cada vez que arranque su máquina virtual de QEMU (see Section 11.18 [Ejecutar Guix en una máquina virtual], page 647), proporcione `-nic tap,ifname=tap0,script=no,downscript=no` a `qemu-system-...`.

'`none`' NetworkManager no modificará `resolv.conf`.

`vpn-plugins` (predeterminados: `'()`)
Esta es la lista de módulos disponibles para redes privadas virtuales (VPN). Un ejemplo es el paquete `network-manager-openvpn`, que permite a NetworkManager la gestión de redes VPN a través de OpenVPN.

`connman-service-type` [Variable]
Este es el tipo de servicio para la ejecución de Connman (`https://01.org.connman`), un gestor de conexiones de red.

Su valor debe ser un registro `connman-configuration` como en este ejemplo:

```
(service connman-service-type
         (connman-configuration
           (disable-vpn? #t)))
```

Véase a continuación más detalles sobre `connman-configuration`.

`connman-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de connman.

`connman` (predeterminado: *connman*)
El paquete connman usado.

`shepherd-requirement` (default: `'()`)
: This option can be used to provide a list of symbols naming Shepherd services that this service will depend on, such as `'wpa-supplicant` or `'iwd` if you require authenticated access for encrypted WiFi or Ethernet networks.

`disable-vpn?` (predeterminado: `#f`)
: Cuando es verdadero, desactiva el módulo vpn de connman.

`general-configuration` (default: `(connman-general-configuration)`)
: Configuration serialized to `main.conf` and passed as `--config` to `connmand`.

`connman-general-configuration` [Data Type]

Available `connman-general-configuration` fields are:

`input-request-timeout` (type: maybe-number)
: Set input request timeout. Default is 120 seconds. The request for inputs like passphrase will timeout after certain amount of time. Use this setting to increase the value in case of different user interface designs.

`browser-launch-timeout` (type: maybe-number)
: Set browser launch timeout. Default is 300 seconds. The request for launching a browser for portal pages will timeout after certain amount of time. Use this setting to increase the value in case of different user interface designs.

`background-scanning?` (type: maybe-boolean)
: Enable background scanning. Default is true. If wifi is disconnected, the background scanning will follow a simple back off mechanism from 3s up to 5 minutes. Then, it will stay in 5 minutes unless user specifically asks for scanning through a D-Bus call. If so, the mechanism will start again from 3s. This feature activates also the background scanning while being connected, which is required for roaming on wifi. When `background-scanning?` is false, ConnMan will not perform any scan regardless of wifi is connected or not, unless it is requested by the user through a D-Bus call.

`use-gateways-as-timeservers?` (type: maybe-boolean)
: Assume that service gateways also function as timeservers. Default is false.

`fallback-timeservers` (type: maybe-list)
: List of Fallback timeservers. These timeservers are used for NTP sync when there are no timeservers set by the user or by the service, and when `use-gateways-as-timeservers?` is `#f`. These can contain a mixed combination of fully qualified domain names, IPv4 and IPv6 addresses.

`fallback-nameservers` (type: maybe-list)
: List of fallback nameservers appended to the list of nameservers given by the service. The nameserver entries must be in numeric format, host names are ignored.

`default-auto-connect-technologies` (type: maybe-list)
: List of technologies that are marked autoconnectable by default. The default value for this entry when empty is `"ethernet"`, `"wifi"`, `"cellular"`. Services that are automatically connected must have been set up and saved to storage beforehand.

`default-favourite-technologies` (type: maybe-list)
: List of technologies that are marked favorite by default. The default value for this entry when empty is `"ethernet"`. Connects to services from this technology even if not setup and saved to storage.

`always-connected-technologies` (type: maybe-list)
: List of technologies which are always connected regardless of preferred-technologies setting (`auto-connect? #t`). The default value is empty and this feature is disabled unless explicitly enabled.

`preferred-technologies` (type: maybe-list)
: List of preferred technologies from the most preferred one to the least preferred one. Services of the listed technology type will be tried one by one in the order given, until one of them gets connected or they are all tried. A service of a preferred technology type in state 'ready' will get the default route when compared to another preferred type further down the list with state 'ready' or with a non-preferred type; a service of a preferred technology type in state 'online' will get the default route when compared to either a non-preferred type or a preferred type further down in the list.

`network-interface-blacklist` (type: maybe-list)
: List of blacklisted network interfaces. Found interfaces will be compared to the list and will not be handled by ConnMan, if their first characters match any of the list entries. Default value is `"vmnet"`, `"vboxnet"`, `"virbr"`, `"ifb"`.

`allow-hostname-updates?` (type: maybe-boolean)
: Allow ConnMan to change the system hostname. This can happen for example if we receive DHCP hostname option. Default value is `#t`.

`allow-domainname-updates?` (type: maybe-boolean)
: Allow connman to change the system domainname. This can happen for example if we receive DHCP domainname option. Default value is `#t`.

`single-connected-technology?` (type: maybe-boolean)
: Keep only a single connected technology at any time. When a new service is connected by the user or a better one is found according to preferred-technologies, the new service is kept connected and all the other previously connected services are disconnected. With this setting it does not matter whether the previously connected services are in 'online' or 'ready' states, the newly connected service is the only one that will be kept connected. A service connected by the user will be used until going out of network coverage. With this setting enabled applications will notice more

network breaks than normal. Note this options can't be used with VPNs. Default value is `#f`.

`tethering-technologies` (type: maybe-list)
: List of technologies that are allowed to enable tethering. The default value is `"wifi"`, `"bluetooth"`, `"gadget"`. Only those technologies listed here are used for tethering. If one wants to tether ethernet, then add `"ethernet"` in the list. Note that if ethernet tethering is enabled, then a DHCP server is started on all ethernet interfaces. Tethered ethernet should never be connected to corporate or home network as it will disrupt normal operation of these networks. Due to this ethernet is not tethered by default. Do not activate ethernet tethering unless you really know what you are doing.

`persistent-tethering-mode?` (type: maybe-boolean)
: Restore earlier tethering status when returning from offline mode, re-enabling a technology, and after restarts and reboots. Default value is `#f`.

`enable-6to4?` (type: maybe-boolean)
: Automatically enable anycast 6to4 if possible. This is not recommended, as the use of 6to4 will generally lead to a severe degradation of connection quality. See RFC6343. Default value is `#f` (as recommended by RFC6343 section 4.1).

`vendor-class-id` (type: maybe-string)
: Set DHCP option 60 (Vendor Class ID) to the given string. This option can be used by DHCP servers to identify specific clients without having to rely on MAC address ranges, etc.

`enable-online-check?` (type: maybe-boolean)
: Enable or disable use of HTTP GET as an online status check. When a service is in a READY state, and is selected as default, ConnMan will issue an HTTP GET request to verify that end-to-end connectivity is successful. Only then the service will be transitioned to ONLINE state. If this setting is false, the default service will remain in READY state. Default value is `#t`.

`online-check-ipv4-url` (type: maybe-string)
: IPv4 URL used during the online status check. Please refer to the README for more detailed information. Default value is `http://ipv4.connman.net/online/status.html`.

`online-check-ipv6-url` (type: maybe-string)
: IPv6 URL used during the online status check. Please refer to the README for more detailed information. Default value is `http://ipv6.connman.net/online/status.html`.

`online-check-initial-interval` (type: maybe-number)
: Range of intervals between two online check requests. Please refer to the README for more detailed information. Default value is '`1`'.

`online-check-max-interval` (type: maybe-number)
: Range of intervals between two online check requests. Please refer to the README for more detailed information. Default value is '`1`'.

`enable-online-to-ready-transition?` (type: maybe-boolean)
: WARNING: This is an experimental feature. In addition to `enable-online-check` setting, enable or disable use of HTTP GET to detect the loss of end-to-end connectivity. If this setting is `#f`, when the default service transitions to ONLINE state, the HTTP GET request is no more called until next cycle, initiated by a transition of the default service to DISCONNECT state. If this setting is `#t`, the HTTP GET request keeps being called to guarantee that end-to-end connectivity is still successful. If not, the default service will transition to READY state, enabling another service to become the default one, in replacement. Default value is `#f`.

`auto-connect-roaming-services?` (type: maybe-boolean)
: Automatically connect roaming services. This is not recommended unless you know you won't have any billing problem. Default value is `#f`.

`address-conflict-detection?` (type: maybe-boolean)
: Enable or disable the implementation of IPv4 address conflict detection according to RFC5227. ConnMan will send probe ARP packets to see if an IPv4 address is already in use before assigning the address to an interface. If an address conflict occurs for a statically configured address, an IPv4LL address will be chosen instead (according to RFC3927). If an address conflict occurs for an address offered via DHCP, ConnMan sends a DHCP DECLINE once and for the second conflict resorts to finding an IPv4LL address. Default value is `#f`.

`localtime` (type: maybe-string)
: Path to localtime file. Defaults to `/etc/localtime`.

`regulatory-domain-follows-timezone?` (type: maybe-boolean)
: Enable regulatory domain to be changed along timezone changes. With this option set to true each time the timezone changes the first present ISO3166 country code is read from `/usr/share/zoneinfo/zone1970.tab` and set as regulatory domain value. Default value is `#f`.

`resolv-conf` (type: maybe-string)
: Path to resolv.conf file. If the file does not exist, but intermediate directories exist, it will be created. If this option is not set, it tries to write into `/var/run/connman/resolv.conf` if it fails (`/var/run/connman` does not exist or is not writeable). If you do not want to update resolv.conf, you can set `/dev/null`.

`wpa-supplicant-service-type` [Variable]
: Este es el tipo de servicio para la ejecución de WPA supplicant (`https://w1.fi/wpa_supplicant/`), un daemon de identificación necesario para la identificación en redes WiFi o ethernet cifradas.

`wpa-supplicant-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de WPA Supplicant.

Toma los siguientes parámetros:

`wpa-supplicant` (predeterminado: `wpa-supplicant`)
El paquete de WPA Supplicant usado.

`requirement` (predeterminados: `'(user-processes loopback syslogd)`
Lista de servicios que deben iniciarse antes del arranque de WPA Supplicant.

`dbus?` (predeterminado: `#t`)
Si se escuchan o no peticiones en D-Bus.

`pid-file` (predeterminado: `"/var/run/wpa_supplicant.pid"`)
Dónde se almacena el archivo con el PID.

`interface` (predeterminado: `#f`)
En caso de proporcionarse un valor, debe especificar el nombre de la interfaz de red que WPA supplicant controlará.

`config-file` (predeterminado: `#f`)
Archivo de configuración opcional usado.

`extra-options` (predeterminadas: `'()`)
Lista de parámetros adicionales a pasar al daemon en la línea de órdenes.

Some networking devices such as modems require special care, and this is what the services below focus on.

`modem-manager-service-type` [Variable]

This is the service type for the ModemManager (`https://wiki.gnome.org/Projects/ModemManager`) service. The value for this service type is a `modem-manager-configuration` record.

Este servicio es parte de `%desktop-services` (see Section 11.10.9 [Servicios de escritorio], page 363).

`modem-manager-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de ModemManager.

`modem-manager` (predeterminado: `modem-manager`)
El paquete de ModemManager usado.

`usb-modeswitch-service-type` [Variable]

This is the service type for the USB_ModeSwitch (`https://www.draisberghof.de/usb_modeswitch/`) service. The value for this service type is a `usb-modeswitch-configuration` record.

Cuando se conectan, algunos modem USB (y otros dispositivos USB) se presentan inicialmente como medios de almacenamiento de sólo-lectura y no como un modem. Deben *cambiar de modo* antes de poder usarse. El tipo de servicio USB_ModeSwitch instala reglas de udev para cambiar automáticamente de modo cuando se conecten estos dispositivos.

Este servicio es parte de `%desktop-services` (see Section 11.10.9 [Servicios de escritorio], page 363).

`usb-modeswitch-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de USB_ModeSwitch.

`usb-modeswitch` (predeterminado: `usb-modeswitch`)
: El paquete USB_ModeSwitch que proporciona los binarios para el cambio de modo.

`usb-modeswitch-data` (predeterminado: `usb-modeswitch-data`)
: El paquete que proporciona los datos de dispositivos y las reglas de udev usadas por USB_ModeSwitch.

`config-file` (predeterminado: `#~(string-append #$usb-modeswitch:dispatcher "/etc/usb_modeswitch.conf")`)
: Archivo de configuración usado para el gestor de eventos (dispatcher) de USB_ModeSwitch. De manera predeterminada se usa el archivo que viene con USB_ModeSwitch, que deshabilita el registro en `/var/log` junto a otras configuraciones. Si se proporciona `#f` no se usa ningún archivo de configuración.

### 11.10.5 Servicios de red

The `(gnu services networking)` module discussed in the previous section provides services for more advanced setups: providing a DHCP service for others to use, filtering packets with iptables or nftables, running a WiFi access point with `hostapd`, running the `inetd` "superdaemon", and more. This section describes those.

`dhcpd-service-type` [Variable]

Este tipo define un servicio que ejecuta el daemon DHCP. Para crear un servicio de este tipo debe proporcionar un objeto `<dhcpd-configuration>`. Por ejemplo:

```
(service dhcpd-service-type
         (dhcpd-configuration
          (config-file (local-file "mi-dhcpd.conf"))
          (interfaces '("enp0s25"))))
```

`dhcpd-configuration` [Tipo de datos]

`package` (predeterminado: `isc-dhcp`)
: The package that provides the DHCP daemon. This package is expected to provide the daemon at `sbin/dhcpd` relative to its output directory. The default package is the ISC's DHCP server (`https://www.isc.org/dhcp/`).

`config-file` (predeterminado: `#f`)
: El archivo de configuración usado. Esta opción es necesaria. Se le proporcionará a `dhcpd` a través de su opción `-cf`. Puede ser cualquier objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167). Véase `man dhcpd.conf` para detalles sobre la sintaxis del archivo de configuración.

`version` (predeterminada: `"4"`)
: La versión DHCP usada. El servidor DHCP de ISC permite los valores "4", "6" y "4o6". Corresponden con las opciones `-4`, `-6` y `-4o6` del programa `dhcpd`. Véase `man dhcpd` para más detalles.

`run-directory` (predeterminado: `"/run/dhcpd"`)
: El directorio de ejecución usado. Durante la activación del servicio se creará en caso de no existir.

`pid-file` (predeterminado: `"/run/dhcpd/dhcpd.pid"`)
: El archivo de PID usado. Corresponde con la opción `-pf` de `dhcpd`. Véase `man dhcpd` para más detalles.

`interfaces` (predeterminadas: `'()`)
: Los nombres de las interfaces de red en las que dhcpd debería esperar retransmisiones. Si la lista no está vacía, entonces sus elementos (que deben ser cadenas) se añadirá a la invocación de `dhcpd` cuando se inicie el daemon. Puede no ser necesaria la especificación explícita aquí de ninguna interfaz; véase `man dhcpd` para más detalles.

`hostapd-service-type` [Variable]

Este es el tipo de servicio que ejecuta el daemon hostapd (`https://w1.fi/hostapd/`) para configurar puntos de acceso WiFi (IEEE 802.11) y servidores de identificación. Su valor debe ser un registro `hostapd-configuration` como en este ejemplo:

```
;; Use wlan1 para ejecutar el punto de acceso para "Mi red".
(service hostapd-service-type
         (hostapd-configuration
          (interface "wlan1")
          (ssid "Mi red")
          (channel 12)))
```

`hostapd-configuration` [Tipo de datos]

Este tipo de datos representa la configuración del servicio hostapd, y tiene los siguientes campos:

`package` (predeterminado: `hostapd`)
: El paquete hostapd usado.

`interface` (predeterminado: `"wlan0"`)
: La interfaz de red en la que se establece el punto de acceso WiFi.

`ssid`
: El SSID (*identificador del servicio*, del inglés “service set identifier”), una cadena que identifica esta red.

`broadcast-ssid?` (predeterminado: `#t`)
: Determina si se emite este SSID.

`channel` (predeterminado: `1`)
: El canal WiFi usado.

`driver` (predeterminado: `"nl80211"`)
: El tipo de controlador de la interfaz. `"nl80211"` se usa con todos los controladores de mac80211 de Linux. Use `"none"` si está construyendo hostapd como un servidor RADIUS independiente que no controla ningún controlador de red cableada o inalámbrica.

`extra-settings` (predeterminado: `""`)
: Configuración adicional que se añade literalmente al archivo de configuración de hostapd. Véase `https://w1.fi/cgit/hostap/plain/hostapd/hostapd.conf` para la referencia del archivo de configuración.

`simulated-wifi-service-type` [Variable]
Tipo de servicio que simula una red inalámbrica ("WiFi"), lo que puede ser útil en máquinas virtuales para realizar pruebas. El servicio carga el módulo `mac80211_hwsim` (`https://www.kernel.org/doc/html/latest/networking/mac80211_hwsim/mac80211_hwsim.html`) del núcleo Linux e inicia hostapd para crear una red inalámbrica virtual que puede verse en `wlan0`, de manera predeterminada.

El valor de este servicio es un registro `hostapd-configuration`.

`iptables-service-type` [Variable]
Este es el tipo de servicio para la aplicación de configuración de iptables. iptables es un entorno de trabajo para el filtrado de paquetes implementado por el núcleo Linux. Este servicio permite la configuración de iptables tanto para IPv4 como IPv6. Un ejemplo simple de cómo rechazar todas las conexiones entrantes excepto aquellas al puerto 22 de ssh se muestra a continuación.

```
(service iptables-service-type
         (iptables-configuration
          (ipv4-rules (plain-file "iptables.rules" "*filter
:INPUT ACCEPT
:FORWARD ACCEPT
:OUTPUT ACCEPT
-A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-port-unreachable
COMMIT
"))
          (ipv6-rules (plain-file "ip6tables.rules" "*filter
:INPUT ACCEPT
:FORWARD ACCEPT
:OUTPUT ACCEPT
-A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp6-port-unreachable
COMMIT
"))))
```

`iptables-configuration` [Tipo de datos]
El tipo de datos que representa la configuración de iptables.

`iptables` (predeterminado: `iptables`)
: El paquete iptables que proporciona `iptables-restore` y `ip6tables-restore`.

`ipv4-rules` (predeterminado: `%iptables-accept-all-rules`)
: Las reglas de iptables usadas. Se le proporcionarán a `iptables-restore`. Puede ser cualquier objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167).

`ipv6-rules` (predeterminadas: `%iptables-accept-all-rules`)
: Las reglas de ip6tables usadas. Se le proporcionarán a `ip6tables-restore`. Puede ser cualquier objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167).

`nftables-service-type` [Variable]
: This is the service type to set up a nftables configuration. nftables is a netfilter project that aims to replace the existing iptables, ip6tables, arptables and ebtables framework. It provides a new packet filtering framework, a new user-space utility `nft`, and a compatibility layer for iptables. This service comes with a default ruleset `%default-nftables-ruleset` that rejecting all incoming connections except those to the ssh port 22. To use it, simply write:

```
(service nftables-service-type)
```

`nftables-configuration` [Tipo de datos]
: El tipo de datos que representa la configuración de nftables.

`package` (predeterminado: `nftables`)
: El paquete nftables que proporciona `nft`.

`ruleset` (predeterminados: `%default-nftables-ruleset`)
: El conjunto de reglas de nftables usado. Puede ser cualquier objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167).

`ntp-service-type` [Variable]
: Este es el tipo del servicio que ejecuta el daemon del protocolo de tiempo en red (NTP) (`https://www.ntp.org`), `ntpd`. El daemon mantendrá el reloj del sistema sincronizado con el de los servidores NTP especificados.

  El valor de este servicio es un objeto `ntpd-configuration`, como se describe a continuación.

`ntp-configuration` [Tipo de datos]
: Este es el tipo de datos para la configuración del servicio NTP.

`servers` (predeterminados: `%ntp-servers`)
: La lista de servidores (registros `<ntp-server>`) con los que la herramienta `ntpd` se sincronizará. Véase la información sobre el tipo de datos `ntp-server` a continuación.

`allow-large-adjustment?` (predeterminado: `#t`)
: Esto determina si se le permite a `ntpd` realizar un ajuste inicial de más de 1000 segundos.

`ntp` (predeterminado: `ntp`)
: El paquete NTP usado.

`%ntp-servers` [Variable]
Lista de nombres de máquinas usadas como servidores NTP predeterminados. Son servidores del NTP Pool Project (`https://www.ntppool.org/en/`).

`ntp-server` [Tipo de datos]
Tipo de datos que representa la configuración de un servidor NTP.

`type` (predeterminado: `'server`)
The type of the NTP server, given as a symbol. One of `'pool`, `'server`, `'peer`, `'broadcast` or `'manycastclient`.

`address` La dirección del servidor, como una cadena.

`options` NTPD options to use with that specific server, given as a list of option names and/or of option names and values tuples. The following example define a server to use with the options `iburst` and `prefer`, as well as `version` 3 and a `maxpoll` time of 16 seconds.

```
(ntp-server
 (type 'server)
 (address "miservidor.ntp.server.org")
 (options `(iburst (version 3) (maxpoll 16) prefer))))
```

`openntpd-service-type` [Variable]
Ejecuta `ntpd`, el daemon del protocolo de tiempo en red (NTP), implementado por OpenNTPD (`http://www.openntpd.org`). El daemon mantendrá el reloj del sistema sincronizado con el de los servidores proporcionados.

```
(service
 openntpd-service-type
 (openntpd-configuration
  (listen-on '("127.0.0.1" "::1"))
  (sensor '("udcf0 correction 70000"))
  (constraint-from '("www.gnu.org"))
  (constraints-from '("https://www.google.com/"))))
```

`%openntpd-servers` [Variable]
Esta variable es una lista de las direcciones de servidores definidos en `%ntp-servers`.

`openntpd-configuration` [Tipo de datos]

`openntpd` (default: `openntpd`)
The openntpd package to use.

`listen-on` (predeterminadas: `'("127.0.0.1" "::1")`)
Una lista de direcciones IP o nombres de máquina en los que el daemon ntpd debe escuchar conexiones.

`query-from` (predeterminadas: `'()`)
Una lista de direcciones IP locales que el daemon ntpd debe usar para consultas salientes.

`sensor` (predeterminados: `'()`)
: Especifica una lista de dispositivos de sensores de tiempo de ntpd debería usar. `ntpd` escuchará cada sensor que realmente exista e ignora los que no. Véase la documentación de las desarrolladoras originales (`https://man.openbsd.org/ntpd.conf`) para más información.

`server` (predeterminado: `'()`)
: Especifica una lista de direcciones IP o nombres de máquina de servidores NTP con los que sincronizarse.

`servers` (predeterminada: `%openntp-servers`)
: Una lista de direcciones IP o nombres de máquina con los que el daemon ntpd se debe sincronizar.

`constraint-from` (predeterminado: `'()`)
: `ntpd` puede configurarse para que solicite la fecha a través del campo "Date" de servidores HTTPS en los que se confíe a través de TLS. Esta información de tiempo no se usa por precisión pero actúa como una condición verificada, por tanto reduciendo el impacto de ataques mediante la intervención del tráfico con servidores NTP no verificados. Especifica una lista de URL, direcciones IP o nombres de máquina de servidores HTTPS que proporcionarán la condición.

`constraints-from` (predeterminadas: `'()`)
: Como en *constraint-from*, proporciona una lista de URL, direcciones IP o nombres de máquina de servidores HTTP para proporcionar la condición. En caso de que el nombre de máquina resuelva en múltiples direcciones IP, `ntpd` calculará la condición mediana de todas ellas.

`inetd-service-type` [Variable]

Este servicio ejecuta el daemon `inetd` (see Section "inetd invocation" in *GNU Inetutils*). `inetd` escucha conexiones en sockets de internet, e inicia bajo demanda el programa servidor cuando se realiza una conexión en uno de esos sockets.

El valor de este servicio es un objeto `inetd-configuration`. El ejemplo siguiente configura el daemon `inetd` para proporcionar el servicio `echo` implementado por él mismo, así como un servicio smtp que reenvía el tráfico smtp por ssh a un servidor `servidor-smtp` tras la pasarela `máquina`:

```
(service
 inetd-service-type
 (inetd-configuration
  (entries (list
            (inetd-entry
             (name "echo")
             (socket-type 'stream)
             (protocol "tcp")
             (wait? #f)
             (user "root"))
            (inetd-entry
             (node "127.0.0.1")
```

```
                    (name "smtp")
                    (socket-type 'stream)
                    (protocol "tcp")
                    (wait? #f)
                    (user "root")
                    (program (file-append openssh "/bin/ssh"))
                    (arguments
                     '("ssh" "-qT" "-i" "/ruta/de/la/clave_ssh"
                       "-W" "servidor-smtp:25" "usuaria@maquina")))))))
```

A continuación se proporcionan más detalles acerca de `inetd-configuration`.

`inetd-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `inetd`.

`program` (predeterminado: `(file-append inetutils "/libexec/inetd")`)
: El ejecutable `inetd` usado.

`entries` (predeterminadas: `'()`)
: Una lista de entradas de servicio de `inetd`. Cada entrada debe crearse con el constructor `inted-entry`.

`inetd-entry` [Tipo de datos]

Tipo de datos que representa una entrada en la configuración de `inetd`. Cada entrada corresponde a un socket en el que `inetd` escuchará a la espera de peticiones.

`node` (predeterminado: `#f`)
: Cadena opcional, una lista separada por comas de direcciones locales que `inetd` debería usar cuando se escuche para este servicio. See Section "Configuration file" in *GNU Inetutils* para una descripción completa de todas las opciones.

`name` Una cadena, el nombre debe corresponder con una entrada en `/etc/services`.

`socket-type`
: Puede ser `'stream`, `'dgram`, `'raw`, `'rdm` o `'seqpacket`.

`protocol` Una cadena, debe corresponder con una entrada en `/etc/protocols`.

`wait?` (predeterminado: `#t`)
: Si `inetd` debe esperar la salida del servidor antes de reiniciar la escucha de nuevas peticiones de servicio.

`user` Una cadena que contiene el nombre (y, opcionalmente, el grupo) de la usuaria como la que se deberá ejecutar el servidor. El nombe de grupo se puede especificar en un sufijo, separado por dos puntos o un punto normal, es decir `"usuaria"`, `"usuaria:grupo"` o `"usuaria.grupo"`.

`program` (predeterminado: `"internal"`)
: El programa servidor que recibirá las peticiones, o `"internal"` si `inetd` debería usar un servicio implementado internamente.

`arguments` (predeterminados: `'()`)
: Una lista de cadenas u objetos "tipo-archivo", que serán los parámetros del programa servidor, empezando con el parámetro 0, es decir, el nombre del programa en sí mismo. Para los servicios internos de `inetd`, esta entrada debe ser `'()` o `'("internal")`.

See Section "Configuration file" in *GNU Inetutils*, para una información más detallada sobre cada campo de la configuración.

`opendht-service-type` [Variable]
: This is the type of the service running a OpenDHT (`https://opendht.net`) node, `dhtnode`. The daemon can be used to host your own proxy service to the distributed hash table (DHT), for example to connect to with Jami, among other applications.

  **Importante:** When using the OpenDHT proxy server, the IP addresses it "sees" from the clients should be addresses reachable from other peers. In practice this means that a publicly reachable address is best suited for a proxy server, outside of your private network. For example, hosting the proxy server on a IPv4 private local network and exposing it via port forwarding could work for external peers, but peers local to the proxy would have their private addresses shared with the external peers, leading to connectivity problems.

  The value of this service is a `opendht-configuration` object, as described below.

`opendht-configuration` [Data Type]
: Available `opendht-configuration` fields are:

  `opendht` (default: `opendht`) (type: file-like)
  : The `opendht` package to use.

  `peer-discovery?` (default: `#f`) (type: boolean)
  : Whether to enable the multicast local peer discovery mechanism.

  `enable-logging?` (default: `#f`) (type: boolean)
  : Whether to enable logging messages to syslog. It is disabled by default as it is rather verbose.

  `debug?` (default: `#f`) (type: boolean)
  : Whether to enable debug-level logging messages. This has no effect if logging is disabled.

  `bootstrap-host` (default: `"bootstrap.jami.net:4222"`) (type: maybe-string)
  : The node host name that is used to make the first connection to the network. A specific port value can be provided by appending the `:PORT` suffix. By default, it uses the Jami bootstrap nodes, but any host can be specified here. It's also possible to disable bootstrapping by explicitly setting this field to the `%unset-value` value.

  `port` (default: `4222`) (type: maybe-number)
  : The UDP port to bind to. When left unspecified, an available port is automatically selected.

`proxy-server-port` (type: maybe-number)
: Spawn a proxy server listening on the specified port.

`proxy-server-port-tls` (type: maybe-number)
: Spawn a proxy server listening to TLS connections on the specified port.

`tor-service-type` [Variable]

Type for a service that runs the Tor (`https://torproject.org`) anonymous networking daemon. The service is configured using a `<tor-configuration>` record. By default, the Tor daemon runs as the `tor` unprivileged user, which is a member of the `tor` group.

Services of this type can be extended by other services to specify *onion services* (in addition to those already specified in `tor-configuration`) as in this example:

```
(simple-service 'my-extra-onion-service tor-service-type
                (list (tor-onion-service-configuration
                        (name "extra-onion-service")
                        (mapping '((80 . "127.0.0.1:8080"))))))
```

`tor-configuration` [Tipo de datos]

`tor` (predeterminado: `tor`)
: El paquete que proporciona el daemon Tor. Se espera que este paquete proporcione el daemon en `bin/tor` de manera relativa al directorio de su salida. El paquete predeterminado es la implementación del Proyecto Tor (`https://www.torproject.org`).

`config-file` (predeterminado: `(plain-file "empty" "")`)
: El archivo de configuración usado. Se agregará al final del archivo de configuración predeterminado, y se proporcionará el archivo de configuración resultante a `tor` a través de su opción `-f`. Puede ser cualquier objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167). Véase `man tor` para detalles sobre la sintaxis del archivo de configuración.

`hidden-services` (predeterminados: `'()`)
: The list of `<tor-onion-service-configuration>` records to use. For any onion service you include in this list, appropriate configuration to enable the onion service will be automatically added to the default configuration file.

`socks-socket-type` (predeterminado: `'tcp`)
: El tipo socket predeterminado que Tor debe usar para su socket SOCKS. Debe ser `'tcp` i `'unix`. Si es `'tcp`, Tor escuchará en el puerto TCP 9050 de la interfaz local (es decir, localhost) de manera predeterminada. Si es `'unix`, tor escuchará en el socket de dominio de UNIX `/var/run/tor/socks-sock`, que tendrá permisos de escritura para miembros del grupo `tor`.

  Si desea personalizar el socket SOCKS de manera más detallada, mantenga `socks-socket-type` con su valor predeterminado de `'tcp` y use `config-file` para modificar el valor predeterminado proporcionando su propia opción `SocksPort`.

`control-socket?` (default: `#f`)
: Whether or not to provide a "control socket" by which Tor can be controlled to, for instance, dynamically instantiate tor onion services. If `#t`, Tor will listen for control commands on the UNIX domain socket `/var/run/tor/control-sock`, which will be made writable by members of the `tor` group.

`tor-onion-service-configuration` [Data Type]
: Data Type representing a Tor *Onion Service* configuration. See the Tor project's documentation (`https://community.torproject.org/onion-services/`) for more information. Available `tor-onion-service-configuration` fields are:

`name` (type: string)
: Name for this Onion Service. This creates a `/var/lib/tor/hidden-services/`*`name`* directory, where the `hostname` file contains the '`.onion`' host name for this Onion Service.

`mapping` (type: alist)
: Association list of port to address mappings. The following example:

```
'((22 . "127.0.0.1:22")
  (80 . "127.0.0.1:8080"))
```

maps ports 22 and 80 of the Onion Service to the local ports 22 and 8080.

El módulo `(gnu services rsync)` proporciona los siguientes servicios:

Puede ser que desee un daemon rsync si tiene archivos que desee tener disponibles de modo que cualquiera (o simplemente usted) pueda descargar archivos existentes o subir nuevos archivos.

`rsync-service-type` [Variable]
: Este es el tipo de servicio para el daemon rsync (`https://rsync.samba.org`). El valor tipo de servicio es un registro `rsync-configuration` como en este ejemplo.

```
;; Export two directories over rsync.  By default rsync listens on
;; all the network interfaces.
(service rsync-service-type
         (rsync-configuration
           (modules (list (rsync-module
                           (name "music")
                           (file-name "/srv/zik")
                           (read-only? #f))
                          (rsync-module
                           (name "movies")
                           (file-name "/home/charlie/movies"))))))
```

Véase a continuación para detalles sobre `rsync-configuration`.

`rsync-configuration` [Tipo de datos]
: Tipo de datos que representa la configuración para `rsync-service`.

`package` (predeterminado: *rsync*)
: Paquete `rsync` usado.

`address` (predeterminada: `#f`)
: IP address on which `rsync` listens for incoming connections. If unspecified, it defaults to listening on all available addresses.

`port-number` (predeterminado: `873`)
: Puerto TCP en el que `rsync` escucha conexiones entrantes. Si el puerto es menor a `1024`, `rsync` necesita iniciarse como `root`, tanto usuaria como grupo.

`pid-file` (predeterminado: `"/var/run/rsyncd/rsyncd.pid"`)
: Nombre del archivo donde `rsync` escribe su PID.

`lock-file` (predeterminado: `"/var/run/rsyncd/rsyncd.lock"`)
: Nombre del archivo donde `rsync` escribe su archivo de bloqueo.

`log-file` (predeterminado: `"/var/log/rsyncd.log"`)
: Nombre del archivo donde `rsync` escribe su archivo de registros.

`user` (predeterminada: `"root"`)
: Propietaria del proceso `rsync`.

`group` (default: `"root"`)
: Grupo del proceso `rsync`.

`uid` (default: `"rsyncd"`)
: Nombre o ID de usuaria bajo la cual se efectúan las transferencias desde y hacia el módulo cuando el daemon se ejecuta como `root`.

`gid` (default: `"rsyncd"`)
: Nombre o ID de grupo que se usa cuando se accede al módulo.

`modules` (predeterminados: `%default-modules`)
: List of "modules"—i.e., directories exported over rsync. Each element must be a `rsync-module` record, as described below.

`rsync-module` [Data Type]

This is the data type for rsync "modules". A module is a directory exported over the rsync protocol. The available fields are as follows:

`name`
: The module name. This is the name that shows up in URLs. For example, if the module is called `music`, the corresponding URL will be `rsync://host.example.org/music`.

`file-name`
: Name of the directory being exported.

`comment` (predeterminado: `""`)
: Comment associated with the module. Client user interfaces may display it when they obtain the list of available modules.

`read-only?` (default: `#t`)
: Whether or not client will be able to upload files. If this is false, the uploads will be authorized if permissions on the daemon side permit it.

`chroot?` (default: `#t`)
: When this is true, the rsync daemon changes root to the module's directory before starting file transfers with the client. This improves security, but requires rsync to run as root.

`timeout` (predeterminado: `300`)
: Idle time in seconds after which the daemon closes a connection with the client.

The `(gnu services syncthing)` module provides the following services:

You might want a syncthing daemon if you have files between two or more computers and want to sync them in real time, safely protected from prying eyes.

`syncthing-service-type` [Variable]
: This is the service type for the syncthing (`https://syncthing.net/`) daemon, The value for this service type is a `syncthing-configuration` record as in this example:

```
(service syncthing-service-type
         (syncthing-configuration (user "alice")))
```

**Nota:** This service is also available for Guix Home, where it runs directly with your user privileges (see Section 13.3.15 [Networking Home Services], page 701).

See below for details about `syncthing-configuration`.

`syncthing-configuration` [Data Type]
: Data type representing the configuration for `syncthing-service-type`.

`syncthing` (default: *syncthing*)
: `syncthing` package to use.

`arguments` (default: *'()*)
: List of command-line arguments passing to `syncthing` binary.

`logflags` (default: *0*)
: Sum of logging flags, see Syncthing documentation logflags (`https://docs.syncthing.net/users/syncthing.html#cmdoption-logflags`).

`user` (default: *#f*)
: The user as which the Syncthing service is to be run. This assumes that the specified user exists.

`group` (default: "*users*")
: The group as which the Syncthing service is to be run. This assumes that the specified group exists.

`home` (default: *#f*)
: Common configuration and data directory. The default configuration directory is `$HOME` of the specified Syncthing `user`.

Es más, `(gnu services ssh)` proporciona los siguientes servicios.

`lsh-service-type` [Variable]
: Type of the service that runs the GNU lsh secure shell (SSH) daemon, `lshd`. The value for this service is a `<lsh-configuration>` object.

`lsh-configuration` [Data Type]

Data type representing the configuration of `lshd`.

`lsh` (default: `lsh`) (type: file-like)
The package object of the GNU lsh secure shell (SSH) daemon.

`daemonic?` (default: `#t`) (type: boolean)
Whether to detach from the controlling terminal.

`host-key` (default: `"/etc/lsh/host-key"`) (type: string)
File containing the *host key*. This file must be readable by root only.

`interfaces` (default: `'()`) (type: list)
List of host names or addresses that `lshd` will listen on. If empty, `lshd` listens for connections on all the network interfaces.

`port-number` (default: `22`) (type: integer)
Port to listen on.

`allow-empty-passwords?` (default: `#f`) (type: boolean)
Whether to accept log-ins with empty passwords.

`root-login?` (default: `#f`) (type: boolean)
Whether to accept log-ins as root.

`syslog-output?` (default: `#t`) (type: boolean)
Whether to log `lshd` standard output to syslogd. This will make the service depend on the existence of a syslogd service.

`pid-file?` (default: `#f`) (type: boolean)
When `#t`, `lshd` writes its PID to the file specified in *pid-file*.

`pid-file` (default: `"/var/run/lshd.pid"`) (type: string)
File that `lshd` will write its PID to.

`x11-forwarding?` (default: `#t`) (type: boolean)
Whether to enable X11 forwarding.

`tcp/ip-forwarding?` (default: `#t`) (type: boolean)
Whether to enable TCP/IP forwarding.

`password-authentication?` (default: `#t`) (type: boolean)
Whether to accept log-ins using password authentication.

`public-key-authentication?` (default: `#t`) (type: boolean)
Whether to accept log-ins using public key authentication.

`initialize?` (default: `#t`) (type: boolean)
When `#f`, it is up to the user to initialize the randomness generator (see Section "lsh-make-seed" in *LSH Manual*), and to create a key pair with the private key stored in file *host-key* (see Section "lshd basics" in *LSH Manual*).

`openssh-service-type` [Variable]

Este es el tipo para el daemon de shell seguro OpenSSH (`http://www.openssh.org`), `sshd`. Su valor debe ser un registro `openssh-configuration` como en este ejemplo:

```
(service openssh-service-type
```

```
(openssh-configuration
  (x11-forwarding? #t)
  (permit-root-login 'prohibit-password)
  (authorized-keys
    `(("alice" ,(local-file "alice.pub"))
      ("bob" ,(local-file "bob.pub")))))
```

Véase a continuación detalles sobre `openssh-configuration`.

Este servicio se puede extender con claves autorizadas adicionales, como en este ejemplo:

```
(service-extension openssh-service-type
                   (const `(("carlos"
                             ,(local-file "carlos.pub")))))
```

`openssh-configuration` [Tipo de datos]

Este es el registro de configuración para `sshd` de OpenSSH.

`openssh` (predeterminado: *openssh*)
: The OpenSSH package to use.

`pid-file` (predeterminado: `"/var/run/sshd.pid"`)
: Nombre del archivo donde `sshd` escribe su PID.

`port-number` (predeterminado: `22`)
: Puerto TCP en el que `sshd` espera conexiones entrantes.

`max-connections` (default: `200`)
: Hard limit on the maximum number of simultaneous client connections, enforced by the inetd-style Shepherd service (see Section "Service De- and Constructors" in *The GNU Shepherd Manual*).

`permit-root-login` (predeterminado: `#f`)
: This field determines whether and when to allow logins as root. If `#f`, root logins are disallowed; if `#t`, they are allowed. If it's the symbol `'prohibit-password`, then root logins are permitted but not with password-based authentication.

`allow-empty-passwords?` (predeterminado: `#f`)
: Cuando es verdadero, las usuarias con contraseñas vacías pueden ingresar en el sistema. Cuando es falso, no pueden.

`password-authentication?` (predeterminado: `#t`)
: Cuando es verdadero, las usuarias pueden ingresar al sistema con su contraseña. En caso falso, tienen otros métodos de identificación.

`public-key-authentication?` (predeterminado: `#t`)
: Cuando es verdadero, las usuarias pueden ingresar en el sistema mediante el uso de clave publica para su identificación. Cuando es falso, las usuarias tienen que usar otros métodos de identificación.

  Las claves públicas autorizadas se almacenan en `~/.ssh/authorized_keys`. Se usa únicamente por la versión 2 del protocolo.

`x11-forwarding?` (predeterminado: `#f`)
: Cuando verdadero, la retransmisión de conexiones del cliente gráfico X11 está desactivada—en otras palabras, las opciones `-X` y `-Y` de `ssh` funcionarán.

`allow-agent-forwarding?` (predeterminado: `#t`)
: Si se permite la retransmisión del agente de claves.

`allow-tcp-forwarding?` (predeterminado: `#t`)
: Si se permite la retransmisión TCP.

`gateway-ports?` (predeterminado: `#f`)
: Si se permiten los puertos pasarela.

`challenge-response-authentication?` (predeterminado: `#f`)
: Especifica si la identificación mediante respuesta de desafío está permitida (por ejemplo, a través de PAM).

`use-pam?` (predeterminado: `#t`)
: Permite el uso de la interfaz de módulos de identificación conectables (PAM). Si es `#t` se activará la identificación PAM mediante el uso de `challenge-response-authentication?` y `password-authentication?`, además del procesado de los módulos de cuenta usuaria y de sesión de PAM en todos los tipos de identificación.

  Debido a que la identificación mediante respuesta de desafío de PAM tiene un rol equivalente a la identificación por contraseña habitualmente, debería desactivar `challenge-response-authentication?` o `password-authentication?`.

`print-last-log?` (predeterminado: `#t`)
: Especifica si `sshd` debe imprimir la fecha y hora del último ingreso al sistema de la usuaria cuando una usuaria ingresa interactivamente.

`subsystems` (predeterminados: `'(("sftp" "internal-sftp"))`)
: Configura subsistemas externos (por ejemplo, el daemon de transmisión de archivos).

  Esta es una lista de listas de dos elementos, cada una de las cuales que contienen el nombre del subsistema y una orden (con parámetros opcionales) para ejecutar tras petición del subsistema.

  La orden `internal-sftp` implementa un servidor SFTP dentro del mismo proceso. De manera alternativa, se puede especificar la orden `sftp-server`:

```
(service openssh-service-type
         (openssh-configuration
          (subsystems
           `(("sftp" ,(file-append openssh "/libexec/sftp-server"))))))
```

`accepted-environment` (predeterminado: `'()`)
: Una lista de cadenas que describe qué variables de entorno pueden ser exportadas.

Cada cadena obtiene su propia línea. Véase la opción `AcceptEnv` en `man sshd_config`.

Este ejemplo permite a clientes ssh exportar la variable `COLORTERM`. La establecen emuladores de terminal que implementan colores. Puede usarla en su archivo de recursos del shell para permitir colores en la línea de órdenes y las propias ordenes si esta variable está definida.

```
(service openssh-service-type
         (openssh-configuration
           (accepted-environment '("COLORTERM"))))
```

`authorized-keys` (predeterminadas: `'()`)

Esta es la lista de claves autorizadas. Cada elemento de la lista es un nombre de usuaria seguido de uno o más objetos "tipo-archivo" que representan claves públicas SSH. Por ejemplo:

```
(openssh-configuration
  (authorized-keys
    `(("rekado" ,(local-file "rekado.pub"))
      ("chris" ,(local-file "chris.pub"))
      ("root" ,(local-file "rekado.pub") ,(local-file "chris.pub")))))
```

registra las claves públicas especificadas para las cuentas `rekado`, `chris` y `root`.

Se pueden especificar claves autorizadas adicionales a través de `service-extension`.

Tenga en cuenta que esto *no* interfiere con el uso de `~/.ssh/authorized_keys`.

`generate-host-keys?` (default: `#t`)

Whether to generate host key pairs with `ssh-keygen -A` under `/etc/ssh` if there are none.

Generating key pairs takes a few seconds when enough entropy is available and is only done once. You might want to turn it off for instance in a virtual machine that does not need it because host keys are provided in some other way, and where the extra boot time is a problem.

`log-level` (predeterminado: `'info`)

Es un símbolo que especifica el nivel de detalle en los registros: `quiet`, `fatal`, `error`, `info`, `verbose`, `debug`, etc. Véase la página del manual de `sshd_config` para la lista completa de los nombres de nivel.

`extra-content` (predeterminado: `""`)

Este campo puede usarse para agregar un texto arbitrario al archivo de configuración. Es especialmente útil para configuraciones elaboradas que no se puedan expresar de otro modo. Esta configuración, por ejemplo, generalmente desactivaría el ingreso al sistema como root, pero lo permite para una dirección IP específica:

```
(openssh-configuration
  (extra-content "\
```

```
                Match Address 192.168.0.1
                  PermitRootLogin yes"))
```

`dropbear-service-type` [Variable]

Type of the service that runs the Dropbear SSH daemon (`https://matt.ucc.asn.au/dropbear/dropbear.html`), whose value is a `<dropbear-configuration>` object.

For example, to specify a Dropbear service listening on port 1234:

```
(service dropbear-service-type (dropbear-configuration
                                 (port-number 1234)))
```

`dropbear-configuration` [Tipo de datos]

Este tipo de datos representa la configuración del daemon Dropbear SSH.

`dropbear` (predeterminado: *dropbear*)
: El paquete de Dropbear usado.

`port-number` (predeterminado: 22)
: Puerto TCP donde el daemon espera conexiones entrantes.

`syslog-output?` (predeterminado: `#t`)
: Determina si se envía la salida a syslog.

`pid-file` (predeterminado: `"/var/run/dropbear.pid"`)
: El nombre de archivo del archivo de PID del daemon.

`root-login?` (predeterminado: `#f`)
: Si se permite el ingreso al sistema como `root`.

`allow-empty-passwords?` (predeterminado: `#f`)
: Si se permiten las contraseñas vacías.

`password-authentication?` (predeterminado: `#t`)
: Determina si se usará identificación basada en contraseña.

`autossh-service-type` [Variable]

Tipo del servicio para el programa AutoSSH (`https://www.harding.motd.ca/autossh`) que ejecuta una copia de `ssh` y la monitoriza, reiniciando la conexión en caso de que se rompa la conexión o deje de transmitir datos. AutoSSH puede ejecutarse manualmente en la línea de órdenes proporcionando los parámetros al binario `autossh` del paquete `autossh`, pero también se puede ejecutar como un servicio de Guix. Este último caso de uso se encuentra documentado aquí.

AutoSSH se puede usar para retransmitir tráfico local a una máquina remota usando un túnel SSH, y respeta el archivo de configuración `~/.ssh/config` de la cuenta bajo la que se ejecute.

Por ejemplo, para especificar un servicio que ejecute autossh con la cuenta `pino` y retransmita todas las conexiones locales al puerto `8081` hacia `remote:8081` usando un túnel SSH, añada esta llamada al campo `services` del sistema operativo:

```
(service autossh-service-type
         (autossh-configuration
          (user "pino")
          (ssh-options (list "-T" "-N" "-L" "8081:localhost:8081" "remote.net")))
```

`autossh-configuration` [Tipo de datos]

Este tipo de datos representa la configuración del servicio AutoSSH.

`user` (predeterminado: `"autossh"`)
: La cuenta de usuaria con la cual se ejecuta el servicio AutoSSH. Se asume que existe dicha cuenta.

`poll` (predeterminado: `600`)
: Especifica el tiempo de comprobación de la conexión en segundos.

`first-poll` (predeterminado: `#f`)
: Especifica cuantos segundos espera AutoSSH antes de la primera prueba de conexión. Tras esta primera prueba las comprobaciones se realizan con la frecuencia definida en `poll`. Cuando se proporciona el valor `#f` la primera comprobación no se trata de manera especial y también usará el valor especificado en `poll`.

`gate-time` (predeterminado: `30`)
: Especifica cuantos segundos debe estar activa una conexión SSH antes de que se considere satisfactoria.

`log-level` (predeterminado: `1`)
: El nivel de registro, corresponde con los niveles usados por—por lo que `0` es el más silencioso y `7` el que contiene más información.

`max-start` (predeterminado: `#f`)
: El número de veces máximo que puede lanzarse (o reiniciarse) SSH antes de que AutoSSH termine. Cuando se proporciona `#f`, no existe un máximo por lo que AutoSSH puede reiniciar la conexión indefinidamente.

`message` (predeterminado: `""`)
: El mensaje que se añade al mensaje de eco que se envía cuando se prueban las conexiones.

`port` (predeterminado: `"0"`)
: Los puertos usados para la monitorización de la conexión. Cuando se proporciona `"0"`, se desactiva la monitorización. Cuando se proporciona `"`*n*`"` donde *n* es un entero positivo, los puertos *n* y *n*+1 se usan para monitorizar la conexión, de tal modo que el puerto *n* es el puerto base de monitorización y `n+1` es el puerto de eco. Cuando se proporciona `"`*n*`:`*m*`"` donde *n* y *m* son enteros positivos, los puertos *n* y *m* se usan para monitorizar la conexión, de tal modo que *n* es el puerto base de monitorización y *m* es el puerto de eco.

`ssh-options` (predeterminados: `'()`)
: Lista de parámetros de línea de órdenes proporcionados a `ssh` cuando se ejecuta. Las opciones `-f` y `-M` están reservadas para AutoSSH y pueden causar un comportamiento indefinido.

`webssh-service-type` [Variable]

Tipo para el programa WebSSH (`https://webssh.huashengdun.org/`) que ejecuta un cliente SSH web. WebSSH puede ejecutarse manualmente en la línea de órdenes

proporcionando los parámetros al binario `wssh` del paquete `webssh`, pero también se puede ejecutar como un servicio de Guix. Este último caso de uso se encuentra documentado aquí.

Por ejemplo, para especificar un servicio que ejecute WebSSH en la interfaz de red local sobre el puerto `8888` con una política de rechazo predeterminado con una lista de máquinas a las que se les permite explícitamente la conexión, y NGINX como pasarela inversa de este servicio a la escucha de conexiones HTTPS, añada esta llamada al campo `services` de su declaración de sistema operativo:

```
(service webssh-service-type
  (webssh-configuration (address "127.0.0.1")
                        (port 8888)
                        (policy 'reject)
                        (known-hosts '("localhost ecdsa-sha2-nistp256 AAAA..."
                                       "127.0.0.1 ecdsa-sha2-nistp256 AAAA..."))))

(service nginx-service-type
         (nginx-configuration
          (server-blocks
           (list
            (nginx-server-configuration
             (inherit %webssh-configuration-nginx)
             (server-name '("webssh.example.com"))
             (listen '("443 ssl"))
             (ssl-certificate (letsencrypt-certificate "webssh.example.com"))
             (ssl-certificate-key (letsencrypt-key "webssh.example.com"))
             (locations
              (cons (nginx-location-configuration
                     (uri "/.well-known")
                     (body '("root /var/www;")))
                    (nginx-server-configuration-locations %webssh-configuration-n
```

`webssh-configuration` [Tipo de datos]

Tipo de datos que representa la configuración para `webssh-service`.

`package` (predeterminado: *webssh*)
Paquete `webssh` usado.

`user-name` (predeterminado: "*webssh*")
Nombre o ID de usuaria bajo la cual se efectúan las transferencias desde y hacia el módulo.

`group-name` (predeterminado: "*webssh*")
Nombre o ID de grupo que se usa cuando se accede al módulo.

`address` (predeterminada: *#f*)
Dirección IP en la que `webssh` espera conexiones entrantes.

`port` (predeterminado: *8888*)
Puerto TCP en el que `webssh` espera conexiones entrantes.

`policy` (predeterminada: *#f*)
: Política de conexión. La política *reject* necesita que se especifique un valor en *known-hosts*.

`known-hosts` (predeterminada: *'()*)
: Lista de máquinas a las que se permite realizar conexiones SSH desde `webssh`.

`log-file` (predeterminado: `"/var/log/webssh.log"`)
: Name of the file where `webssh` writes its log file.

`log-level` (predeterminado: *#f*)
: Nivel de registro.

`block-facebook-hosts-service-type` [Variable]
: This service type adds a list of known Facebook hosts to the `/etc/hosts` file. (see Section "Host Names" in *The GNU C Library Reference Manual*) Each line contains an entry that maps a known server name of the Facebook on-line service—e.g., `www.facebook.com`—to unroutable IPv4 and IPv6 addresses.

  Este mecanismo puede impedir a los programas que se ejecutan localmente, como navegadores Web, el acceso a Facebook.

El módulo `(gnu services avahi)` proporciona la siguiente definición.

`avahi-service-type` [Variable]
: Es el servicio que ejecuta `avahi-daemon`, un servidor mDNS/DNS-SD a nivel del sistema que permite el descubrimiento de servicios y la búsqueda de nombres de máquina "sin configuración" ("zero-configuration", véase `https://avahi.org/`). Su valor debe ser un registro `avahi-configuration`—véase a continuación.

  Este servicio extiende el daemon de la caché del servicio de nombres (nscd) de manera que pueda resolver nombres de máquina `.local` mediante el uso de nss-mds (`https://0pointer.de/lennart/projects/nss-mdns`). See Section 11.13 [Selector de servicios de nombres], page 622, para información sobre la resolución de nombres de máquina.

  De manera adicional, añade el paquete *avahi* al perfil del sistema de manera que ordenes como `avahi-browse` estén disponibles de manera directa.

`avahi-configuration` [Tipo de datos]
: Tipo de datos que representa la configuración de Avahi.

  `host-name` (predeterminado: `#f`)
  : Si es diferente de `#f`, se usa como el nombre de máquina a publicar para esta máquina; en otro caso, usa el nombre actual de la máquina.

  `publish?` (predeterminado: `#t`)
  : Cuando es verdadero, permite la publicación (retransmisión) de nombres de máquina y servicios a través de la red.

  `publish-workstation?` (predeterminado: `#t`)
  : Cuando es verdadero, `avahi-daemon` publica el nombre de máquina y la dirección IP a través de mDNS en la red local. Para ver los nombres de máquina publicados en su red local, puede ejecutar:

```
avahi-browse _workstation._tcp
```

`wide-area?` (predeterminado: `#f`)
: Cuando es verdadero, se permite DNS-SD sobre DNS unicast.

`ipv4?` (predeterminado: `#t`)
`ipv6?` (predeterminado: `#t`)
: Estos campos determinan si usar sockets IPv4/IPv6.

`domains-to-browse` (predeterminado: `'()`)
: Esta es la lista de dominios a explorar.

`openvswitch-service-type` [Variable]
Este es el tipo del servicio Open vSwitch (`https://www.openvswitch.org`), cuyo valor debe ser un objeto `openvswitch-configuration`.

`openvswitch-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de Open vSwitch, un switch virtual multicapa que está diseñado para permitir una automatización masiva en la red a través de extensión programática.

`package` (predeterminado: *openvswitch*)
: El objeto paquete de Open vSwitch.

`pagekite-service-type` [Variable]
El tipo de servicio para el servicio PageKite (`https://pagekite.net`), una solución de encaminado para hacer servidores de la red local visibles públicamente, incluso detrás de cortafuegos restrictivos o NAT sin redirección de puertos. El valor para este servicio es un registro `pagekite-configuration`.

Este es un ejemplo que expone los daemon HTTP y SSH locales:

```
(service pagekite-service-type
         (pagekite-configuration
           (kites '("http:@kitename:localhost:80:@kitesecret"
                    "raw/22:@kitename:localhost:22:@kitesecret"))
           (extra-file "/etc/pagekite.rc")))
```

`pagekite-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de PageKite.

`package` (predeterminado: *pagekite*)
: El objeto paquete de PageKite.

`kitename` (predeterminado: `#f`)
: Nombre de PageKite para la identificación con el servidor de fachada.

`kitesecret` (predeterminado: `#f`)
: Secreto compartido para la comunicación con el servidor. Probablemente debería almacenarlo dentro `extra-file` en vez de aquí.

`frontend` (predeterminado: `#f`)
: Conecta al servidor de fachada de PageKite con este nombre en vez de al servicio de pagekite.net.

`kites` (predeterminados: `'("http:@kitename:localhost:80:@kitesecret")`)
: List of service kites to use. Exposes HTTP on port 80 by default. The format is `proto:kitename:host:port:secret`.

`extra-file` (predeterminado: `#f`)
: Archivo adicional de configuración que debe leerse, el cual se espera que sea creado de forma manual. Úselo para añadir opciones adicionales y gestionar secretos compartidos fuera de banda.

`yggdrasil-service-type` [Variable]

The service type for connecting to the Yggdrasil network (`https://yggdrasil-network.github.io/`), an early-stage implementation of a fully end-to-end encrypted IPv6 network.

> Yggdrasil provides name-independent routing with cryptographically generated addresses. Static addressing means you can keep the same address as long as you want, even if you move to a new location, or generate a new address (by generating new keys) whenever you want. `https://yggdrasil-network.github.io/2018/07/28/addressing.html`

Pass it a value of `yggdrasil-configuration` to connect it to public peers and/or local peers.

Here is an example using public peers and a static address. The static signing and encryption keys are defined in `/etc/yggdrasil-private.conf` (the default value for `config-file`).

```
;; part of the operating-system declaration
(service yggdrasil-service-type
               (yggdrasil-configuration
                (autoconf? #f) ;; use only the public peers
                (json-config
                ;; choose one from
                ;; https://github.com/yggdrasil-network/public-peers
                 '((peers . #("tcp://1.2.3.4:1337"))))
                ;; /etc/yggdrasil-private.conf is the default value for config-fil
                ))

# sample content for /etc/yggdrasil-private.conf
{
  # Your private key. DO NOT share this with anyone!
  PrivateKey: 5c750...
}
```

`yggdrasil-configuration` [Data Type]

Data type representing the configuration of Yggdrasil.

`package` (default: `yggdrasil`)
: Package object of Yggdrasil.

`json-config` (default: `'()`)
: Contents of `/etc/yggdrasil.conf`. Will be merged with `/etc/yggdrasil-private.conf`. Note that these settings are stored in

the Guix store, which is readable to all users. **Do not store your private keys in it**. See the output of `yggdrasil -genconf` for a quick overview of valid keys and their default values.

`autoconf?` (default: `#f`)
: Whether to use automatic mode. Enabling it makes Yggdrasil use a dynamic IP and peer with IPv6 neighbors.

`log-level` (predeterminado: `'info`)
: How much detail to include in logs. Use `'debug` for more detail.

`log-to` (default: `'stdout`)
: Where to send logs. By default, the service logs standard output to `/var/log/yggdrasil.log`. The alternative is `'syslog`, which sends output to the running syslog service.

`config-file` (default: `"/etc/yggdrasil-private.conf"`)
: What HJSON file to load sensitive data from. This is where private keys should be stored, which are necessary to specify if you don't want a randomized address after each restart. Use `#f` to disable. Options defined in this file take precedence over `json-config`. Use the output of `yggdrasil -genconf` as a starting point. To configure a static address, delete everything except PrivateKey option.

`ipfs-service-type` [Variable]
: The service type for connecting to the IPFS network (`https://ipfs.io`), a global, versioned, peer-to-peer file system. Pass it a `ipfs-configuration` to change the ports used for the gateway and API.

  Here's an example configuration, using some non-standard ports:

```
(service ipfs-service-type
         (ipfs-configuration
          (gateway "/ip4/127.0.0.1/tcp/8880")
          (api "/ip4/127.0.0.1/tcp/8881")))
```

`ipfs-configuration` [Data Type]
: Data type representing the configuration of IPFS.

  `package` (default: `go-ipfs`)
  : Package object of IPFS.

  `gateway` (default: `"/ip4/127.0.0.1/tcp/8082"`)
  : Address of the gateway, in 'multiaddress' format.

  `api` (default: `"/ip4/127.0.0.1/tcp/5001"`)
  : Address of the API endpoint, in 'multiaddress' format.

`keepalived-service-type` [Variable]
: This is the type for the Keepalived (`https://www.keepalived.org/`) routing software, `keepalived`. Its value must be an `keepalived-configuration` record as in this example for master machine:

```
(service keepalived-service-type
```

```
            (keepalived-configuration
              (config-file (local-file "keepalived-master.conf"))))
```

where `keepalived-master.conf`:

```
vrrp_instance my-group {
  state MASTER
  interface enp9s0
  virtual_router_id 100
  priority 100
  unicast_peer { 10.0.0.2 }
  virtual_ipaddress {
    10.0.0.4/24
  }
}
```

and for backup machine:

```
(service keepalived-service-type
         (keepalived-configuration
          (config-file (local-file "keepalived-backup.conf"))))
```

where `keepalived-backup.conf`:

```
vrrp_instance my-group {
  state BACKUP
  interface enp9s0
  virtual_router_id 100
  priority 99
  unicast_peer { 10.0.0.3 }
  virtual_ipaddress {
    10.0.0.4/24
  }
}
```

### 11.10.6 Actualizaciones no-atendidas

Guix proporciona un servicio para realizar *actualizaciones desatendidas*: periodicamente el sistema se reconfigura automáticamente con la última revisión de Guix. El sistema Guix tiene varias propiedades que hacen que las actualizaciones desatendidas sean seguras:

- las actualizaciones son transaccionales (o bien la actualización se lleva a cabo con éxito o bien falla, pero no puede acabar en un estado "intermedio" del sistema);
- el registro de actualizaciones se mantiene—puede acceder a dicho registro con `guix system list-generations`—y puede volver a una generación previa, en caso de que alguna generación no funcione como desee;
- el código del canal se verifica de modo que únicamente pueda ejecutar código que ha sido firmado (see Chapter 6 [Canales], page 69);
- `guix system reconfigure` evita la instalación de versiones previas, que lo hace inmune a *ataques de versión anterior*.

Para configurar las actualizaciones desatendidas, añada una instancia de `unattended-upgrade-service-type` como la que se muestra a continuación a la lista de servicios de su sistema operativo:

```
(service unattended-upgrade-service-type)
```

El valor predeterminado configura actualizaciones semanales: cada domingo a medianoche. No es necesario que proporcione el archivo de configuración de su sistema operativo: el servicio usa `/run/current-system/configuration.scm`, lo que asegura el uso de su última configuración—see [provenance-service-type], page 656, para obtener más información sobre este archivo.

Hay varios aspectos que se pueden configurar, en particular la periodicidad y los servicios (daemon) que se reiniciarán tras completar la actualización. Cuando la actualización se lleva a cabo satisfactoriamente el servicio se encarga de borrar las generaciones del sistema cuya antigüedad es superior a determinado valor, usando `guix system delete-generations`. Véase la referencia a continuación para obtener más detalles.

Para asegurar que las actualizaciones se están levando a cabo realmente, puede ejecutar `guix system describe`. Para investigar fallos en las actualizaciones, visite el archivo de registro de las actualizaciones desatendidas (véase a continuación).

`unattended-upgrade-service-type` [Variable]

Es el tipo de servicio para las actualizaciones desatendidas. Configura un trabajo de mcron (see Section 11.10.2 [Ejecución de tareas programadas], page 297) que ejecuta `guix system reconfigure` a partir de la última versión de los canales especificados.

Su valor debe ser un registro `unattended-upgrade-configuration` (véase a continuación).

`unattended-upgrade-configuration` [Tipo de datos]

Este tipo de datos representa la configuración del servicio de actualizaciones desatendidas. Los siguientes campos están disponibles:

`schedule` (predeterminada: `"30 01 * * 0"`)

La planificación de las actualizaciones, expresada en una expresión-G que contiene una planificación de trabajo de mcron (see Section “Syntax” in *GNU mcron*).

`channels` (predeterminada: `#~%default-channels`)

Esta expresión-G especifica los canales usados para la actualización (see Chapter 6 [Canales], page 69). De manera predeterminada, se usa la última revisión del canal oficial `guix`.

`operating-system-file` (predeterminado: `"/run/current-system/configuration.scm"`)

Este campo especifica el archivo de configuración del sistema operativo usado. El valor predeterminado reutiliza el archivo de configuración de la configuración actual.

No obstante hay casos en los que no es suficiente hacer referencia a `/run/current-system/configuration.scm`, por ejemplo porque dicho archivo hace referencia a archivos adicionales (claves públicas de SSH, archivos de configuración adicionales, etcétera) a través de `local-file` y construcciones similares. Para estos casos recomendamos una configuración parecida a esta:

```
(unattended-upgrade-configuration
```

```
(operating-system-file
  (file-append (local-file "." "config-dir" #:recursive? #t)
               "/config.scm")))
```

El efecto que esto tiene es la importación del directorio actual al completo en el almacén, y hace referencia a `config.scm` dentro de dicho directorio. Por lo tanto, los usos de `local-file` dentro de `config.scm` funcionarán como se espera. See Section 8.12 [Expresiones-G], page 167, para más información acerca de `local-file` y `file-append`.

`operating-system-expression` (default: `#f`)
: This field specifies an expression that evaluates to the operating system to use for the upgrade. If no value is provided the `operating-system-file` field value is used.

```
(unattended-upgrade-configuration
  (operating-system-expression
    #~(@ (guix system install) installation-os)))
```

`services-to-restart` (predeterminados: `'(mcron)`)
: Este campo especifica los servicios de Shepherd que se reiniciarán cuando se complete una actualización.

  Estos servicios se reinician tras completarse la actualización, como ejecutando `herd restart`, lo que asegura la ejecución de la última versión—recuerde que de manera predeterminada `guix system reconfigure` únicamente reinicia los servicios que no se están ejecutando en este momento, lo que es una aproximación conservadora: minimiza la disrupción pero mantiene servicios en ejecución con código previo a la actualización.

  Use `herd status` to find out candidates for restarting. See Section 11.10 [Servicios], page 275, for general information about services. Common services to restart would include `ntpd` and `ssh-daemon`.

  De manera predeterminada se reinicia el servicio `mcron`. Esto asegura que la última versión del trabajo de actualización desatendida será la que se use la próxima vez.

`system-expiration` (predeterminado: `(* 3 30 24 3600)`)
: Tiempo de expiración en segundos de las generaciones del sistema. Las generaciones del sistema cuya antigüedad sea mayor que esta cantidad de tiempo se borran con `guix system delete-generations` cuando se completa una actualización.

  > **Nota:** El servicio de actualizaciones desatendidas no ejecuta el proceso de recolección de basura. Probablemente quiera añadir su propio trabajo a mcron para ejecutar `guix gc` de manera periódica.

`maximum-duration` (predeterminado: `3600`)
: Duración máxima en segundos de la actualización; tras pasar este tiempo se aborta la actualización.

Esto es útil principalmente para asegurar que una actualización no reconstruye o vuelve a descargarse "el mundo entero".

`log-file` (predeterminado: `"/var/log/unattended-upgrade.log"`)
Archivo donde se registran las actualizaciones desatendidas.

### 11.10.7 Sistema X Window

El sistema gráfico X Window—específicamente Xorg—se proporciona en el módulo `(gnu services xorg)`. Fíjese que no existe un procedimiento `xorg-service`. En vez de eso, el servidor X se inicia por el *gestor de ingreso al sistema*, de manera predeterminada el gestor de acceso de GNOME (GDM).

GDM por supuesto que permite a las usuarias ingresar al sistema con gestores de ventanas y entornos de escritorio distintos a GNOME; para aquellas que usan GNOME, GDM es necesario para características como el bloqueo automático de pantalla.

Para usar X11, debe instalar al menos un *gestor de ventanas*—por ejemplo los paquetes `windowmaker` o `openbox`—, preferiblemente añadiendo el que desee al campo `packages` de su definición de sistema operativo (see Section 11.3 [Referencia de operating-system], page 253).

GDM also supports Wayland: it can itself use Wayland instead of X11 for its user interface, and it can also start Wayland sessions. The former is required for the latter, to enable, set `wayland?` to `#t` in `gdm-configuration`.

`gdm-service-type` [Variable]
Este es el tipo para el gestor de acceso de GNOME (`https://wiki.gnome.org/Projects/GDM/`) (GDM), un programa que gestiona servidores gráficos y maneja de forma gráfica el ingreso al sistema de usuarias. Su valor debe ser un`gdm-configuration` (véase a continuación).

GDM looks for *session types* described by the `.desktop` files in `/run/current-system/profile/share/xsessions` (for X11 sessions) and `/run/current-system/profile/share/wayland-sessions` (for Wayland sessions) and allows users to choose a session from the log-in screen. Packages such as `gnome`, `xfce`, `i3` and `sway` provide `.desktop` files; adding them to the system-wide set of packages automatically makes them available at the log-in screen.

Además, se respetan los archivos `~/.xsession`. Cuando esté disponible, `~/.xsession` debe ser un ejecutable que inicie un gestor de ventanas y/o otros clientes de X.

`gdm-configuration` [Tipo de datos]
`auto-login?` (predeterminado: `#f`)
`default-user` (predeterminado: `#f`)
Cuando `auto-login?` es falso, GDM presenta una pantalla de ingreso.

Cuando `auto-login?` es verdadero, GDM ingresa directamente al sistema como `default-user`.

`auto-suspend?` (default `#t`)
When true, GDM will automatically suspend to RAM when nobody is physically connected. When a machine is used via remote desktop or SSH, this should be set to false to avoid GDM interrupting remote sessions or rendering the machine unavailable.

`debug?` (predeterminado: `#f`)
: Cuando tiene valor verdadero, GDM escribe los mensajes de depuración en su registro.

`gnome-shell-assets` (predeterminados: ...)
: Lista de activos de GNOME Shell necesarios para GDM: tema de iconos, fuentes, etc.cc

`xorg-configuration` (predeterminada: `(xorg-configuration)`)
: Configuración del servidor gráfico Xorg.

`x-session` (default: `(xinitrc)`)
: Guión a ejecutar antes de iniciar una sesión X.

`xdmcp?` (default: `#f`)
: When true, enable the X Display Manager Control Protocol (XDMCP). This should only be enabled in trusted environments, as the protocol is not secure. When enabled, GDM listens for XDMCP queries on the UDP port 177.

`dbus-daemon` (predeterminado: `dbus-daemon-wrapper`)
: El nombre de archivo del ejecutable `dbus-daemon`.

`gdm` (predeterminado: `gdm`)
: El paquete GDM usado.

`wayland?` (default: `#f`)
: When true, enables Wayland in GDM, necessary to use Wayland sessions.

`wayland-session` (default: `gdm-wayland-session-wrapper`)
: The Wayland session wrapper to use, needed to setup the environment.

`slim-service-type` [Variable]

Este es el tipo para el gestor de ingreso al sistema gráfico para X11 SLiM.

Como GDM, SLiM busca tipos de sesión descritos por archivos `.desktop` y permite a las usuarias la selección de sesión en la pantalla de ingreso al sistema mediante el uso de *F1*. También respeta los archivos `~/.xsession`.

Al contrario que GDM, SLiM no lanza las sesiones de las usuarias en terminales virtuales diferentes al usado para el ingreso, lo que significa que únicamente puede iniciar una sesión gráfica. Si desea ejecutar varias sesiones gráficas de manera simultánea, debe añadir múltiples servicios de SLiM a los servicios de su sistema. El ejemplo siguiente muestra cómo sustituir el servicio GDM predeterminado con dos servicios de SLiM en tty7 y tty8.

```
(use-modules (gnu services)
             (gnu services desktop)
             (gnu services xorg))

(operating-system
  ;; ...
  (services (cons* (service slim-service-type (slim-configuration
                                               (display ":0")
```

```
                                    (vt "vt7")))
                (service slim-service-type (slim-configuration
                                    (display ":1")
                                    (vt "vt8")))
                (modify-services %desktop-services
                  (delete gdm-service-type)))))
```

`slim-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `slim-service-type`.

`allow-empty-passwords?` (predeterminado: `#t`)
: Si se permite el ingreso al sistema con contraseñas vacías.

`gnupg?` (default: `#f`)
: If enabled, `pam-gnupg` will attempt to automatically unlock the user's GPG keys with the login password via `gpg-agent`. The keygrips of all keys to be unlocked should be written to `~/.pam-gnupg`, and can be queried with `gpg -K --with-keygrip`. Presetting passphrases must be enabled by adding `allow-preset-passphrase` in `~/.gnupg/gpg-agent.conf`.

`auto-login?` (predeterminado: `#f`)
`default-user` (predeterminado: `""`)
: Cuando `auto-login?` es falso, SLiM presenta una pantalla de ingreso.

  Cuando `auto-login?` es verdadero, SLiM ingresa en el sistema directamente como `default-user`.

`theme` (predeterminado: `%default-slim-theme`)
`theme-name` (predeterminado: `%default-slim-theme-name`)
: El tema gráfico usado y su nombre.

`auto-login-session` (predeterminado: `#f`)
: Si es verdadero, debe ser el nombre del ejecutable a arrancar como la sesión predeterminada—por ejemplo, `(file-append windowmaker "/bin/windowmaker")`.

  Si es falso, se usará una sesión de las descritas en uno de los archivos `.desktop` disponibles en `/run/current-system/profile` y `~/.guix-profile`.

  > **Nota:** Debe instalar al menos un gestor de ventanas en el perfil del sistema o en su perfil de usuaria. En caso de no hacerlo, si `auto-login-session` es falso, no podrá ingresar al sistema.

`xorg-configuration` (predeterminada `(xorg-configuration)`)
: Configuración del servidor gráfico Xorg.

`display` (predeterminada: `":0"`)
: La pantalla en la que se iniciará el servidor gráfico Xorg.

`vt` (predeterminado: `"vt7"`)
: El terminal virtual (VT) en el que se iniciará el servidor gráfico Xorg.

`xauth` (predeterminado: `xauth`)
: El paquete XAuth usado.

`shepherd` (predeterminado: `shepherd`)
: El paquete de Shepherd usado para la invocación de `halt` y `reboot`.

`sessreg` (predeterminado: `sessreg`)
: El paquete sessreg usado para el registro de la sesión.

`slim` (predeterminado: `slim`)
: El paquete SLiM usado.

`%default-theme` [Variable]
`%default-theme-name` [Variable]
El tema predeterminado de SLiM y su nombre.

`sddm-service-type` [Variable]
Es el tipo del servicio que ejecuta el gestor de entrada SDDM (`https://github.com/sddm/sddm`). Su valor es un registro `sddm-configuration` (véase a continuación).

Este es un ejemplo de su uso:

```
(service sddm-service-type
         (sddm-configuration
           (auto-login-user "alicia")
           (auto-login-session "xfce.desktop")))
```

`sddm-configuration` [Tipo de datos]
Este tipo de datos representa la configuración del gestor de ingreso al sistema SDDM. Los campos disponibles son:

`sddm` (predeterminado: `sddm`)
: El paquete SDDM usado.

`display-server` (predeterminado: `"x11"`)
: Selecciona el servidor de pantalla usado para el saludo. Los valores validos son ‘`"x11"`’ o ‘`"wayland"`’.

`numlock` (predeterminado: `"on"`)
: Son valores válidos ‘`"on"`’, ‘`"off"`’ o ‘`"none"`’.

`halt-command` (default `#~(string-append #$shepherd "/sbin/halt")`)
: Orden a ejecutar para parar el sistema.

`reboot-command` (predeterminado `#~(string-append #$shepherd "/sbin/reboot")`)
: Orden a ejecutar para reiniciar el sistema.

`theme` (predeterminado `"maldives"`)
: Tema usado. Los temas predeterminados proporcionados por SDDM son ‘`"elarun"`’, ‘`"maldives"`’ o ‘`"maya"`’.

`themes-directory` (predeterminado `"/run/current-system/profile/share/sddm/themes"`)
: Directorio en el que buscar temas.

`faces-directory` (predeterminado `"/run/current-system/profile/share/sddm/faces"`)
: Directorio en el que buscar caras.

`default-path` (predeterminado `"/run/current-system/profile/bin"`)
: El valor predeterminado del PATH.

`minimum-uid` (predeterminado: 1000)
: UID mínimo mostrado en SDDM y al que se le permite el acceso.

`maximum-uid` (predeterminado: 2000)
: UID máximo mostrado en SDDM.

`remember-last-user?` (predeterminado #t)
: Recuerda la última usuaria.

`remember-last-session?` (predeterminado #t)
: Recuerda la última sesión.

`hide-users` (predeterminado `""`)
: Nombres de usuaria a ocultar de la pantalla de inicio de SDDM.

`hide-shells` (predeterminado `#~(string-append #$shadow "/sbin/nologin")`)
: Las usuarias que tengan alguno de los shell enumerados se ocultarán de la pantalla de inicio de SDDM.

`session-command` (predeterminado `#~(string-append #$sddm "/share/sddm/scripts/wayland-session")`)
: Guión a ejecutar antes de iniciar una sesión wayland.

`sessions-directory` (predeterminado `"/run/current-system/profile/share/wayland-sessions"`)
: Directorio en el que buscar archivos desktop que inicien sesiones wayland.

`xorg-configuration` (predeterminada `(xorg-configuration)`)
: Configuración del servidor gráfico Xorg.

`xauth-path` (predeterminado `#~(string-append #$xauth "/bin/xauth")`)
: Ruta de xauth.

`xephyr-path` (predeterminado `#~(string-append #$xorg-server "/bin/Xephyr")`)
: Ruta de Xephyr.

`xdisplay-start` (predeterminado `#~(string-append #$sddm "/share/sddm/scripts/Xsetup")`)
: Guión a ejecutar tras iniciar xorg-server.

`xdisplay-stop` (predeterminado `#~(string-append #$sddm "/share/sddm/scripts/Xstop")`)
: Guión a ejecutar antes de parar xorg-server.

`xsession-command` (predeterminado: `xinitrc`)
: Guión a ejecutar antes de iniciar una sesión X.

`xsessions-directory` (predeterminado: "/run/current-system/profile/share/xsessions")
: Directorio para buscar archivos desktop que inicien sesiones X.

`minimum-vt` (predeterminado: 7)
: VT mínimo usado.

`auto-login-user` (predeterminado "")
: User account that will be automatically logged in. Setting this to the empty string disables auto-login.

`auto-login-session` (predeterminado "")
: The `.desktop` file name to use as the auto-login session, or the empty string.

`relogin?` (predeterminado #f)
: Volver a ingresar en el sistema tras salir.

`lightdm-service-type` [Variable]

This is the type of the service to run the LightDM display manager (`https://github.com/canonical/lightdm`). Its value must be a `lightdm-configuration` record, which is documented below. Among its distinguishing features are TigerVNC integration for easily remoting your desktop as well as support for the XDMCP protocol, which can be used by remote clients to start a session from the login manager.

In its most basic form, it can be used simply as:

```
(service lightdm-service-type)
```

A more elaborate example making use of the VNC capabilities and enabling more features and verbose logs could look like:

```
(service lightdm-service-type
         (lightdm-configuration
          (allow-empty-passwords? #t)
          (xdmcp? #t)
          (vnc-server? #t)
          (vnc-server-command
           (file-append tigervnc-server "/bin/Xvnc"
                        "  -SecurityTypes None"))
          (seats
           (list (lightdm-seat-configuration
                  (name "*")
                  (user-session "ratpoison"))))))
```

`lightdm-configuration` [Data Type]

Available `lightdm-configuration` fields are:

`lightdm` (default: `lightdm`) (type: file-like)
: The lightdm package to use.

`allow-empty-passwords?` (default: `#f`) (type: boolean)
: Whether users not having a password set can login.

`debug?` (default: `#f`) (type: boolean)
Enable verbose output.

`xorg-configuration` (type: xorg-configuration)
The default Xorg server configuration to use to generate the Xorg server start script. It can be refined per seat via the `xserver-command` of the `<lightdm-seat-configuration>` record, if desired.

`greeters` (type: list-of-greeter-configurations)
The LightDM greeter configurations specifying the greeters to use.

`seats` (type: list-of-seat-configurations)
The seat configurations to use. A LightDM seat is akin to a user.

`xdmcp?` (default: `#f`) (type: boolean)
Whether a XDMCP server should listen on port UDP 177.

`xdmcp-listen-address` (type: maybe-string)
The host or IP address the XDMCP server listens for incoming connections. When unspecified, listen on for any hosts/IP addresses.

`vnc-server?` (default: `#f`) (type: boolean)
Whether a VNC server is started.

`vnc-server-command` (type: file-like)
The Xvnc command to use for the VNC server, it's possible to provide extra options not otherwise exposed along the command, for example to disable security:

```
(vnc-server-command (file-append tigervnc-server "/bin/Xvnc"
                                 " -SecurityTypes None" ))
```

Or to set a PasswordFile for the classic (unsecure) VncAuth mechanism:

```
(vnc-server-command (file-append tigervnc-server "/bin/Xvnc"
                                 " -PasswordFile /var/lib/lightdm/.vnc/
```

The password file should be manually created using the `vncpasswd` command. Note that LightDM will create new sessions for VNC users, which means they need to authenticate in the same way as local users would.

`vnc-server-listen-address` (type: maybe-string)
The host or IP address the VNC server listens for incoming connections. When unspecified, listen for any hosts/IP addresses.

`vnc-server-port` (default: `5900`) (type: number)
The TCP port the VNC server should listen to.

`extra-config` (default: `'()`) (type: list-of-strings)
Extra configuration values to append to the LightDM configuration file.

`lightdm-gtk-greeter-configuration` [Data Type]
Available `lightdm-gtk-greeter-configuration` fields are:

`lightdm-gtk-greeter` (default: `lightdm-gtk-greeter`) (type: file-like)
The lightdm-gtk-greeter package to use.

`assets` (default: `(adwaita-icon-theme gnome-themes-extra hicolor-icon-theme)`) (type: list-of-file-likes)
The list of packages complementing the greeter, such as package providing icon themes.

`theme-name` (default: `"Adwaita"`) (type: string)
The name of the theme to use.

`icon-theme-name` (default: `"Adwaita"`) (type: string)
The name of the icon theme to use.

`cursor-theme-name` (default: `"Adwaita"`) (type: string)
The name of the cursor theme to use.

`cursor-theme-size` (default: `16`) (type: number)
The size to use for the cursor theme.

`allow-debugging?` (type: maybe-boolean)
Set to #t to enable debug log level.

`background` (type: file-like)
The background image to use.

`at-spi-enabled?` (default: `#f`) (type: boolean)
Enable accessibility support through the Assistive Technology Service Provider Interface (AT-SPI).

`a11y-states` (default: `(contrast font keyboard reader)`) (type: list-of-a11y-states)
The accessibility features to enable, given as list of symbols.

`reader` (type: maybe-file-like)
The command to use to launch a screen reader.

`extra-config` (default: `'()`) (type: list-of-strings)
Extra configuration values to append to the LightDM GTK Greeter configuration file.

`lightdm-seat-configuration` [Data Type]
Available `lightdm-seat-configuration` fields are:

`name` (type: seat-name)
The name of the seat. An asterisk (*) can be used in the name to apply the seat configuration to all the seat names it matches.

`user-session` (type: maybe-string)
The session to use by default. The session name must be provided as a lowercase string, such as `"gnome"`, `"ratpoison"`, etc.

`type` (default: `local`) (type: seat-type)
The type of the seat, either the `local` or `xremote` symbol.

`autologin-user` (type: maybe-string)
The username to automatically log in with by default.

`greeter-session` (default: `lightdm-gtk-greeter`) (type: greeter-session)
: The greeter session to use, specified as a symbol. Currently, only `lightdm-gtk-greeter` is supported.

`xserver-command` (type: maybe-file-like)
: The Xorg server command to run.

`session-wrapper` (type: file-like)
: The xinitrc session wrapper to use.

`extra-config` (default: `'()`) (type: list-of-strings)
: Extra configuration values to append to the seat configuration section.

`xorg-configuration` [Tipo de datos]

This data type represents the configuration of the Xorg graphical display server. Note that there is no Xorg service; instead, the X server is started by a "display manager" such as GDM, SDDM, LightDM or SLiM. Thus, the configuration of these display managers aggregates an `xorg-configuration` record.

`modules` (predeterminados: `%default-xorg-modules`)
: Esta es la lista de *paquetes de módulos* cargados por el servidor Xorg—por ejemplo, `xf86-video-vesa`, `xf86-input-keyboard`, etcétera.

`fonts` (predeterminadas: `%default-xorg-fonts`)
: Es una lista de directorios de tipografías a añadir a la *ruta de tipografías* del servidor.

`drivers` (predeterminados: `'()`)
: Debe ser o bien la lista vacía, en cuyo caso Xorg selecciona el controlador gráfico automáticamente, o una lista de nombres de controladores que se intentarán en el orden especificado—por ejemplo, `'("modesetting" "vesa")`.

`resolutions` (predeterminadas: `'()`)
: Cuando `resolutions` es la lista vacía, Xorg selecciona una resolución de pantalla adecuada. En otro caso, debe ser una lista de resoluciones—por ejemplo, `'((1024 768) (640 480))`.

`keyboard-layout` (predeterminada: `#f`)
: Si es `#f`, Xorg usa la distribución de teclado predeterminada—normalmente inglés de EEUU ("qwerty") para un teclado de PC de 105 teclas.

  En otro caso, debe ser un objeto `keyboard-layout` que especifique la distribución de teclado usada para la ejecución de Xorg. See Section 11.8 [Distribución de teclado], page 271, para más información sobre cómo especificar la distribución de teclado.

`extra-config` (predeterminada: `'()`)
: Es una lista de cadenas u objetos añadida al final del archivo de configuración. Se usa para proporcionar texto adicional para ser introducido de forma literal en el archivo de configuración.

`server` (predeterminado: `xorg-server`)
: Este es el paquete que proporciona el servidor Xorg.

`server-arguments` (predeterminados: `%default-xorg-server-arguments`)
: Es la lista de parámetros de línea de órdenes que se proporcionarán al servidor X. El valor predeterminado es `-nolisten tcp`.

`set-xorg-configuration` *config* [*login-manager-service-type*] [Procedure]
Tell the log-in manager (of type *login-manager-service-type*) to use *config*, an `<xorg-configuration>` record.

Debido a que la configuración de Xorg se embebe en la configuración del gestor de ingreso en el sistema—por ejemplo, `gdm-configuration`—este procedimiento proporciona un atajo para establecer la configuración de Xorg.

`xorg-start-command` [*config*] [Procedure]
Devuelve un script `startx` en el que los módulos, las tipografías, etcétera, especificadas en *config* están disponibles. El resultado debe usarse en lugar de `startx`.

Habitualmente el servidor X es iniciado por un gestor de ingreso al sistema.

`xorg-start-command-xinit` [*config*] [Procedure]
Return a `startx` script in which the modules, fonts, etc. specified in *config* are available. The result should be used in place of `startx` and should be invoked by the user from a tty after login. Unlike `xorg-start-command`, this script calls xinit. Therefore it works well when executed from a tty. This script can be set up as `startx` using `startx-command-service-type` or `home-startx-command-service-type`. If you are using a desktop environment, you are unlikely to need this procedure.

`screen-locker-service-type` [Variable]
Type for a service that adds a package for a screen locker or screen saver to the set of setuid programs and/or add a PAM entry for it. The value for this service is a `<screen-locker-configuration>` object.

While the default behavior is to setup both a setuid program and PAM entry, these two methods are redundant. Screen locker programs may not execute when PAM is configured and `setuid` is set on their executable. In this case, `using-setuid?` can be set to `#f`.

For example, to make XlockMore usable:

```
(service screen-locker-service-type
         (screen-locker-configuration
           (name "xlock")
           (program (file-append xlockmore "/bin/xlock"))))
```

permite usar el viejo XlockMore.

For example, swaylock fails to execute when compiled with PAM support and setuid enabled. One can thus disable setuid:

```
(service screen-locker-service-type
         (screen-locker-configuration
           (name "swaylock")
           (program (file-append swaylock "/bin/swaylock"))
```

```
                    (using-pam? #t)
                    (using-setuid? #f)))
```

`screen-locker-configuration` [Data Type]

Available `screen-locker-configuration` fields are:

`name` (type: string)
: Name of the screen locker.

`program` (type: file-like)
: Path to the executable for the screen locker as a G-Expression.

`allow-empty-password?` (default: `#f`) (type: boolean)
: Si se permiten las contraseñas vacías.

`using-pam?` (default: `#t`) (type: boolean)
: Whether to setup PAM entry.

`using-setuid?` (default: `#t`) (type: boolean)
: Whether to setup program as setuid binary.

`startx-command-service-type` [Variable]

Add `startx` to the system profile putting it onto `PATH`.

The value for this service is a `<xorg-configuration>` object which is passed to the `xorg-start-command-xinit` procedure producing the `startx` used. Default value is `(xorg-configuration)`.

### 11.10.8 Servicios de impresión

El módulo `(gnu services cups)` proporciona una definición de servicio Guix para el servicio de impresión CUPS. Para usar impresoras en un sistema Guix, añada un servicio `cups-service` en su definición de sistema operativo:

`cups-service-type` [Variable]

El tipo de servicio para el servidor de impresión CUPS. Su valor debe ser una configuración de CUPS válida (véase a continuación). Para usar la configuración predeterminada, simplemente escriba:

```
(service cups-service-type)
```

La configuración de CUPS controla los aspectos básicos de su instalación de CUPS: sobre qué interfaces se escuchará, qué hacer si falla un trabajo de impresión, cuanta información registrar, etcétera. Para realmente añadir una impresora, debe visitar la URL `http://localhost:631`, o usar una herramienta como los servicios de configuración de impresión de GNOME. De manera predeterminada, la configuración de un servicio CUPS generará un certificado auto-firmado en caso de ser necesario, para ofrecer conexiones seguras con el servidor de impresión.

Suppose you want to enable the Web interface of CUPS and also add support for Epson printers *via* the `epson-inkjet-printer-escpr` package and for HP printers *via* the `hplip-minimal` package. You can do that directly, like this (you need to use the `(gnu packages cups)` module):

```
(service cups-service-type
```

```
(cups-configuration
  (web-interface? #t)
  (extensions
    (list cups-filters epson-inkjet-printer-escpr hplip-minimal))))
```

**Nota:** If you wish to use the Qt5 based GUI which comes with the hplip package then it is suggested that you install the `hplip` package, either in your OS configuration file or as your user.

A continuación se encuentran los parámetros de configuración disponibles. El tipo de cada parámetro antecede la definición del mismo; por ejemplo, '`string-list foo`' indica que el parámetro `foo` debe especificarse como una lista de cadenas. También existe la posibilidad de especificar la configuración como una cadena, si tiene un archivo `cupsd.conf` antiguo que quiere trasladar a otro sistema; véase el final para más detalles.

Los campos disponibles de `cups-configuration` son:

`package cups` [parámetro de `cups-configuration`]
El paquete CUPS.

`package-list extensions` (*default:* (`list brlaser cups-filters epson-inkjet-printer-escpr foomatic-filters hplip-minimal splix`)) [`cups-configuration` parameter]
Controladores y otras extensiones al paquete CUPS.

`archivos-conf files-configuration` [parámetro de `cups-configuration`]
Configuración sobre dónde escribir los registros, qué directorios usar para las colas de impresión y parámetros de configuración privilegiados relacionados.

Los campos disponibles de `files-configuration` son:

`ruta-registro access-log` [parámetro de `files-configuration`]
Define el nombre de archivo del registro de acceso. La especificación de un nombre de archivo en blanco desactiva la generación de registros de acceso. El valor `stderr` hace que las entradas de registro se envíen al archivo de la salida estándar de error cuando el planificador se ejecute en primer plano, o al daemon de registro del sistema cuando se ejecute en segundo plano. El valor `syslog` envía las entradas de registro al daemon de registro del sistema. El nombre de servidor puede incluirse en los nombres de archivo mediante el uso de la cadena `%s`, como en `/var/log/cups/%s-access_log`.

El valor predeterminado es '`"/var/log/cups/access_log"`'.

`nombre-archivo cache-dir` [parámetro de `files-configuration`]
Donde CUPS debe almacenar los datos de la caché.

El valor predeterminado es '`"/var/cache/cups"`'.

`string config-file-perm` [parámetro de `files-configuration`]
Especifica los permisos para todos los archivos de configuración que escriba el planficador.

Tenga en cuenta que los permisos para el archivo printers.conf están configurados actualmente de modo que únicamente la usuaria del planificador (habitualmente root) tenga acceso. Se hace de esta manera debido a que las URI de las

impresoras a veces contienen información sensible sobre la identificación que no debería conocerse de manera general en el sistema. No hay forma de desactivar esta característica de seguridad.

El valor predeterminado es '`"0640"`'.

`ruta-registro error-log` [parámetro de `files-configuration`]
Define el nombre de archivo del registro de error. La especificación de un nombre de archivo en blanco desactiva la generación de registros de error. El valor `stderr` hace que las entradas del registro se envíen al archivo de la salida de error estándar cuando el planificador se ejecute en primer plano, o al daemon de registro del sistema cuando se ejecute en segundo plano. El valor `syslog` provoca que las entradas del registro se envíen al daemon de registro del sistema. El nombre del servidor puede incluirse en los nombres de archivo mediante el uso de la cadena `%s`, como en `/var/log/cups/%s-error_log`.

El valor predeterminado es '`"/var/log/cups/error_log"`'.

`string fatal-errors` [parámetro de `files-configuration`]
Especifica qué errores son fatales, los cuales provocan la salida del planificador. El tipo de cadenas son:

`none` Ningún error es fatal.

`all` Todos los errores a continuación son fatales.

`browse` Los errores de la inicialización de exploración son fatales, por ejemplo las conexiones fallidas al daemon DNS-SD.

`config` Los errores de sintaxis en el archivo de configuración son fatales.

`listen` Los errores de escucha o de puertos son fatales, excepto fallos IPv6 en la red local o en direcciones `any`.

`log` Los errores de creación o escritura en el archivo de registros son fatales.

`permissions`
La mala configuración de los permisos de los archivos al inicio son fatales, por ejemplo certificados TLS compartidos y archivos de claves con permisos de escritura para todo el mundo.

El valor predeterminado es '`"all -browse"`'.

`boolean file-device?` [parámetro de `files-configuration`]
Especifica si el pseudo-dispositivo del archivo puede usarse para nuevas colas de impresión. Siempre se permite la URI `file:///dev/null`.

El valor predeterminado es '`#f`'

`string group` [parámetro de `files-configuration`]
Especifica el nombre de grupo o ID usado para la ejecución de programas externos.

El valor predeterminado es '`"lp"`'.

`string log-file-group` [`files-configuration` parameter]
Specifies the group name or ID that will be used for log files.

Defaults to '`"lpadmin"`'.

`string log-file-perm` [parámetro de `files-configuration`]
Especifica los permisos para todos los archivos de registro que el planificador escriba.

El valor predeterminado es '`"0644"`'.

`ruta-registro page-log` [parámetro de `files-configuration`]
Define el nombre de archivo del registro de páginas. La especificación de un nombre de archivo en blanco desactiva la generación de registro de páginas. El valor `stderr` hace que las entradas del registro se envíen al archivo de la salida de error cuando el planificador se ejecute en primer plano, o al daemon de registro del sistema cuando se ejecuten en segundo plano. El valor `syslog` provoca que las entradas del registro se envíen al daemon de registro del sistema. El nombre del servidor puede incluirse en los nombres de archivo mediante el uso de la cadena `%s`, como en `/var/log/cups/%s-page_log`.

El valor predeterminado es '`"/var/log/cups/page_log"`'.

`string remote-root` [parámetro de `files-configuration`]
Especifica el nombre de la usuaria asociado con accesos sin identificación por parte de clientes que digan ser la usuaria root. La usuaria predeterminada es `remroot`.

El valor predeterminado es '`"remroot"`'.

`nombre-archivo request-root` [parámetro de `files-configuration`]
Especifica el directorio que contiene los trabajos de impresión y otros datos de peticiones HTTP.

El valor predeterminado es '`"/var/spool/cups"`'.

`aislamiento sandboxing` [parámetro de `files-configuration`]
Especifica el nivel de seguridad del aislamiento (sandbox) que se aplica sobre los filtros de impresión, motores y otros procesos lanzados por el planificador; o bien `relaxed` o bien `strict`. Esta directiva únicamente tiene uso actualmente en macOS.

El valor predeterminado es '`strict`'.

`nombre-archivo server-keychain` [parámetro de `files-configuration`]
Especifica la localización de los certificados TLS y las claves privadas. CUPS buscará claves públicas y privadas en este directorio: un archivo `.crt` para certificados codificados con PEM y los correspondientes archivo `.key` para las claves privadas codificadas con PEM.

El valor predeterminado es '`"/etc/cups/ssl"`'.

`nombre-archivo server-root` [parámetro de `files-configuration`]
Especifica el directorio que contiene los archivos de configuración del servidor.

El valor predeterminado es '`"/etc/cups"`'.

`boolean sync-on-close?` [parámetro de `files-configuration`]
Especifica si el planificador llama fsync(2) tras la escritura de los archivos de configuración o estado.

El valor predeterminado es '`#f`'

`cadenas-separadas-por-espacios system-group` [parámetro de `files-configuration`]
Especifica el o los grupos usados para la identificación del grupo `@SYSTEM`.

`nombre-archivo temp-dir` [parámetro de `files-configuration`]
Especifica el directorio donde se escriben los archivos temporales.

El valor predeterminado es '`"/var/spool/cups/tmp"`'.

`string user` [parámetro de `files-configuration`]
Especifica el nombre de usuaria o ID usado para la ejecución de programas externos.

El valor predeterminado es '`"lp"`'.

`string set-env` [parámetro de `files-configuration`]
Establece el valor de la variable de entorno especificada que se proporcionará a los procesos lanzados.

El valor predeterminado es '`"variable value"`'.

`nivel-registro-acceso access-log-level` [parámetro de `cups-configuration`]
Especifica el nivel de registro para el archivo AccessLog. El nivel `config` registra la adición, borrado o modificación de impresoras y clases, y el acceso y modificación de los archivos de configuración. El nivel `actions` registra cuando los trabajos de impresión se envían, mantienen a la espera, liberan, modifican o cancelan, además de todas las condiciones de `config`. El nivel `all` registra todas las peticiones.

El valor predeterminado es '`actions`'.

`boolean auto-purge-jobs?` [parámetro de `cups-configuration`]
Especifica si se purgan los datos del histórico de trabajos de manera automática cuando ya no son necesarios para las cuotas.

El valor predeterminado es '`#f`'

`lista-cadenas-separada-comas browse-dns-sd-sub-types` [parámetro de `cups-configuration`]
Specifies a list of DNS-SD sub-types to advertise for each shared printer.

The default '`(list "_cups" "_print" "_universal")`' tells clients that CUPS sharing, IPP Everywhere, AirPrint, and Mopria are supported.

`protocolos browse-local-protocols` [parámetro de `cups-configuration`]
Especifica qué protocolos deben usarse para compartir las impresoras locales.

El valor predeterminado es '`dnssd`'.

`boolean browse-web-if?` [parámetro de `cups-configuration`]
Especifica si se anuncia la interfaz web de CUPS.

El valor predeterminado es '`#f`'

`boolean browsing?` [parámetro de `cups-configuration`]
Especifica si se anuncian las impresoras compartidas.

El valor predeterminado es '`#f`'

`tipo-id-pred default-auth-type` [parámetro de `cups-configuration`]
Especifica el tipo de identificación usado por omisión.

El valor predeterminado es '`Basic`'.

`cifrado-pred default-encryption` [parámetro de `cups-configuration`]
Especifica si se usará cifrado para peticiones con identificación.

El valor predeterminado es '`Required`'.

`string default-language` [parámetro de `cups-configuration`]
Especifica el idioma predeterminado usado para el texto y contenido de la web.

El valor predeterminado es '`"en"`'.

`cadena default-paper-size` [parámetro de `cups-configuration`]
Especifica el tamaño predeterminado del papel para colas de impresión nuevas. '`"Auto"`' usa el valor predeterminado de la localización, mientras que '`"None"`' especifica que no hay un tamaño de papel predeterminado. Los nombres de tamaños específicos habitualmente son '`"Letter"`' o '`"A4"`'[7].

El valor predeterminado es '`"Auto"`'.

`string default-policy` [parámetro de `cups-configuration`]
Especifica la política de acceso usada por omisión.

El valor predeterminado es '`"default"`'.

`boolean default-shared?` [parámetro de `cups-configuration`]
Especifica si las impresoras locales se comparten de manera predeterminada.

El valor predeterminado es '`#t`'

`entero-no-negativo` [parámetro de `cups-configuration`]
`dirty-clean-interval`
Especifica el retraso para la actualización de los archivos de configuración y estado, en segundo. Un valor de 0 hace que la actualización se lleve a cabo tan pronto sea posible, en algunos milisegundos habitualmente.

El valor predeterminado es '`30`'.

[7] NdT: '`Letter`' es el formato estándar de ANSI, de 215,9x279,4 milímetros de tamaño, mientras que A4 es el formato estándar de ISO, de 210x297 milímetros de tamaño.

`política-error error-policy` [parámetro de `cups-configuration`]
Especifica qué hacer cuando ocurra un error. Los valores posibles son `abort-job`, que descartará el trabajo de impresión fallido; `retry-job`, que intentará llevar de nuevo a cabo el trabajo en un momento posterior; `retry-current-job`, que reintenta el trabajo que falló de manera inmediata; y `stop-printer`, que para la impresora.

El valor predeterminado es '`stop-printer`'.

`entero-no-negativo filter-limit` [parámetro de `cups-configuration`]
Especifica el coste máximo de filtros que se ejecutan de manera concurrente, lo que puede usarse para minimizar problemas de recursos de disco, memoria y procesador. Un límite de 0 desactiva la limitación del filtrado. Una impresión media con una impresora no-PostScript necesita una limitación del filtrado de 200 más o menos. Una impresora PostScript necesita cerca de la mitad (100). Establecer un límite por debajo de estos valores limitará de forma efectiva al planificador a la ejecución de un único trabajo de impresión en cualquier momento.

El valor predeterminado es '`0`'.

`entero-no-negativo filter-nice` [parámetro de `cups-configuration`]
Especifica la prioridad de planificación de los filtros que se ejecuten para la impresión de un trabajo. El valor de "nice" va desde 0, la mayor prioridad, a 19, la menor prioridad.

El valor predeterminado es '`0`'.

`búsqueda-nombres-máquina` [parámetro de `cups-configuration`]
`host-name-lookups`
Especifica si se realizarán las búsquedas inversas en las conexiones de clientes. La opción `double` instruye a `cupsd` para que verifique que el nombre de máquina al que resuelve la dirección corresponde con la dirección devuelta por dicho nombre de máquina. Las búsquedas dobles también evitan que clientes con direcciones sin registrar se conecten a su servidor. Configure esta opción con `#t` o `double` únicamente si es absolutamente necesario.

El valor predeterminado es '`#f`'

`entero-no-negativo job-kill-delay` [parámetro de `cups-configuration`]
Especifica el número de segundos a esperar antes de terminar los filtros y el motor asociados con un trabajo cancelado o puesto en espera.

El valor predeterminado es '`30`'.

`entero-no-negativo job-retry-interval` [parámetro de `cups-configuration`]
Especifica el intervalo entre los reintentos de trabajos en segundos. Se usa de manera habitual en colas de fax pero también puede usarse con colas de impresión normales cuya política de error sea `retry-job` o `retry-current-job`.

El valor predeterminado es '`30`'.

`entero-no-negativo job-retry-limit` [parámetro de `cups-configuration`]
Especifica el número de reintentos que se llevan a cabo con los trabajos. De manera habitual se usa con colas de fax pero también puede usarse con colas de impresión normal cuya política de error sea `retry-job` o `retry-current-job`.

El valor predeterminado es '`5`'.

`boolean keep-alive?` [parámetro de `cups-configuration`]
Especifica si se permiten conexiones "keep-alive" de HTTP.

El valor predeterminado es '`#t`'

`entero-no-negativo limit-request-body` [parámetro de `cups-configuration`]
Especifica el tamaño máximo de los archivos de impresión, peticiones IPP y datos de formularios HTTP. Un límite de 0 desactiva la comprobación del límite.

El valor predeterminado es '`0`'.

`lista-cadenas-multilínea listen` [parámetro de `cups-configuration`]
Escucha a la espera de conexiones en las interfaces especificadas. Se aceptan valores con la forma *dirección*:*puerto*, donde *dirección* es o bien una dirección IPv6 entre corchetes, una dirección IPv4 o `*` para indicar todas las direcciones. Los valores también pueden ser nombres de archivo de sockets de dominio de UNIX locales. La directiva "Listen" es similar a la directiva "Port", pero le permite la restricción del acceso a interfaces o redes específicas.

`lista-location-access-control` [parámetro de `cups-configuration`]
`location-access-controls`
Especifica un conjunto adicional de controles de acceso.

Los campos disponibles de `location-access-controls` son:

`nombre-archivo path` [parámetro de `location-access-controls`]
Especifica la ruta URI sobre la que el control de acceso tendrá efecto.

`lista-access-control` [parámetro de `location-access-controls`]
`access-controls`
Controles de acceso para todos los accesos a esta ruta, en el mismo formato que `access-controls` de `operation-access-control`.

El valor predeterminado es '`'()`'.

`lista-method-access-control` [parámetro de `location-access-controls`]
`method-access-controls`
Controles de acceso para accesos con métodos específicos para esta ruta.

El valor predeterminado es '`'()`'.

Los campos disponibles de `method-access-controls` son:

`boolean reverse?` [parámetro de `method-access-controls`]
Si es `#t`, los controles de acceso son efectivos con todos los métodos excepto los métodos listados. En otro caso, son efectivos únicamente con los métodos listados.

El valor predeterminado es '`#f`'

`lista-métodos methods` [parámetro de `method-access-controls`]
Métodos con los cuales este control de acceso es efectivo.

El valor predeterminado es '`'()`'.

`lista-control-acceso` [parámetro de `method-access-controls`]
`access-controls`

Directivas de control de acceso, como una lista de cadenas. Cada cadena debe ser una directiva, como '`"Order allow,deny"`'.

El valor predeterminado es '`'()`'.

`entero-no-negativo log-debug-history` [parámetro de `cups-configuration`]

Especifica el número de mensajes de depuración que se retienen para el registro si sucede un error en un trabajo de impresión. Los mensajes de depuración se registran independientemente de la configuración de "LogLevel".

El valor predeterminado es '`100`'.

`nivel-registro log-level` [parámetro de `cups-configuration`]

Especifica el nivel de depuración del archivo "ErrorLog". El valor `none` inhibe todos los registros mientras que `debug2` registra todo.

El valor predeterminado es '`info`'

`formato-tiempo-registro` [parámetro de `cups-configuration`]
`log-time-format`

Especifica el formato de la fecha y el tiempo en los archivos de registro. El valor `standard` registra con segundos completos mientras que `usecs` registra con microsegundos.

El valor predeterminado es '`standard`'.

`entero-no-negativo max-clients` [parámetro de `cups-configuration`]

Especifica el número de clientes simultáneos máximo que son admitidos por el planificador.

El valor predeterminado es '`100`'.

`entero-no-negativo` [parámetro de `cups-configuration`]
`max-clients-per-host`

Especifica el número de clientes simultáneos máximo que se permiten desde una única dirección.

El valor predeterminado es '`100`'.

`entero-no-negativo max-copies` [parámetro de `cups-configuration`]

Especifica el número de copias máximo que una usuaria puede imprimir con cada trabajo.

El valor predeterminado es '`9999`'.

`entero-no-negativo max-hold-time` [parámetro de `cups-configuration`]

Especifica el tiempo máximo que un trabajo puede permanecer en el estado de espera `indefinite` antes de su cancelación. Un valor de 0 desactiva la cancelación de trabajos en espera.

El valor predeterminado es '`0`'.

`entero-no-negativo max-jobs` [parámetro de `cups-configuration`]
Especifica el número de trabajos simultáneos máximo permitido. El valor 0 permite un número ilimitado de trabajos.

El valor predeterminado es '`500`'.

`entero-no-negativo max-jobs-per-printer` [parámetro de `cups-configuration`]
Especifica el número de trabajos simultáneos máximo que se permite por impresora. Un valor de 0 permite hasta `max-jobs` por impresora.

El valor predeterminado es '`0`'.

`entero-no-negativo max-jobs-per-user` [parámetro de `cups-configuration`]
Especifica el número de trabajos simultáneos máximo que se permite por usuaria. Un valor de 0 permite hasta `max-jobs` por usuaria.

El valor predeterminado es '`0`'.

`entero-no-negativo max-job-time` [parámetro de `cups-configuration`]
Especifica el tiempo de duración de la impresión máximo que un trabajo puede tomar antes de ser cancelado, en segundos. El valor 0 desactiva la cancelación de trabajos "atascados".

El valor predeterminado es '`10800`'.

`entero-no-negativo max-log-size` [parámetro de `cups-configuration`]
Especifica el tamaño máximo de los archivos de registro antes de su rotación, en bytes. El valor 0 desactiva la rotación de registros.

El valor predeterminado es '`1048576`'.

`non-negative-integer max-subscriptions` [`cups-configuration` parameter]
Specifies the maximum number of simultaneous event subscriptions that are allowed. Set to '`0`' to allow an unlimited number of subscriptions.

El valor predeterminado es '`0`'.

`non-negative-integer max-subscriptions-per-job` [`cups-configuration` parameter]
Specifies the maximum number of simultaneous event subscriptions that are allowed per job. A value of '`0`' allows up to `max-subscriptions` per job.

El valor predeterminado es '`0`'.

`non-negative-integer max-subscriptions-per-printer` [`cups-configuration` parameter]
Specifies the maximum number of simultaneous event subscriptions that are allowed per printer. A value of '`0`' allows up to `max-subscriptions` per printer.

El valor predeterminado es '`0`'.

`non-negative-integer max-subscriptions-per-user` [`cups-configuration` parameter]
Specifies the maximum number of simultaneous event subscriptions that are allowed per user. A value of '`0`' allows up to `max-subscriptions` per user.

El valor predeterminado es '`0`'.

`entero-no-negativo` [parámetro de `cups-configuration`]
`multiple-operation-timeout`
Especifica el tiempo máximo permitido entre archivos en un trabajo de impresión con múltiples archivos, en segundos.

Defaults to '`900`'.

`variables-entorno` [parámetro de `cups-configuration`]
`environment-variables`
Proporciona la o las variables de entorno especificadas a los procesos iniciados; una lista de cadenas.

El valor predeterminado es '`'()`'.

`lista-policy-configuration policies` [parámetro de `cups-configuration`]
Especifica las políticas de control de acceso con nombre.

Los campos disponibles de `policy-configuration` son:

`string name` [parámetro de `policy-configuration`]
El nombre de la política.

`string job-private-access` [parámetro de `policy-configuration`]
Specifies an access list for a job's private values. `@ACL` maps to the printer's requesting-user-name-allowed or requesting-user-name-denied values. `@OWNER` maps to the job's owner. `@SYSTEM` maps to the groups listed for the `system-group` field of the `files-configuration`, which is reified into the `cups-files.conf(5)` file. Other possible elements of the access list include specific user names, and `@`*`group`* to indicate members of a specific group. The access list may also be simply `all` or `default`.

El valor predeterminado es '`"@OWNER @SYSTEM"`'.

`string job-private-values` [parámetro de `policy-configuration`]
Especifica la lista de valores de trabajos a hacer privados, o bien `all`, `default`, o `none`.

El valor predeterminado es '`"job-name job-originating-host-name job-originating-user-name phone"`'.

`string` [parámetro de `policy-configuration`]
`subscription-private-access`
Specifies an access list for a subscription's private values. `@ACL` maps to the printer's requesting-user-name-allowed or requesting-user-name-denied values. `@OWNER` maps to the job's owner. `@SYSTEM` maps to the groups listed for the `system-group` field of the `files-configuration`, which is reified into the `cups-files.conf(5)` file. Other possible elements of the access list include specific user names, and `@`*`group`* to indicate members of a specific group. The access list may also be simply `all` or `default`.

El valor predeterminado es '`"@OWNER @SYSTEM"`'.

`string` [parámetro de `policy-configuration`]
`subscription-private-values`

Especifica la lista de valores de trabajos a hacer privados, o bien `all`, `default`, o `none`.

El valor predeterminado es '`"notify-events notify-pull-method notify-recipient-uri notify-subscriber-user-name notify-user-data"`'.

`lista-operation-access-control` [parámetro de `policy-configuration`]
`access-controls`

Control de acceso para operaciones de IPP.

El valor predeterminado es '`'()`'.

`boolean-o-entero-no-negativo` [parámetro de `cups-configuration`]
`preserve-job-files`

Especifica si los archivos del trabajo (documentos) se preservan tras la impresión de un trabajo. Si se especifica un valor numérico, los archivos del trabajo se preservan durante el número indicado de segundos tras la impresión. En otro caso, el valor booleano determina la conservación de manera indefinida.

El valor predeterminado es '`86400`'.

`boolean-o-entero-no-negativo` [parámetro de `cups-configuration`]
`preserve-job-history`

Especifica si la historia del trabajo se preserva tras la impresión de un trabajo. Si se especifica un valor numérico, la historia del trabajo se conserva tras la impresión el número de segundos indicado. Si es `#t`, la historia del trabajo se conserva hasta que se alcance el límite de trabajos "MaxJobs".

El valor predeterminado es '`#t`'

`comma-separated-string-list-or-#f` [`cups-configuration` parameter]
`ready-paper-sizes`

Specifies a list of potential paper sizes that are reported as ready, that is: loaded. The actual list will contain only the sizes that each printer supports.

The default value of `#f` is a special case: CUPS will use '`(list \"Letter\" \"Legal\" \"Tabloid\" \"4x6\" \"Env10\")`' if the default paper size is \"Letter\", and '`(list \"A3\" \"A4\" \"A5\" \"A6\" \"EnvDL\")`' otherwise.

`entero-no-negativo reload-timeout` [parámetro de `cups-configuration`]

Especifica el tiempo a esperar hasta la finalización del trabajo antes de reiniciar el planificador.

El valor predeterminado es '`30`'.

`string server-admin` [parámetro de `cups-configuration`]

Especifica la dirección de correo electrónico de la administradora del servidor.

El valor predeterminado es '`"root@localhost.localdomain"`'.

`lista-nombres-máquina-o-*` [parámetro de `cups-configuration`]
`server-alias`

La directiva ServerAlias se usa para la validación de la cabecera HTTP Host cuando los clientes se conecten al planificador desde interfaces externas. El uso del nombre especial `*` puede exponer su sistema a ataques basados en el navegador web de reenlazado DNS ya conocidos, incluso cuando se accede a páginas a través de un cortafuegos. Si el descubrimiento automático de nombres alternativos no funcionase, le recomendamos enumerar cada nombre alternativo con una directiva ServerAlias en vez del uso de `*`.

El valor predeterminado es '`*`'.

`string server-name` [parámetro de `cups-configuration`]

Especifica el nombre de máquina completamente cualificado del servidor.

El valor predeterminado es '`"localhost"`'.

`server-tokens server-tokens` [parámetro de `cups-configuration`]

Especifica qué información se incluye en la cabecera Server de las respuestas HTTP. `None` desactiva la cabecera Server. `ProductOnly` proporciona `CUPS`. `Major` proporciona `CUPS 2`. `Minor` proporciona `CUPS 2.0`. `Minimap` proporciona `CUPS 2.0.0`. `OS` proporciona `CUPS 2.0.0 (`*uname*`)` donde *uname* es la salida de la orden `uname`. `Full` proporciona `CUPS 2.0.0 (`*uname*`) IPP/2.0`.

El valor predeterminado es '`Minimal`'.

`lista-cadenas-multilínea ssl-listen` [parámetro de `cups-configuration`]

Escucha en las interfaces especificadas a la espera de conexiones cifradas. Se aceptan valores con la forma *dirección*:*puerto*, siendo *dirección* o bien una dirección IPv6 entre corchetes, o bien una dirección IPv4, o bien `*` que representa todas las direcciones.

El valor predeterminado es '`'()`'.

`opciones-ssl ssl-options` [parámetro de `cups-configuration`]

Determina las opciones de cifrado. De manera predeterminada, CUPS permite únicamente el cifrado mediante TLS v1.0 o superior mediante el uso de modelos de cifrado de conocida seguridad. La seguridad se reduce cuando se usan opciones `Allow` y se aumenta cuando se usan opciones `Deny`. La opción `AllowRC4` permite el cifrado RC4 de 128 bits, necesario para algunos clientes antiguos que no implementan los modelos más modernos. La opción `AllowSSL3` desactiva SSL v3.0, necesario para algunos clientes antiguos que no implementan TLS v1.0. La opción `DenyCBC` desactiva todos los modelos de cifrado CBC. La opción `DenyTLS1.0` desactiva TLS v1.0—esto fuerza la versión mínima del protocolo a TLS v1.1.

El valor predeterminado es '`'()`'.

`boolean strict-conformance?` [parámetro de `cups-configuration`]

Especifica si el planificador exige que los clientes se adhieran de manera estricta a las especificaciones IPP.

El valor predeterminado es '`#f`'

`entero-no-negativo timeout` [parámetro de `cups-configuration`]

Especifica el plazo de las peticiones HTTP, en segundos.

Defaults to '`900`'.

`boolean web-interface?` [parámetro de `cups-configuration`]
Especifica si se debe activar la interfaz web.

El valor predeterminado es '`#f`'

En este punto probablemente esté pensando, "querido manual de Guix, me gusta todo esto, pero... ¡¿cuando se acaban las opciones de configuración?!". De hecho ya terminan. No obstante, hay un punto más: puede ser que ya tenga un archivo `cupsd.conf` que desee usar. En ese caso, puede proporcionar un objeto `opaque-cups-configuration` como la configuración de `cups-service-type`.

Los campos disponibles de `opaque-cups-configuration` son:

`paquete cups` [parámetro de `opaque-cups-configuration`]
El paquete CUPS.

`string cupsd.conf` [parámetro de `opaque-cups-configuration`]
El contenido de `cupsd.conf`, como una cadena.

`string cups-files.conf` [parámetro de `opaque-cups-configuration`]
El contenido del archivo `cups-files.conf`, como una cadena.

Por ejemplo, si el contenido de sus archivos `cupsd.conf` y `cups-files.conf` estuviese en cadenas del mismo nombre, podría instanciar un servicio CUPS de esta manera:

```
(service cups-service-type
         (opaque-cups-configuration
           (cupsd.conf cupsd.conf)
           (cups-files.conf cups-files.conf)))
```

### 11.10.9 Servicios de escritorio

El módulo `(gnu services desktop)` proporciona servicios que son útiles habitualmente en el contexto de una configuración de "escritorio"—es decir, en una máquina que ejecute un servidor gráfico, posiblemente con interfaces gráficas, etcétera. También define servicios que proporcionan entornos de escritorio específicos como GNOME, Xfce o MATE.

Para simplificar las cosas, el módulo define una variable que contiene el conjunto de servicios que las usuarias esperarían de manera habitual junto a un entorno gráfico y de red:

`%desktop-services` [Variable]
Es una lista de servicios que se construye en base a `%base-services` y añade o ajusta servicios para una configuración de "escritorio" típica.

In particular, it adds a graphical login manager (see Section 11.10.7 [Sistema X Window], page 340), screen lockers, a network management tool (see Section 11.10.5 [Servicios de red], page 314) with modem support (see Section 11.10.5 [Servicios de red], page 314), energy and color management services, the `elogind` login and seat manager, the Polkit privilege service, the GeoClue location service, the AccountsService daemon that allows authorized users change system passwords, a NTP client (see Section 11.10.5 [Servicios de red], page 314) and the Avahi daemon.

La variable `%desktop-services` puede usarse como el campo `services` de una declaración `operating-system` (see Section 11.3 [Referencia de operating-system], page 253).

De manera adicional, los procedimientos `gnome-desktop-service-type`, `xfce-desktop-service`, `mate-desktop-service-type`, `lxqt-desktop-service-type` y `enlightenment-desktop-service-type` pueden añadir GNOME, Xfce, MATE y/o Enlightenment al sistema. "Añadir GNOME" significa que servicios a nivel de sistema como las herramientas de ayuda para el ajuste de la intensidad de luz de la pantalla y de gestión de energía se añaden al sistema, extendiendo `polkit` y `dbus` de manera apropiada, y permitiendo a GNOME operar con privilegios elevados en un número de interfaces del sistema de propósito especial. Además, la adición de un servicio generado por `gnome-desktop-service-type` añade el metapaquete GNOME al perfil del sistema. Del mismo modo, la adición del servicio Xfde no añade únicamente el metapaquete `xfce` al perfil del sistema, sino que también le proporciona al gestor de archivos Thunar la posibilidad de abrir una ventana de gestión de archivos en "modo root", si la usuaria se identifica mediante la contraseña de administración a través de la interfaz gráfica estándar polkit. "Añadir MATE" significa que `polkit` y `dbus` se extienden de manera apropiada, permitiendo a MATE operar con privilegios elevados en un número de interfaces del sistema de propósito especial. De manera adicional, la adición de un servicio de tipo `mate-desktop-service-type` añade el metapaquete MATE al perfil del sistema. "Añadir Enlightenment" significa que `dbus` se extiende de manera apropiada y varios ejecutables de Enlightenment se marcan como "setuid", para permitir el funcionamiento esperado del sistema bloqueo de pantalla de Enlightenment entre otras funcionalidades.

The desktop environments in Guix use the Xorg display server by default. If you'd like to use the newer display server protocol called Wayland, you need to enable Wayland support in GDM (see [wayland-gdm], page 340). Another solution is to use the `sddm-service` instead of GDM as the graphical login manager. You should then select the "GNOME (Wayland)" session in SDDM. Alternatively you can also try starting GNOME on Wayland manually from a TTY with the command "XDG_SESSION_TYPE=wayland exec dbus-run-session gnome-session". Currently only GNOME has support for Wayland.

`gnome-desktop-service-type` [Variable]

Es el tipo del servicio que añade el entorno de escritorio GNOME (`https://www.gnome.org`). Su valor es un objeto `gnome-desktop-configuration` (véase a continuación).

Este servicio añade el paquete `gnome` al perfil del sistema, y extiende polkit con las acciones de `gnome-settings-daemon`.

`gnome-desktop-configuration` [Tipo de datos]

Configuration record for the GNOME desktop environment. Available `gnome-desktop-configuration` fields are:

`core-services` (type: list-of-packages)
A list of packages that the GNOME Shell and applications may rely on.

`shell` (type: list-of-packages)
A list of packages that constitute the GNOME Shell, without applications.

`utilities` (type: list-of-packages)
: A list of packages that serve as applications to use on top of the GNOME Shell.

`gnome` (type: maybe-package)
: This field used to be the only configuration point and specified a GNOME meta-package to install system-wide. Since the meta-package itself provides neither sources nor the actual packages and is only used to propagate them, this field is deprecated.

`extra-packages` (type: list-of-packages)
: A list of GNOME-adjacent packages to also include. This field is intended for users to add their own packages to their GNOME experience. Note, that it already includes some packages that are considered essential by some (most?) GNOME users.

`udev-ignorelist` (default: `()`) (type: list-of-strings)
: A list of regular expressions denoting udev rules or hardware file names provided by any package that should not be installed. By default, every udev rule and hardware file specified by any package referenced in the other fields are installed.

`polkit-ignorelist` (default: `()`) (type: list-of-strings)
: A list of regular expressions denoting polkit rules provided by any package that should not be installed. By default, every polkit rule added by any package referenced in the other fields are installed.

`plasma-desktop-service-type` [Variable]

This is the type of the service that adds the Plasma (`https://kde.org/plasma-desktop/`) desktop environment. Its value is a `plasma-desktop-configuration` object (see below).

This service adds the `plasma` package to the system profile.

`plasma-desktop-configuration` [Data Type]

Configuration record for the Plasma desktop environment.

`plasma` (default: `plasma`)
: The Plasma package to use.

`xfce-desktop-service-type` [Variable]

Este es el tipo de un servicio para ejecutar el entorno de escritorio https://xfce.org (`Xfce`). Su valor es un objeto `xfce-desktop-configuration` (véase a continuación).

Este servicio añade el paquete `xfce` al perfil del sistema, y extiende polkit con la capacidad de `thunar` para manipular el sistema de archivos como root dentro de una sesión de usuaria, tras la identificación de la usuaria con la contraseña de administración.

Note that `xfce4-panel` and its plugin packages should be installed in the same profile to ensure compatibility. When using this service, you should add extra plugins (`xfce4-whiskermenu-plugin`, `xfce4-weather-plugin`, etc.) to the `packages` field of your `operating-system`.

`xfce-desktop-configuration` [Tipo de datos]
Registro de configuración para el entorno de escritorio Xfce.

`xfce` (predeterminado: `xfce`)
El paquete Xfce usado.

`mate-desktop-service-type` [Variable]
Es el tipo del servicio que ejecuta el entorno de escritorio MATE (`https://mate-desktop.org/`). Su valor es un objeto `mate-desktop-configuration` (véase a continuación).

Este servicio añade el paquete `mate` al perfil del sistema, y extiende polkit con acciones de `mate-settings-daemon`.

`mate-desktop-configuration` [Tipo de datos]
Registro de configuración para el entorno de escritorio MATE.

`mate` (predeterminado: `mate`)
El paquete MATE usado.

`lxqt-desktop-service-type` [Variable]
This is the type of the service that runs the LXQt desktop environment (`https://lxqt-project.org`). Its value is a `lxqt-desktop-configuration` object (see below).

Este servicio añade el paquete `lxqt` al perfil del sistema.

`lxqt-desktop-configuration` [Tipo de datos]
Registro de configuración para el entorno de escritorio LXQt.

`lxqt` (predeterminado: `lxqt`)
El paquete LXQT usado.

`sugar-desktop-service-type` [Variable]
This is the type of the service that runs the Sugar desktop environment (`https://www.sugarlabs.org`). Its value is a `sugar-desktop-configuration` object (see below).

This service adds the `sugar` package to the system profile, as well as any selected Sugar activities. By default it only includes a minimal set of activities.

`sugar-desktop-configuration` [Data Type]
Configuration record for the Sugar desktop environment.

`sugar` (default: `sugar`)
The Sugar package to use.

`gobject-introspection` (default: `gobject-introspection`)
The `gobject-introspection` package to use. This package is used to access libraries installed as dependencies of Sugar activities.

`activities` (default: `(list sugar-help-activity)`)
A list of Sugar activities to install.

The following example configures the Sugar desktop environment with a number of useful activities:

```
(use-modules (gnu))
(use-package-modules sugar)
(use-service-modules desktop)
(operating-system
  ...
  (services (cons* (service sugar-desktop-service-type
                            (sugar-desktop-configuration
                              (activities (list sugar-browse-activity
                                                sugar-help-activity
                                                sugar-jukebox-activity
                                                sugar-typing-turtle-activity))))
                   %desktop-services))
  ...)
```

`enlightenment-desktop-service-type` [Variable]

Devuelve un servicio que añade el paquete `enlightenment` al perfil del sistema, y extiende dbus con acciones de `efl`.

`enlightenment-desktop-service-configuration` [Tipo de datos]

`enlightenment` (predeterminado: `enlightenment`)
El paquete enlightenment usado.

Debido a que los servicios de escritorio GNOME, Xfce y MATE incorporan tantos paquetes, la variable `%desktop-services` no incluye ninguno de manera predeterminada. Para añadir GNOME, Xfce o MATE, simplemente use `cons` junto a `%desktop-services` en el campo `services` de su declaración `operating-system`:

```
(use-modules (gnu))
(use-service-modules desktop)
(operating-system
  ...
  ;; cons* añade elementos a la lista proporcionada en el último
  ;; parámetro.
  (services (cons* (service gnome-desktop-service-type)
                   (service xfce-desktop-service)
                   %desktop-services))
  ...)
```

Una vez realizado, estos entornos de escritorio se encontrarán como opciones disponibles en la ventana del gestor gráfico de ingreso al sistema.

Las definiciones de servicio incluidas realmente en `%desktop-services` y proporcionadas por `(gnu services dbus)` y `(gnu services desktop)` se describen a continuación.

`dbus-root-service-type` [Variable]

Type for a service that runs the D-Bus "system bus".[8]

[8] D-Bus (`https://dbus.freedesktop.org/`) is an inter-process communication facility. Its system bus is used to allow system services to communicate and to be notified of system-wide events.

The value for this service type is a `<dbus-configuration>` record.

`dbus-configuration` [Data Type]

Data type representing the configuration for `dbus-root-service-type`.

`dbus` (default: `dbus`) (type: file-like)
Package object for dbus.

`services` (default: `'()`) (type: list)
List of packages that provide an `etc/dbus-1/system.d` directory containing additional D-Bus configuration and policy files. For example, to allow avahi-daemon to use the system bus, *services* must be equal to `(list avahi)`.

`verbose?` (default: `#f`) (type: boolean)
When `#t`, D-Bus is launched with environment variable '`DBUS_VERBOSE`' set to '`1`'. A verbose-enabled D-Bus package such as `dbus-verbose` should be provided to *dbus* in this scenario. The verbose output is logged to `/var/log/dbus-daemon.log`.

## Elogind

Elogind (`https://github.com/elogind/elogind`) is a login and seat management daemon that also handles most system-level power events for a computer, for example suspending the system when a lid is closed, or shutting it down when the power button is pressed.

It also provides a D-Bus interface that can be used to know which users are logged in, know what kind of sessions they have open, suspend the system, inhibit system suspend, reboot the system, and other tasks.

`elogind-service-type` [Variable]

Type of the service that runs `elogind`, a login and seat management daemon. The value for this service is a `<elogind-configuration>` object.

`elogind-configuration` [Data Type]

Data type representing the configuration of `elogind`.

`elogind` (default: `elogind`) (type: file-like)
...

`kill-user-processes?` (default: `#f`) (type: boolean)
...

`kill-only-users` (default: `'()`) (type: list)
...

`kill-exclude-users` (default: `'("root")`) (type: list-of-string)
...

`inhibit-delay-max-seconds` (default: `5`) (type: integer)
...

`handle-power-key` (default: `'poweroff`) (type: symbol)
...

`handle-suspend-key` (default: `'suspend`) (type: symbol)
...

`handle-hibernate-key` (default: `'hibernate`) (type: symbol)
...

`handle-lid-switch` (default: `'suspend`) (type: symbol)
...

`handle-lid-switch-docked` (default: `'ignore`) (type: symbol)
...

`handle-lid-switch-external-power` (default: `*unspecified*`) (type: symbol)
...

`power-key-ignore-inhibited?` (default: `#f`) (type: boolean)
...

`suspend-key-ignore-inhibited?` (default: `#f`) (type: boolean)
...

`hibernate-key-ignore-inhibited?` (default: `#f`) (type: boolean)
...

`lid-switch-ignore-inhibited?` (default: `#t`) (type: boolean)
...

`holdoff-timeout-seconds` (default: `30`) (type: integer)
...

`idle-action` (default: `'ignore`) (type: symbol)
...

`idle-action-seconds` (default: `(* 30 60)`) (type: integer)
...

`runtime-directory-size-percent` (default: `10`) (type: integer)
...

`runtime-directory-size` (default: `#f`) (type: integer)
...

`remove-ipc?` (default: `#t`) (type: boolean)
...

`suspend-state` (default: `'("mem" "standby" "freeze")`) (type: list)
...

`suspend-mode` (default: `'()`) (type: list)
...

`hibernate-state` (default: `'("disk")`) (type: list)
...

`hibernate-mode` (default: `'("platform" "shutdown")`) (type: list)
...

`hybrid-sleep-state` (default: `'("disk")`) (type: list)
...

`hybrid-sleep-mode` (default: `'("suspend" "platform" "shutdown")`) (type: list)
...

`accountsservice-service-type` [Variable]
Type for the service that runs AccountsService, a system service that can list available accounts, change their passwords, and so on. AccountsService integrates with PolicyKit to enable unprivileged users to acquire the capability to modify their system configuration. See AccountsService (`https://www.freedesktop.org/wiki/Software/AccountsService/`) for more information.

The value for this service is a file-like object, by default it is set to `accountsservice` (the package object for AccountsService).

`polkit-service-type` [Variable]
Type for the service that runs the Polkit privilege management service (`https://www.freedesktop.org/wiki/Software/polkit/`), which allows system administrators to grant access to privileged operations in a structured way. By querying the Polkit service, a privileged system component can know when it should grant additional capabilities to ordinary users. For example, an ordinary user can be granted the capability to suspend the system if the user is logged in locally.

The value for this service is a `<polkit-configuration>` object.

`polkit-wheel-service` [Variable]
Servicio que añade a las usuarias del grupo `wheel` como administradoras del servicio Polkit. Esto hace que se solicite su propia contraseña a las usuarias del grupo `wheel` cuando realicen acciones administrativas en vez de la contraseña de `root`, de manera similar al comportamiento de `sudo`.

`upower-service-type` [Variable]
Servicio que ejecuta `https://upower.freedesktop.org/`, `upowerd`, un monitor a nivel de sistema de consumo de energía y niveles de batería, con las opciones de configuración proporcionadas.

Implementa la interfaz D-Bus `org.freedesktop.UPower`, y se usa de forma notable en GNOME.

`upower-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de UPower.

`upower` (predeterminado: *upower*)
Paquete usado para `upower`.

`watts-up-pro?` (predeterminado: `#f`)
Permite el uso del dispositivo Watts Up Pro.

`poll-batteries?` (predeterminado: `#t`)
Usa el servicio de consulta del núcleo para los cambios en niveles de batería.

`ignore-lid?` (predeterminado: `#f`)
: Ignora el estado de la tapa, puede ser útil en caso de ser incorrecto un dispositivo determinado.

`use-percentage-for-policy?` (default: `#t`)
: Whether to use a policy based on battery percentage rather than on estimated time left. A policy based on battery percentage is usually more reliable.

`percentage-low` (default: `20`)
: Cuando `use-percentaje-for-policy?` es `#t`, determina el porcentaje en el que la carga de la batería se considera baja.

`percentage-critical` (default: `5`)
: Cuando `use-percentaje-for-policy?` es `#t`, determina el porcentaje en el que la carga de la batería se considera crítica.

`percentage-action` (predeterminado: `2`)
: Cuando `use-percentaje-for-policy?` es `#t`, determina el porcentaje en el que se tomará la acción.

`time-low` (predeterminado: `1200`)
: Cuando `use-percentaje-for-policy?` es `#t`, determina el tiempo restante en segundos con el que carga de la batería se considera baja.

`time-critical` (predeterminado: `300`)
: Cuando `use-percentaje-for-policy?` es `#t`, determina el tiempo restante en segundos con el que carga de la batería se considera crítica.

`time-action` (predeterminado: `120`)
: Cuando `use-percentaje-for-policy?` es `#t`, determina el tiempo restante en segundos con el que se tomará la acción.

`critical-power-action` (predeterminada: `'hybrid-sleep`)
: La acción tomada cuando se alcanza `percentage-action` o `time-action` (dependiendo de la configuración de `use-percentage-for-policy?`).

  Los valores posibles son:

  - `'power-off`
  - `'hibernate`
  - `'hybrid-sleep`.

`udisks-service-type` [Variable]

Type for the service that runs UDisks (`https://udisks.freedesktop.org/docs/latest/`), a *disk management* daemon that provides user interfaces with notifications and ways to mount/unmount disks. Programs that talk to UDisks include the `udisksctl` command, part of UDisks, and GNOME Disks. Note that Udisks relies on the `mount` command, so it will only be able to use the file-system utilities installed in the system profile. For example if you want to be able to mount NTFS file-systems in read and write fashion, you'll need to have `ntfs-3g` installed system-wide.

The value for this service is a `<udisks-configuration>` object.

`udisks-configuration` [Data Type]
Data type representing the configuration for `udisks-service-type`.

`udisks` (default: `udisks`) (type: file-like)
Package object for UDisks.

`gvfs-service-type` [Variable]
Type for the service that provides virtual file systems for GIO applicaitons, which enables support for `trash:///`, `ftp://`, `sftp://` and many other location schemas in file managers like Nautilus (GNOME Files) and Thunar.

The value for this service is a `<gvfs-configuration>` object.

`gvfs-configuration` [Data Type]
Data type representing the configuration for `gvfs-service-type`.

`gvfs` (default: `gvfs`) (type: file-like)
Package object for GVfs.

`colord-service-type` [Variable]
Devuelve un servicio que ejecuta `colord`, un servicio del sistema con una interfaz D-Bus para la gestión de perfiles de dispositivos de entrada y salida como la pantalla y el escáner. Se usa de forma notable por parte de la herramienta gráfica de "Gestión de color" de GNOME. Véase la página web de colord (`https://www.freedesktop.org/software/colord/`) para más información.

`sane-service-type` [Variable]
This service provides access to scanners *via* SANE (`http://www.sane-project.org`) by installing the necessary udev rules. It is included in `%desktop-services` (see Section 11.10.9 [Servicios de escritorio], page 363) and relies by default on `sane-backends-minimal` package (see below) for hardware support.

`sane-backends-minimal` [Variable]
The default package which the `sane-service-type` installs. It supports many recent scanners.

`sane-backends` [Variable]
This package includes support for all scanners that `sane-backends-minimal` supports, plus older Hewlett-Packard scanners supported by `hplip` package. In order to use this on a system which relies on `%desktop-services`, you may use `modify-services` (see Section 11.19.3 [Referencia de servicios], page 652) as illustrated below:

```
(use-modules (gnu))
(use-service-modules
  ...
  desktop)
(use-package-modules
  ...
  scanner)

(define %my-desktop-services
```

```
;; List of desktop services that supports a broader range of scanners.
(modify-services %desktop-services
  (sane-service-type _ => sane-backends)))

(operating-system
  ...
  (services %my-desktop-services))
```

`geoclue-application-name` [*#:allowed? #t*] [*#:system? #f*] [*#:users '()*] [Procedimiento]

Devuelve una configuración que permite a una aplicación el acceso a los datos de posicionamiento de GeoClue. *nombre* es el Desktop ID de la aplicación, sin la parte `.desktop`. Si el valor de *allowed?* es verdadero, la aplicación tendrá acceso a la información de posicionamiento de manera predeterminada. El valor booleano *system?* indica si una aplicación es un componente de sistema o no. Por último, *users* es una lista de UID de todas las usuarias para las que esta aplicación tiene permitido el acceso de información. Una lista de usuarias vacía significa que se permiten todas las usuarias.

`%standard-geoclue-applications` [Variable]

La lista estándar de configuraciones de GeoClue de aplicaciones conocidas, proporcionando autoridad a la utilidad de fecha y hora de GNOME para obtener la localización actual para ajustar la zona horaria, y permitiendo que los navegadores Icecat y Epiphany puedan solicitar información de localización. Tanto IceCat como Epiphany solicitan permiso a la usuaria antes de permitir a una página web conocer la ubicación de la usuaria.

`geoclue-service-type` [Variable]

Type for the service that runs the GeoClue (`https://wiki.freedesktop.org/www/Software/GeoClue/`) location service. This service provides a D-Bus interface to allow applications to request access to a user's physical location, and optionally to add information to online location databases.

The value for this service is a `<geoclue-configuration>` object.

`bluetooth-service-type` [Variable]

This is the type for the Linux Bluetooth Protocol Stack (`https://bluez.org/`) (BlueZ) system, which generates the `/etc/bluetooth/main.conf` configuration file. The value for this type is a `bluetooth-configuration` record as in this example:

```
(service bluetooth-service-type)
```

See below for details about `bluetooth-configuration`.

`bluetooth-configuration` [Data Type]

Data type representing the configuration for `bluetooth-service`.

`bluez` (default: `bluez`)
: `bluez` package to use.

`name` (default: `"BlueZ"`)
: Default adapter name.

`class` (default: `#x000000`)
: Default device class. Only the major and minor device class bits are considered.

`discoverable-timeout` (default: `180`)
: How long to stay in discoverable mode before going back to non-discoverable. The value is in seconds.

`always-pairable?` (default: `#f`)
: Always allow pairing even if there are no agents registered.

`pairable-timeout` (default: `0`)
: How long to stay in pairable mode before going back to non-discoverable. The value is in seconds.

`device-id` (default: `#f`)
: Use vendor id source (assigner), vendor, product and version information for DID profile support. The values are separated by `":"` and *assigner*, *VID*, *PID* and *version*.

  Los valores posibles son:

  - `#f` to disable it,
  - `"assigner:1234:5678:abcd"`, where *assigner* is either `usb` (default) or `bluetooth`.

`reverse-service-discovery?` (default: `#t`)
: Do reverse service discovery for previously unknown devices that connect to us. For BR/EDR this option is really only needed for qualification since the BITE tester doesn't like us doing reverse SDP for some test cases, for LE this disables the GATT client functionally so it can be used in system which can only operate as peripheral.

`name-resolving?` (default: `#t`)
: Enable name resolving after inquiry. Set it to `#f` if you don't need remote devices name and want shorter discovery cycle.

`debug-keys?` (default: `#f`)
: Enable runtime persistency of debug link keys. Default is false which makes debug link keys valid only for the duration of the connection that they were created for.

`controller-mode` (default: `'dual`)
: Restricts all controllers to the specified transport. `'dual` means both BR/EDR and LE are enabled (if supported by the hardware).

  Los valores posibles son:

  - `'dual`
  - `'bredr`
  - `'le`

`multi-profile` (default: `'off`)
: Enables Multi Profile Specification support. This allows to specify if system supports only Multiple Profiles Single Device (MPSD) configuration

or both Multiple Profiles Single Device (MPSD) and Multiple Profiles Multiple Devices (MPMD) configurations.

Los valores posibles son:

- `'off`
- `'single`
- `'multiple`

`fast-connectable?` (default: `#f`)
: Permanently enables the Fast Connectable setting for adapters that support it. When enabled other devices can connect faster to us, however the tradeoff is increased power consumptions. This feature will fully work only on kernel version 4.1 and newer.

`privacy` (default: `'off`)
: Default privacy settings.

- `'off`: Disable local privacy
- `'network/on`: A device will only accept advertising packets from peer devices that contain private addresses. It may not be compatible with some legacy devices since it requires the use of RPA(s) all the time
- `'device`: A device in device privacy mode is only concerned about the privacy of the device and will accept advertising packets from peer devices that contain their Identity Address as well as ones that contain a private address, even if the peer device has distributed its IRK in the past

and additionally, if *controller-mode* is set to `'dual`:

- `'limited-network`: Apply Limited Discoverable Mode to advertising, which follows the same policy as to BR/EDR that publishes the identity address when discoverable, and Network Privacy Mode for scanning
- `'limited-device`: Apply Limited Discoverable Mode to advertising, which follows the same policy as to BR/EDR that publishes the identity address when discoverable, and Device Privacy Mode for scanning.

`just-works-repairing` (default: `'never`)
: Specify the policy to the JUST-WORKS repairing initiated by peer.

Possible values:

- `'never`
- `'confirm`
- `'always`

`temporary-timeout` (default: `30`)
: How long to keep temporary devices around. The value is in seconds. `0` disables the timer completely.

`refresh-discovery?` (default: `#t`)
: Enables the device to issue an SDP request to update known services when profile is connected.

`experimental` (default: `#f`)
: Enables experimental features and interfaces, alternatively a list of UUIDs can be given.

  Possible values:

  - `#t`
  - `#f`
  - `(list (uuid <uuid-1>) (uuid <uuid-2>) ...)`.

  List of possible UUIDs:

  - `d4992530-b9ec-469f-ab01-6c481c47da1c`: BlueZ Experimental Debug,
  - `671b10b5-42c0-4696-9227-eb28d1b049d6`: BlueZ Experimental Simultaneous Central and Peripheral,
  - `15c0a148-c273-11ea-b3de-0242ac130004`: BlueZ Experimental LL privacy,
  - `330859bc-7506-492d-9370-9a6f0614037f`: BlueZ Experimental Bluetooth Quality Report,
  - `a6695ace-ee7f-4fb9-881a-5fac66c629af`: BlueZ Experimental Offload Codecs.

`remote-name-request-retry-delay` (default: `300`)
: The duration to avoid retrying to resolve a peer's name, if the previous try failed.

`page-scan-type` (default: `#f`)
: BR/EDR Page scan activity type.

`page-scan-interval` (default: `#f`)
: BR/EDR Page scan activity interval.

`page-scan-window` (default: `#f`)
: BR/EDR Page scan activity window.

`inquiry-scan-type` (default: `#f`)
: BR/EDR Inquiry scan activity type.

`inquiry-scan-interval` (default: `#f`)
: BR/EDR Inquiry scan activity interval.

`inquiry-scan-window` (default: `#f`)
: BR/EDR Inquiry scan activity window.

`link-supervision-timeout` (default: `#f`)
: BR/EDR Link supervision timeout.

`page-timeout` (default: `#f`)
: BR/EDR Page timeout.

`min-sniff-interval` (default: `#f`)
: BR/EDR minimum sniff interval.

`max-sniff-interval` (default: `#f`)
: BR/EDR maximum sniff interval.

`min-advertisement-interval` (default: `#f`)
: LE minimum advertisement interval (used for legacy advertisement only).

`max-advertisement-interval` (default: `#f`)
: LE maximum advertisement interval (used for legacy advertisement only).

`multi-advertisement-rotation-interval` (default: `#f`)
: LE multiple advertisement rotation interval.

`scan-interval-auto-connect` (default: `#f`)
: LE scanning interval used for passive scanning supporting auto connect.

`scan-window-auto-connect` (default: `#f`)
: LE scanning window used for passive scanning supporting auto connect.

`scan-interval-suspend` (default: `#f`)
: LE scanning interval used for active scanning supporting wake from suspend.

`scan-window-suspend` (default: `#f`)
: LE scanning window used for active scanning supporting wake from suspend.

`scan-interval-discovery` (default: `#f`)
: LE scanning interval used for active scanning supporting discovery.

`scan-window-discovery` (default: `#f`)
: LE scanning window used for active scanning supporting discovery.

`scan-interval-adv-monitor` (default: `#f`)
: LE scanning interval used for passive scanning supporting the advertisement monitor APIs.

`scan-window-adv-monitor` (default: `#f`)
: LE scanning window used for passive scanning supporting the advertisement monitor APIs.

`scan-interval-connect` (default: `#f`)
: LE scanning interval used for connection establishment.

`scan-window-connect` (default: `#f`)
: LE scanning window used for connection establishment.

`min-connection-interval` (default: `#f`)
: LE default minimum connection interval. This value is superseded by any specific value provided via the Load Connection Parameters interface.

`max-connection-interval` (default: `#f`)
: LE default maximum connection interval. This value is superseded by any specific value provided via the Load Connection Parameters interface.

`connection-latency` (default: `#f`)
: LE default connection latency. This value is superseded by any specific value provided via the Load Connection Parameters interface.

`connection-supervision-timeout` (default: `#f`)
: LE default connection supervision timeout. This value is superseded by any specific value provided via the Load Connection Parameters interface.

`autoconnect-timeout` (default: `#f`)
: LE default autoconnect timeout. This value is superseded by any specific value provided via the Load Connection Parameters interface.

`adv-mon-allowlist-scan-duration` (default: `300`)
: Allowlist scan duration during interleaving scan. Only used when scanning for ADV monitors. The units are msec.

`adv-mon-no-filter-scan-duration` (default: `500`)
: No filter scan duration during interleaving scan. Only used when scanning for ADV monitors. The units are msec.

`enable-adv-mon-interleave-scan?` (default: `#t`)
: Enable/Disable Advertisement Monitor interleave scan for power saving.

`cache` (default: `'always`)
: GATT attribute cache.

  Los valores posibles son:

  - `'always`: Always cache attributes even for devices not paired, this is recommended as it is best for interoperability, with more consistent reconnection times and enables proper tracking of notifications for all devices
  - `'yes`: Only cache attributes of paired devices
  - `'no`: Never cache attributes.

`key-size` (default: `0`)
: Minimum required Encryption Key Size for accessing secured characteristics.

  Los valores posibles son:

  - `0`: Don't care
  - `7 <= N <= 16`

`exchange-mtu` (default: `517`)
: Exchange MTU size. Possible values are:

  - `23 <= N <= 517`

`att-channels` (default: `3`)
: Number of ATT channels. Possible values are:

  - `1`: Disables EATT
  - `2 <= N <= 5`

`session-mode` (default: `'basic`)
AVDTP L2CAP signalling channel mode.

Los valores posibles son:

- `'basic`: Use L2CAP basic mode
- `'ertm`: Use L2CAP enhanced retransmission mode.

`stream-mode` (default: `'basic`)
AVDTP L2CAP transport channel mode.

Los valores posibles son:

- `'basic`: Use L2CAP basic mode
- `'streaming`: Use L2CAP streaming mode.

`reconnect-uuids` (default: `'()`)
The ReconnectUUIDs defines the set of remote services that should try to be reconnected to in case of a link loss (link supervision timeout). The policy plugin should contain a sane set of values by default, but this list can be overridden here. By setting the list to empty the reconnection feature gets disabled.

Possible values:

- `'()`
- `(list (uuid <uuid-1>) (uuid <uuid-2>) ...)`.

`reconnect-attempts` (default: `7`)
Defines the number of attempts to reconnect after a link lost. Setting the value to 0 disables reconnecting feature.

`reconnect-intervals` (default: `'(1 2 4 8 16 32 64)`)
Defines a list of intervals in seconds to use in between attempts. If the number of attempts defined in *reconnect-attempts* is bigger than the list of intervals the last interval is repeated until the last attempt.

`auto-enable?` (default: `#f`)
Defines option to enable all controllers when they are found. This includes adapters present on start as well as adapters that are plugged in later on.

`resume-delay` (default: `2`)
Audio devices that were disconnected due to suspend will be reconnected on resume. *resume-delay* determines the delay between when the controller resumes from suspend and a connection attempt is made. A longer delay is better for better co-existence with Wi-Fi. The value is in seconds.

`rssi-sampling-period` (default: `#xFF`)
Default RSSI Sampling Period. This is used when a client registers an advertisement monitor and leaves the RSSISamplingPeriod unset.

Los valores posibles son:

- `#x0`: Report all advertisements
- `N = #xXX`: Report advertisements every N x 100 msec (range: #x01 to #xFE)

- `#xFF`: Report only one advertisement per device during monitoring period.

`gnome-keyring-service-type` [Variable]

Es el tipo del servicio que añade el entorno de escritorio el anillo de claves de GNOME (`https://wiki.gnome.org/Projects/GnomeKeyring`). Su valor es un objeto `gnome-keyring-configuration` (véase a continuación).

Este servicio añade el paquete `gnome-keyring` al perfil del sistema y extiende PAM con entradas que usan `pam_gnome_keyring.so`, las cuales desbloquean el anillo de claves del sistema de la usuaria cuando ingrese en el sistema o cambie su contraseña con passwd.

`gnome-keyring-configuration` [Tipo de datos]

Registro de configuración para el servicio del anillo de claves de GNOME.

`keyring` (predeterminado: `gnome-keyring`)
: El paquete GNOME keyring usado.

`pam-services`
: Una lista de pares `(`*`servicio . tipo`*`)` que denotan los servicios de PAM que deben extenderse, donde *servicio* es el nombre de un servicio existente que debe extenderse y *tipo* es `login` o `passwd`.

  Si se proporciona `login`, añade un campo opcional `pam_gnome_keyring.so` al bloque de identificación sin parámetros y al bloque de sesión con `auto_start`. Si se proporciona `passwd`, añade un campo opcional `pam_gnome_keyring.so` al bloque de contraseña sin parámetros.

  De manera predeterminada, este campo contiene "gdm-password" con el valor `login` y "passwd" tiene valor `passwd`.

`seatd-service-type` [Variable]

seatd (`https://sr.ht/~kennylevinsen/seatd/`) is a minimal seat management daemon.

Seat management takes care of mediating access to shared devices (graphics, input), without requiring the applications needing access to be root.

```
(append
  (list
   ;; make sure seatd is running
   (service seatd-service-type))

  ;; normally one would want %base-services
  %base-services)
```

`seatd` operates over a UNIX domain socket, with `libseat` providing the client side of the protocol. Applications that acquire access to the shared resources via `seatd` (e.g. `sway`) need to be able to talk to this socket. This can be achieved by adding the user they run under to the group owning `seatd`'s socket (usually "seat"), like so:

```
(user-account
```

```
(name "alice")
(group "users")
(supplementary-groups '("wheel"   ; allow use of sudo, etc.
                        "seat"    ; seat management
                        "audio"   ; sound card
                        "video"   ; video devices such as webcams
                        "cdrom")) ; the good ol' CD-ROM
(comment "Bob's sister"))
```

Depending on your setup, you will have to not only add regular users, but also system users to this group. For instance, some greetd greeters require graphics and therefore also need to negotiate with seatd.

`seatd-configuration` [Data Type]

Configuration record for the seatd daemon service.

`seatd` (default: `seatd`)
: The seatd package to use.

`group` (default: '`"seat"`')
: Group to own the seatd socket.

`socket` (default: '`"/run/seatd.sock"`')
: Where to create the seatd socket.

`logfile` (default: '`"/var/log/seatd.log"`')
: Log file to write to.

`loglevel` (default: '`"error"`')
: Log level to output logs. Possible values: '`"silent"`', '`"error"`', '`"info"`' and '`"debug"`'.

### 11.10.10 Servicios de sonido

El módulo `(gnu services sound)` proporciona un servicio para la configuración del sistema ALSA (arquitectura avanzada de sonido de Linux), el cual establece PulseAudio como el controlador de ALSA preferido para salida de sonido.

`alsa-service-type` [Variable]

Es el tipo para el sistema ALSA (`https://alsa-project.org/`) (Arquitectura de sonido avanzada de Linux), que genera el archivo de configuración `/etc/asound.conf`. El valor para este tipo es un registro `alsa-configuration` como en el ejemplo:

```
(service alsa-service-type)
```

Véase a continuación más detalles sobre `alsa-configuration`.

`alsa-configuration` [Tipo de datos]

Tipo de datos que representa la configuración para `alsa-service`.

`alsa-plugins` (predeterminados: *alsa-plugins*)
: El paquete `alsa-plugins` usado.

`pulseaudio?` (predeterminado: *#t*)
: Determina si las aplicaciones ALSA deben usar el servidor de sonido PulseAudio (`https://www.pulseaudio.org/`) de manera transparente.

El uso de PulseAudio le permite la ejecución de varias aplicaciones que produzcan sonido al mismo tiempo y su control individual mediante `pavucontrol`, entre otras opciones.

`extra-options` (predeterminado: `""`)
Cadena a añadir al final del archivo `/etc/asound.conf`.

Las usuarias individuales que deseen forzar la configuración de ALSA en el sistema para sus cuentas pueden hacerlo con el archivo `~/.asoundrc`:

```
# En guix tenemos que especificar la ruta absoluta del módulo.
pcm_type.jack {
  lib "/home/alicia/.guix-profile/lib/alsa-lib/libasound_module_pcm_jack.so"
}

# Redirección de ALSA a jack:
# <http://jackaudio.org/faq/routing_alsa.html>.
pcm.rawjack {
  type jack
  playback_ports {
    0 system:playback_1
    1 system:playback_2
  }

  capture_ports {
    0 system:capture_1
    1 system:capture_2
  }
}

pcm.!default {
  type plug
  slave {
    pcm "rawjack"
  }
}
```

Véase `https://www.alsa-project.org/main/index.php/Asoundrc` para obtener más detalles.

`pulseaudio-service-type` [Variable]
Tipo de servicio del servidor de sonido PulseAudio. Existe para permitir los cambios a nivel de sistema de la configuración predeterminada a través de `pulseaudio-configuration`, véase a continuación.

**Aviso:** Este servicio hace que se ignoren los archivos de configuración de cada usuaria. Si desea que PulseAudio respete los archivos de configuración en `~/.config/pulse` tiene que eliminar del entorno (con `unset`) las variables `PULSE_CONFIG` y `PULSE_CLIENTCONFIG` en su archivo `~/.bash_profile`.

> **Aviso:** Este servicio no asegura en sí que el paquete `pulseaudio` exista en su máquina. Únicamente añade los archivos de configuración, como se detalla a continuación. En el caso (ciertamente poco probable), de que se encuentre si un paquete pulseaudio `pulseaudio`, considere activarlo a través del tipo `alsa-service-type` mostrado previamente.

`pulseaudio-configuration` [Tipo de datos]

Tipo de datos que representa la configuración para `pulseaudio-service`.

`client-conf` (predeterminada: `'()`)
: List of settings to set in `client.conf`. Accepts a list of strings or symbol-value pairs. A string will be inserted as-is with a newline added. A pair will be formatted as "key = value", again with a newline added.

`daemon-conf` (predeterminada: `'((flat-volumes . no))`)
: Lista de opciones de configuración de `daemon.conf`, con el mismo formato que *client-conf*.

`script-file` (predeterminado: `(file-append pulseaudio "/etc/pulse/default.pa")`)
: Script file to use as `default.pa`. In case the `extra-script-files` field below is used, an `.include` directive pointing to `/etc/pulse/default.pa.d` is appended to the provided script.

`extra-script-files` (default: `'()`)
: A list of file-like objects defining extra PulseAudio scripts to run at the initialization of the `pulseaudio` daemon, after the main `script-file`. The scripts are deployed to the `/etc/pulse/default.pa.d` directory; they should have the '`.pa`' file name extension. For a reference of the available commands, refer to `man pulse-cli-syntax`.

`system-script-file` (predeterminado: `(file-append pulseaudio "/etc/pulse/system.pa")`)
: Archivo del guión usado como `system.pa`

The example below sets the default PulseAudio card profile, the default sink and the default source to use for a old SoundBlaster Audigy sound card:

```
(pulseaudio-configuration
 (extra-script-files
  (list (plain-file "audigy.pa"
                    (string-append "\
set-card-profile alsa_card.pci-0000_01_01.0 \
  output:analog-surround-40+input:analog-mono
set-default-source alsa_input.pci-0000_01_01.0.analog-mono
set-default-sink alsa_output.pci-0000_01_01.0.analog-surround-40\n")))))
```

Note that `pulseaudio-service-type` is part of `%desktop-services`; if your operating system declaration was derived from one of the desktop templates, you'll want to adjust the above example to modify the existing `pulseaudio-service-type` via `modify-services` (see Section 11.19.3 [Referencia de servicios], page 652), instead of defining a new one.

`ladspa-service-type` [Variable]

Este servicio proporciona valor a la variable *LADSPA_PATH*, de manera que los programas que lo tengan en cuenta, por ejemplo PulseAudio, puedan cargar módulos LADSPA.

El siguiente ejemplo configura el servicio para permitir la activación de los módulos del paquete `swh-plugins`:

```
(service ladspa-service-type
         (ladspa-configuration (plugins (list swh-plugins))))
```

Véase `http://plugin.org.uk/ladspa-swh/docs/ladspa-swh.html` para obtener más detalles.

### 11.10.11 File Search Services

The services in this section populate *file databases* that let you search for files on your machine. These services are provided by the `(gnu services admin)` module.

The first one, `file-database-service-type`, periodically runs the venerable `updatedb` command (see Section "Invoking updatedb" in *GNU Findutils*). That command populates a database of file names that you can then search with the `locate` command (see Section "Invoing locate" in *GNU Findutils*), as in this example:

```
locate important-notes.txt
```

You can enable this service with its default settings by adding this snippet to your operating system services:

```
(service file-database-service-type)
```

This updates the database once a week, excluding files from `/gnu/store`—these are more usefully handled by `guix locate` (see Section 5.5 [Invoking guix locate], page 52). You can of course provide a custom configuration, as described below.

`file-database-service-type` [Variable]

This is the type of the file database service, which runs `updatedb` periodically. Its associated value must be a `file-database-configuration` record, as described below.

`file-database-configuration` [Data Type]

Record type for the `file-database-service-type` configuration, with the following fields:

`package` (default: `findutils`)
: The GNU Findutils package from which the `updatedb` command is taken.

`schedule` (default: `%default-file-database-update-schedule`)
: String or G-exp denoting an mcron schedule for the periodic `updatedb` job (see Section "Guile Syntax" in *GNU mcron*).

`excluded-directories` (default `%default-file-database-excluded-directories`)
: List of regular expressions of directories to ignore when building the file database. By default, this includes `/tmp` and `/gnu/store`; the latter should instead be indexed by `guix locate` (see Section 5.5 [Invoking guix

locate], page 52). This list is passed to the `--prunepaths` option of `updatedb` (see Section "Invoking updatedb" in *GNU Findutils*).

The second service, `package-database-service-type`, builds the database used by `guix locate`, which lets you search for packages that contain a given file (see Section 5.5 [Invoking guix locate], page 52). The service periodically updates a system-wide database, which will be readily available to anyone running `guix locate` on the system. To use this service with its default settings, add this snippet to your service list:

```
(service package-database-service-type)
```

This will run `guix locate --update` once a week.

`package-database-service-type` [Variable]

This is the service type for periodic `guix locate` updates (see Section 5.5 [Invoking guix locate], page 52). Its value must be a `package-database-configuration` record, as shown below.

`package-database-configuration` [Data Type]

Data type to configure periodic package database updates. It has the following fields:

`package` (default: `guix`)
: El paquete Guix usado.

`schedule` (default: `%default-package-database-update-schedule`)
: String or G-exp denoting an mcron schedule for the periodic `guix locate --update` job (see Section "Guile Syntax" in *GNU mcron*).

`method` (default: `'store`)
: Indexing method for `guix locate`. The default value, `'store`, yields a more complete database but is relatively expensive in terms of CPU and input/output.

`channels` (predeterminada: `#~%default-channels`)
: G-exp denoting the channels to use when updating the database (see Chapter 6 [Canales], page 69).

### 11.10.12 Servicios de bases de datos

El módulo `(gnu services databases)` proporciona los siguientes servicios.

## PostgreSQL

`postgresql-service-type` [Variable]

The service type for the PostgreSQL database server. Its value should be a valid `postgresql-configuration` object, documented below. The following example describes a PostgreSQL service with the default configuration.

```
(service postgresql-service-type
         (postgresql-configuration
          (postgresql postgresql-10)))
```

Si los servicios fallan al arrancar puede deberse a que ya se encuentra presente otro cluster incompatible en *data-directory*. Puede modificar el valor (o, si no necesita más dicho cluster, borrar *data-directory*), y reiniciar el servicio.

La identificación de pares se usa de manera predeterminada y la cuenta de usuaria `postgres` no tiene intérprete predeterminado, lo que evita la ejecución directa de órdenes `psql` bajo dicha cuenta. Para usar `psql`, puede ingresar temporalmente al sistema como `postgres` usando un intérprete de órdenes, crear una cuenta de administración de PostgreSQL con el mismo nombre de una de las usuarias del sistema y, tras esto, crear la base de datos asociada.

```
sudo -u postgres -s /bin/sh
createuser --interactive
createdb $MI_CUENTA_DE_USUARIA  # Sustituir por el valor apropiado.
```

`postgresql-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `postgresql-service-type`.

`postgresql` (default: `postgresql-10`)
: El paquete PostgreSQL usado para este servicio.

`port` (predeterminado: `5432`)
: Puerto en el que debe escuchar PostgreSQL.

`locale` (predeterminado: `"en_US.utf8"`)
: Localización predeterminada cuando se crea el cluster de la base de datos.

`config-file` (predeterminado: `(postgresql-config-file)`)
: The configuration file to use when running PostgreSQL. The default behaviour uses the postgresql-config-file record with the default values for the fields.

`log-directory` (default: `"/var/log/postgresql"`)
: The directory where `pg_ctl` output will be written in a file named `"pg_ctl.log"`. This file can be useful to debug PostgreSQL configuration errors for instance.

`data-directory` (predeterminado: `"/var/lib/postgresql/data"`)
: Directorio en el que se almacenan los datos.

`extension-packages` (predeterminado: `'()`)
: Las extensiones adicionales se cargan de paquetes enumerados en *extension-packages*. Las extensiones están disponibles en tiempo de ejecución. Por ejemplo, para crear una base de datos geográfica con la extensión `postgis`, una usuaria podría configurar el servicio postgresql-service como en este ejemplo:

```
(use-package-modules databases geo)

(operating-system
  ...
  ;; postgresql es necesario para ejecutar `psql' pero no se necesita
  ;; postgis para un funcionamiento correcto.
  (packages (cons* postgresql %base-packages))
  (services
    (cons*
      (service postgresql-service-type
```

```
                          (postgresql-configuration
                           (postgresql postgresql-10)
                           (extension-packages (list postgis))))
                    %base-services)))
```

Una vez hecho, la extensión estará visible y podrá inicializar una base de datos geográfica de este modo:

```
psql -U postgres
> create database pruebapostgis;
> \connect pruebapostgis;
> create extension postgis;
> create extension postgis_topology;
```

No es necesaria la adición de este campo para extensiones incluidas en la distribución oficial[9] como hstore o dblink, puesto que ya pueden cargarse en postgresql. Este campo únicamente es necesario para extensiones proporcionadas por otros paquetes.

`create-account?` (default: `#t`)
: Whether or not the `postgres` user and group should be created.

`uid` (predeterminado: `#f`)
: Explicitly specify the UID of the `postgres` daemon account. You normally do not need to specify this, in which case a free UID will be automatically assigned.

  One situation where this option might be useful is if the *data-directory* is located on a mounted network share.

`gid` (default: `#f`)
: Explicitly specify the GID of the `postgres` group.

`postgresql-config-file` [Tipo de datos]

Data type representing the PostgreSQL configuration file. As shown in the following example, this can be used to customize the configuration of PostgreSQL. Note that you can use any G-expression or filename in place of this record, if you already have a configuration file you'd like to use for example.

```
(service postgresql-service-type
         (postgresql-configuration
          (config-file
           (postgresql-config-file
            (log-destination "stderr")
            (hba-file
             (plain-file "pg_hba.conf"
                         "
local all all trust
host all all 127.0.0.1/32  md5
host all all ::1/128  md5"))
            (extra-config
```

[9] NdT: "contrib" de "contributed" en inglés, "contribuciones" podría entenderse en castellano.

```
                    '(("session_preload_libraries"     "auto_explain")
                      ("random_page_cost"              2)
                      ("auto_explain.log_min_duration" "100 ms")
                      ("work_mem"                      "500 MB")
                      ("logging_collector"             #t)
                      ("log_directory"                 "/var/log/postgresql")))))))
```

`log-destination` (predeterminado: `"syslog"`)
: The logging method to use for PostgreSQL. Multiple values are accepted, separated by commas.

`hba-file` (predeterminado: `%default-postgres-hba`)
: Nombre de archivo o expresión-G para la configuración de identificación basada en nombres de máquina.

`ident-file` (predeterminado: `%default-postgres-ident`)
: Nombre de archivo o expresión-G para la configuración de asociación de nombres de usuaria.

`socket-directory` (default: `"/var/run/postgresql"`)
: Specifies the directory of the Unix-domain socket(s) on which PostgreSQL is to listen for connections from client applications. If set to `""` PostgreSQL does not listen on any Unix-domain sockets, in which case only TCP/IP sockets can be used to connect to the server.

  By default, the `#false` value means the PostgreSQL default value will be used, which is currently '`/tmp`'.

`extra-config` (predeterminada: `'()`)
: Claves y valores adicionales que se incluirán en el archivo de configuración de PostgreSQL. Cada entrada en la lista debe ser una lista cuyo primer elemento sea la clave y los elementos restantes son los valores.

  The values can be numbers, booleans or strings and will be mapped to PostgreSQL parameters types `Boolean`, `String`, `Numeric`, `Numeric with Unit` and `Enumerated` described here (`https://www.postgresql.org/docs/current/config-setting.html`).

`postgresql-role-service-type` [Variable]

This service allows to create PostgreSQL roles and databases after PostgreSQL service start. Here is an example of its use.

```
(service postgresql-role-service-type
            (postgresql-role-configuration
             (roles
              (list (postgresql-role
                     (name "test")
                     (create-database? #t))))))
```

This service can be extended with extra roles, as in this example:

```
(service-extension postgresql-role-service-type
                   (const (postgresql-role
                           (name "alice")
                           (create-database? #t))))
```

`postgresql-role` [Data Type]

PostgreSQL manages database access permissions using the concept of roles. A role can be thought of as either a database user, or a group of database users, depending on how the role is set up. Roles can own database objects (for example, tables) and can assign privileges on those objects to other roles to control who has access to which objects.

`name` The role name.

`permissions` (default: `'(createdb login)`)
The role permissions list. Supported permissions are `bypassrls`, `createdb`, `createrole`, `login`, `replication` and `superuser`.

`create-database?` (default: `#f`)
whether to create a database with the same name as the role.

`encoding` (default: `"UTF8"`)
The character set to use for storing text in the database.

`collation` (default: `"en_US.utf8"`)
The string sort order locale setting.

`ctype` (default: `"en_US.utf8"`)
The character classification locale setting.

`template` (default: `"template1"`)
The default template to copy the new database from when creating it. Use `"template0"` for a pristine database with no system-local modifications.

`postgresql-role-configuration` [Data Type]

Data type representing the configuration of *postgresql-role-service-type*.

`host` (default: `"/var/run/postgresql"`)
The PostgreSQL host to connect to.

`log` (default: `"/var/log/postgresql_roles.log"`)
File name of the log file.

`roles` (default: `'()`)
The initial PostgreSQL roles to create.

## MariaDB/MySQL

`mysql-service-type` [Variable]

This is the service type for a MySQL or MariaDB database server. Its value is a `mysql-configuration` object that specifies which package to use, as well as various settings for the `mysqld` daemon.

`mysql-configuration` [Tipo de datos]

Data type representing the configuration of *mysql-service-type*.

`mysql` (predeterminado: *mariadb*)
Objeto de paquete del servidor de bases de datos MySQL, puede ser tanto *mariadb* como *mysql*.

Para MySQL, se mostrará una contraseña de root temporal durante el tiempo de activación. Para MariaDB, la contraseña de root está vacía.

`bind-address` (predeterminada: `"127.0.0.1"`)
The IP on which to listen for network connections. Use `"0.0.0.0"` to bind to all available network interfaces.

`port` (predeterminado: `3306`)
Puerto TCP en el que escucha el servidor de bases de datos a la espera de conexiones entrantes.

`socket` (default: `"/run/mysqld/mysqld.sock"`)
Socket file to use for local (non-network) connections.

`extra-content` (predeterminado: `""`)
Additional settings for the `my.cnf` configuration file.

`extra-environment` (default: `#~'()`)
List of environment variables passed to the `mysqld` process.

`auto-upgrade?` (default: `#t`)
Whether to automatically run `mysql_upgrade` after starting the service. This is necessary to upgrade the *system schema* after "major" updates (such as switching from MariaDB 10.4 to 10.5), but can be disabled if you would rather do that manually.

## Memcached

`memcached-service-type` [Variable]
Este es el tipo de servicio para el servicio Memcached (`https://memcached.org/`), que proporciona caché distribuida en memoria. El valor para este tipo de servicio es un objeto `memcached-configuration`.

```
(service memcached-service-type)
```

`memcached-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de memcached.

`memcached` (predeterminado: `memcached`)
El paquete de Memcached usado.

`interfaces` (predeterminadas: `'("0.0.0.0")`)
Interfaces de red por las que se esperan conexiones.

`tcp-port` (predeterminado: `11211`)
Port on which to accept connections.

`udp-port` (predeterminado: `11211`)
Puerto en el que se deben aceptar conexiones UDP, el valor 0 desactiva la escucha en un socket UDP.

`additional-options` (predeterminadas: `'()`)
Opciones de línea de órdenes adicionales que se le proporcionarán a `memcached`.

### Redis

`redis-service-type` [Variable]

Es el tipo de servicio para el almacén de clave/valor Redis (`https://redis.io/`), cuyo valor es un objeto `redis-configuration`.

`redis-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de redis.

`redis` (predeterminado: `redis`)
: El paquete Redis usado.

`bind` (predeterminada: `"127.0.0.1"`)
: La interfaz de red en la que se escucha.

`port` (predeterminado: `6379`)
: Puerto en el que se aceptan conexiones, el valor 0 desactiva la escucha en un socket TCP.

`working-directory` (predeterminado: `"/var/lib/redis"`)
: Directorio en el que se almacena los archivos de base de datos y relacionados.

## 11.10.13 Servicios de correo

El módulo `(gnu services mail)` proporciona definiciones de servicios Guix para servicios de correo electrónico: servidores IMAP, POP3 y LMTP, así como agentes de transporte de correo (MTA). ¡Muchos acrónimos! Estos servicios se detallan en las subsecciones a continuación.

### Servicio Dovecot

`dovecot-service-type` [Variable]

Type for the service that runs the Dovecot IMAP/POP3/LMTP mail server, whose value is a `<dovecot-configuration>` object.

Habitualmente Dovecot no necesita mucha configuración; el objeto de configuración predeterminado creado por `(dovecot-configuration)` es suficiente si su correo se entrega en `~/Maildir`. Un certificado auto-firmado se generará para las conexiones protegidas por TLS, aunque Dovecot también escuchará en puertos sin cifrar de manera predeterminada. Existe un amplio número de opciones no obstante, las cuales las administradoras del correo puede que deban cambiar, y como en el caso de otros servicios, Guix permite a la administradora del sistema la especificación de dichos parámetros a través de una interfaz Scheme uniforme.

Por ejemplo, para especificar que el correo se encuentra en `maildir:~/.correo`, se debe instanciar el servicio de Dovecot de esta manera:

```
(service dovecot-service-type
         (dovecot-configuration
           (mail-location "maildir:~/.mail")))
```

A continuación se encuentran los parámetros de configuración disponibles. El tipo de cada parámetro antecede la definición del mismo; por ejemplo, '`string-list foo`' indica que

el parámetro `foo` debe especificarse como una lista de cadenas. También existe la posibilidad de especificar la configuración como una cadena, si tiene un archivo `dovecot.conf` antiguo que quiere trasladar a otro sistema; véase el final para más detalles.

Los campos disponibles de `dovecot-configuration` son:

`package dovecot` [parámetro de `dovecot-configuration`]
El paquete dovecot.

`lista-cadenas-separada-comas listen` [parámetro de `dovecot-configuration`]
Una lista de direcciones IP o nombres de máquina donde se escucharán conexiones. '`*`' escucha en todas las interfaces IPv4, '`::`' escucha en todas las interfaces IPv6. Si desea especificar puertos distintos a los predefinidos o algo más complejo, personalice los campos de dirección y puerto del '`inet-listener`' de los servicios específicos en los que tenga interés.

`lista-protocol-configuration protocols` [parámetro de `dovecot-configuration`]
Lista de protocolos que se desea ofrecer. Los protocolos disponibles incluyen '`imap`', '`pop3`' y '`lmtp`'.

Los campos disponibles de `protocol-configuration` son:

`string name` [parámetro de `protocol-configuration`]
El nombre del protocolo.

`string auth-socket-path` [parámetro de `protocol-configuration`]
Ruta del socket de UNIX del servidor de identificación maestro para la búsqueda de usuarias. Se usa por parte de imap (para usuarias compartidas) y lda. Su valor predeterminado es '`"/var/run/dovecot/auth-userdb"`'.

`boolean imap-metadata?` [`protocol-configuration` parameter]
Whether to enable the `IMAP METADATA` extension as defined in RFC 5464 (`https://tools.ietf.org/html/rfc5464`), which provides a means for clients to set and retrieve per-mailbox, per-user metadata and annotations over IMAP.

If this is '`#t`', you must also specify a dictionary *via* the `mail-attribute-dict` setting.

El valor predeterminado es '`#f`'

`space-separated-string-list managesieve-notify-capabilities` [`protocol-configuration` parameter]
Which NOTIFY capabilities to report to clients that first connect to the ManageSieve service, before authentication. These may differ from the capabilities offered to authenticated users. If this field is left empty, report what the Sieve interpreter supports by default.

El valor predeterminado es '`'()`'.

`space-separated-string-list` [`protocol-configuration` parameter]
`managesieve-sieve-capability`

Which SIEVE capabilities to report to clients that first connect to the ManageSieve service, before authentication. These may differ from the capabilities offered to authenticated users. If this field is left empty, report what the Sieve interpreter supports by default.

El valor predeterminado es ‘`'()`’.

`list-cadenas-separada-espacios` [parámetro de `protocol-configuration`]
`mail-plugins`

Lista separada por espacios de módulos a cargar.

`entero-no-negativo` [parámetro de `protocol-configuration`]
`mail-max-userip-connections`

Número máximo de conexiones IMAP permitido para una usuaria desde cada dirección IP. ATENCIÓN: El nombre de usuaria es sensible a las mayúsculas. Su valor predeterminado es ‘`10`’.

`lista-service-configuration` [parámetro de `dovecot-configuration`]
`services`

Lista de servicios activados. Los servicios disponibles incluyen ‘`imap`’, ‘`imap-login`’, ‘`pop3`’, ‘`pop3-login`’, ‘`auth`’ y ‘`lmtp`’.

Los campos disponibles de `service-configuration` son:

`string kind` [parámetro de `service-configuration`]

El tipo del servicio. Entre los valores aceptados se incluye `director`, `imap-login`, `pop3-login`, `lmtp`, `imap`, `pop3`, `auth`, `auth-worker`, `dict`, `tcpwrap`, `quota-warning` o cualquier otro.

`lista-listener-configuration` [parámetro de `service-configuration`]
`listeners`

Procesos de escucha para el servicio. Un proceso de escucha es un objeto `unix-listener-configuration`, un objeto `fifo-listener-configuration` o un objeto `inet-listener-configuration`. Su valor predeterminado es ‘`'()`’.

Los campos disponibles de `unix-listener-configuration` son:

`string path` [parámetro de `unix-listener-configuration`]

Ruta al archivo, relativa al campo `base-dir`. También se usa como nombre de la sección.

`string mode` [parámetro de `unix-listener-configuration`]

Modo de acceso del socket. Su valor predeterminado es ‘`"0600"`’.

`string user` [parámetro de `unix-listener-configuration`]

Usuaria que posee el socket. Su valor predeterminado es ‘`""`’.

`string group` [parámetro de `unix-listener-configuration`]

Grupo que posee el socket. Su valor predeterminado es ‘`""`’.

Los campos disponibles de `fifo-listener-configuration` son:

`string path` [parámetro de `fifo-listener-configuration`]
Ruta al archivo, relativa al campo `base-dir`. También se usa como nombre de la sección.

`string mode` [parámetro de `fifo-listener-configuration`]
Modo de acceso del socket. Su valor predeterminado es '`"0600"`'.

`string user` [parámetro de `fifo-listener-configuration`]
Usuaria que posee el socket. Su valor predeterminado es '`""`'.

`string group` [parámetro de `fifo-listener-configuration`]
Grupo que posee el socket. Su valor predeterminado es '`""`'.

Los campos disponibles de `inet-listener-configuration` son:

`string protocol` [parámetro de `inet-listener-configuration`]
El protocolo con el que se esperan las conexiones.

`string address` [parámetro de `inet-listener-configuration`]
La dirección en la que se escuchará, o vacío para escuchar en todas las direcciones. Su valor predeterminado es '`""`'.

`entero-no-negativo port` [parámetro de `inet-listener-configuration`]
El puerto en el que esperarán conexiones.

`boolean ssl?` [parámetro de `inet-listener-configuration`]
Si se usará SSL para este servicio; '`yes`' (sí), '`no`' o '`required`' (necesario). Su valor predeterminado es '`#t`'.

`entero-no-negativo client-limit` [parámetro de `service-configuration`]
Número máximo de conexiones simultáneas por cliente por proceso. Una vez se reciba este número de conexiones, la siguiente conexión entrante solicitará a Dovecot el inicio de un nuevo proceso. Si se proporciona el valor 0, se usa `default-client-limit`.

El valor predeterminado es '`0`'.

`entero-no-negativo service-count` [parámetro de `service-configuration`]
Número de conexiones a manejar antes de iniciar un nuevo proceso. Habitualmente los únicos valores útiles son 0 (ilimitadas) o 1. 1 es más seguro, pero 0 es más rápido. <doc/wiki/LoginProcess.txt>. Su valor predeterminado es '`1`'.

`entero-no-negativo process-limit` [parámetro de `service-configuration`]
Número máximo de procesos que pueden existir para este servicio. Si se proporciona el valor 0, se usa `default-process-limit`.

El valor predeterminado es '`0`'.

`entero-no-negativo` [parámetro de `service-configuration`]
`process-min-avail`
: Número de procesos que siempre permanecerán a la espera de más conexiones. Su valor predeterminado es '`0`'.

`entero-no-negativo vsz-limit` [parámetro de `service-configuration`]
: Si proporciona '`service-count 0`', probablemente necesitará incrementar este valor. Su valor predeterminado es '`256000000`'.

`dict-configuration dict` [parámetro de `dovecot-configuration`]
: Configuración de Dict, como la creada por el constructor `dict-configuration`.

  Los campos disponibles de `dict-configuration` son:

  `campos-libres entries` [parámetro de `dict-configuration`]
  : Una lista de pares clave-valor que este diccionario debe incorporar. Su valor predeterminado es '`'()`'.

`lista-passdb-configuration passdbs` [parámetro de `dovecot-configuration`]
: Una lista de configuraciones de passdb, cada una creada por el constructor `passdb-configuration`.

  Los campos disponibles de `passdb-configuration` son:

  `string driver` [parámetro de `passdb-configuration`]
  : El controlador que passdb debe usar. Entre los valores aceptados se incluye '`pam`', '`passwd`', '`shadow`', '`bsdauth`' y '`static`'. Su valor predeterminado es '`"pam"`'.

  `lista-cadenas-separada-espacios` [parámetro de `passdb-configuration`]
  `args`
  : Lista de parámetros separados por espacios para proporcionar al controlador passdb. Su valor predeterminado es '`""`'.

`lista-userdb-configuration userdbs` [parámetro de `dovecot-configuration`]
: Lista de configuraciones userdb, cada una creada por el constructor `userdb-configuration`.

  Los campos disponibles de `userdb-configuration` son:

  `string driver` [parámetro de `userdb-configuration`]
  : El controlador que userdb debe usar. Entre los valores aceptados se incluye '`passwd`' y '`static`'. Su valor predeterminado es '`"passwd"`'.

  `lista-cadenas-separada-espacios` [parámetro de `userdb-configuration`]
  `args`
  : Lista separada por espacios de parámetros usados para el controlador userdb. Su valor predeterminado es '`""`'.

  `parámetros-libres` [parámetro de `userdb-configuration`]
  `override-fields`
  : Sustituye los valores de campos de passwd. Su valor predeterminado es '`'()`'.

`plugin-configuration` [parámetro de `dovecot-configuration`]
`plugin-configuration`
Configuración del módulo, creada por el constructor `plugin-configuration`.

`lista-namespace-configuration` [parámetro de `dovecot-configuration`]
`namespaces`
Lista de espacios de nombres. Cada elemento de la lista debe crearse con el constructor `namespace-configuration`.

Los campos disponibles de `namespace-configuration` son:

`string name` [parámetro de `namespace-configuration`]
Nombre para este espacio de nombres.

`string type` [parámetro de `namespace-configuration`]
Tipo del espacio de nombres: '`private`', '`shared`' o '`public`'. Su valor predeterminado es '`"private"`'.

`string separator` [parámetro de `namespace-configuration`]
Hierarchy separator to use. You should use the same separator for all namespaces or some clients get confused. '`/`' is usually a good one. The default however depends on the underlying mail storage format. Defaults to '`""`'.

`string prefix` [parámetro de `namespace-configuration`]
Prefix required to access this namespace. This needs to be different for all namespaces. For example '`Public/`'. Defaults to '`""`'.

`string location` [parámetro de `namespace-configuration`]
Localización física de la bandeja de correo. En el mismo formato que la localización del correo, que también es su valor predeterminado. Su valor predeterminado es '`""`'.

`boolean inbox?` [parámetro de `namespace-configuration`]
Únicamente puede existir una bandeja de entrada (INBOX), y esta configuración define qué espacio de nombres la posee. Su valor predeterminado es '`#f`'.

`boolean hidden?` [parámetro de `namespace-configuration`]
If namespace is hidden, it's not advertised to clients via NAMESPACE extension. You'll most likely also want to set '`list? #f`'. This is mostly useful when converting from another server with different namespaces which you want to deprecate but still keep working. For example you can create hidden namespaces with prefixes '`~/mail/`', '`~%u/mail/`' and '`mail/`'. Defaults to '`#f`'.

`boolean list?` [parámetro de `namespace-configuration`]
Show the mailboxes under this namespace with the LIST command. This makes the namespace visible for clients that do not support the NAMESPACE extension. The special `children` value lists child mailboxes, but hides the namespace prefix. Defaults to '`#t`'.

`boolean subscriptions?` [parámetro de `namespace-configuration`]
El espacio de nombres maneja sus propias subscripciones. Si es `#f`, el espacio de nombres superior las maneja. El prefijo vacío siempre debe tener este valor como `#t`. Su valor predeterminado es '`#t`'.

`lista-mailbox-configuration mailboxes` [parámetro de `namespace-configuration`]
Lista de bandejas de correo predefinidas en este espacio de nombres. Su valor predeterminado es '`'()`'.

Los campos disponibles de `mailbox-configuration` son:

`string name` [parámetro de `mailbox-configuration`]
Nombre de esta bandeja de correo.

`string auto` [parámetro de `mailbox-configuration`]
Con '`create`' se crea de forma automática esta bandeja de correo. Con '`subscribe`' se crea y se suscribe a la bandeja de correo. Su valor predeterminado es '`"no"`'.

`lista-cadenas-separada-espacios special-use` [parámetro de `mailbox-configuration`]
Lista de atributos `SPECIAL-USE` de IMAP como se especifican en el RFC 6154. Entre los valores aceptados se incluye `\All`, `\Archive`, `\Drafts`, `\Flagged`, `\Junk`, `\Sent` y `\Trash`. Su valor predeterminado es '`'()`'.

`nombre-archivo base-dir` [parámetro de `dovecot-configuration`]
Directorio base donde se almacenan los datos de tiempo de ejecución. Su valor predeterminado es '`"/var/run/dovecot/"`'.

`string login-greeting` [parámetro de `dovecot-configuration`]
Mensaje de saludo para clientes. Su valor predeterminado es '`"Dovecot ready."`'.

`lista-cadenas-separada-espacios login-trusted-networks` [parámetro de `dovecot-configuration`]
Lista de rangos de red en los que se confía. Las conexiones desde estas IP tienen permitido forzar sus direcciones IP y puertos (para el registro y las comprobaciones de identidad). '`disable-plaintext-auth`' también se ignora en estas redes. Habitualmente aquí se especificarían los servidores proxy IMAP. Su valor predeterminado es '`'()`'.

`lista-cadenas-separada-espacios login-access-sockets` [parámetro de `dovecot-configuration`]
Lista de sockets para la comprobación de acceso al sistema (por ejemplo tcpwrap). Su valor predeterminado es '`'()`'.

`boolean verbose-proctitle?` [parámetro de `dovecot-configuration`]
Muestra títulos de procesamiento más detallados (en la postdata). Actualmente muestra el nombre de la usuaria y su dirección IP. Es útil para ver quién está usando procesos IMAP realmente (por ejemplo bandejas de correo compartidas o si el mismo identificador numérico de usuaria se usa para varias cuentas). Su valor predeterminado es '`#f`'.

`boolean shutdown-clients?` [parámetro de `dovecot-configuration`]
Determina si se deben finalizar todos los procesos cuando el proceso maestro de Dovecot termine su ejecución. El valor `#f` significa que Dovecot puede actualizarse sin forzar el cierre de las conexiones existentes (aunque esto puede ser un problema si la actualización se debe, por ejemplo, a la corrección de una vulnerabilidad). Su valor predeterminado es '`#t`'.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`doveadm-worker-count`
Si no es cero, ejecuta las ordenes del correo a través de este número de conexiones al servidor doveadm, en vez de ejecutarlas directamente en el mismo proceso. Su valor predeterminado es '`0`'.

`string doveadm-socket-path` [parámetro de `dovecot-configuration`]
Socket UNIX o máquina:puerto usados para la conexión al servidor doveadm. Su valor predeterminado es '`"doveadm-server"`'.

`lista-cadenas-separada-espacios` [parámetro de `dovecot-configuration`]
`import-environment`
Lista de variables de entorno que se preservan al inicio de Dovecot y se proporcionan a los procesos iniciados. También puede proporcionar pares clave=valor para proporcionar siempre dicha configuración específica.

`boolean disable-plaintext-auth?` [parámetro de `dovecot-configuration`]
Desactiva la orden LOGIN y todas las otras identificaciones en texto plano a menos que se use SSL/TLS (capacidad LOGINDISABLED). Tenga en cuenta que si la IP remota coincide con la IP local (es decir, se ha conectado desde la misma máquina), la conexión se considera segura y se permite la identificación en texto plano. Véase también la configuración '`ssl=required`'. Su valor predeterminado es '`#t`'.

`entero-no-negativo auth-cache-size` [parámetro de `dovecot-configuration`]
Tamaño de la caché de identificaciones (por ejemplo, '`#e10e6`'. 0 significa que está desactivada. Tenga en cuenta que bsdauth, PAM y vpopmail necesitan un valor en '`cache-key`' para que se use la caché. Su valor predeterminado es '`0`'.

`string auth-cache-ttl` [parámetro de `dovecot-configuration`]
Tiempo de vida de los datos almacenados en caché. Los registros de caché no se usan tras su expiración, *excepto* si la base de datos principal devuelve un fallo interno. También se intentan manejar los cambios de contraseña de manera automática: si la identificación previa de la usuaria fue satisfactoria, pero no esta última, la caché no se usa. En estos momentos únicamente funciona con identificación en texto plano. Su valor predeterminado es '`"1 hour"`'.

`string auth-cache-negative-ttl` [parámetro de `dovecot-configuration`]
Tiempo de vida para fallos de búsqueda en la caché (usuaria no encontrada, la contraseña no coincide). 0 desactiva completamente su almacenamiento en caché. Su valor predeterminado es '`"1 hour"`'.

`lista-cadenas-separada-espacios auth-realms` [parámetro de `dovecot-configuration`]
Lista de dominios para los mecanismos de identificación de SASL que necesite. Puede dejarla vacía si no desea permitir múltiples dominios. Muchos clientes simplemente usarán el primero de la lista, por lo que debe mantener el dominio predeterminado en primera posición. Su valor predeterminado es ‘`'()`’.

`string auth-default-realm` [parámetro de `dovecot-configuration`]
Dominio predeterminado usado en caso de no especificar ninguno. Esto se usa tanto en dominios SASL y como al añadir @dominio al nombre de usuaria en los ingresos al sistema a través de texto en claro. Su valor predeterminado es ‘`""`’.

`string auth-username-chars` [parámetro de `dovecot-configuration`]
Lista de caracteres permitidos en los nombres de usuaria. Si el nombre de usuaria proporcionado contiene un carácter no enumerado aquí, el login falla automáticamente. Es únicamente una comprobación adicional para asegurarse de que las usuarias no pueden explotar ninguna potencial vulnerabilidad con el escape de comillas en bases de datos SQL/LDAP. Si desea permitir todos los caracteres, establezca este valor a la cadena vacía. Su valor predeterminado es ‘`"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890.-_@"`’.

`string auth-username-translation` [parámetro de `dovecot-configuration`]
Traducciones de caracteres de nombres de usuaria antes de que se busque en las bases de datos. El valor contiene series de caracteres "original `->` transformado". Por ejemplo ‘`#@/@`’ significa que ‘`#`’ y ‘`/`’ se traducen en ‘`@`’. Su valor predeterminado es ‘`""`’.

`string auth-username-format` [parámetro de `dovecot-configuration`]
Formato proporcionado al nombre de usuaria antes de buscarlo en las bases de datos. Puede usar variables estándar aquí, por ejemplo %Lu transformará el nombre a minúsculas, %n eliminará el fragmento del dominio si se proporcionó, o ‘`%n-AT-%d`’ cambiaría ‘`@`’ en ‘`-AT-`’. Esta traducción se realiza tras los cambios de ‘`auth-username-translation`’. Su valor predeterminado es ‘`"%Lu"`’.

`string auth-master-user-separator` [parámetro de `dovecot-configuration`]
Si desea permitir que usuarias maestras ingresen mediante la especificación del nombre de usuaria maestra dentro de la cadena de nombre de usuaria normal (es decir, sin usar el mecanismo para ello implementado por SASL), puede especificar aquí el carácter separador. El formato es entonces `<usuaria><separador><usuaria maestra>`. UW-IMAP usa ‘`*`’ como separador, por lo que esa puede ser una buena elección. Su valor predeterminado es ‘`""`’.

`string auth-anonymous-username` [parámetro de `dovecot-configuration`]
Usuaria usada para las usuarias que ingresen al sistema con el mecanismo de SASL ANONYMOUS. Su valor predeterminado es ‘`"anonymous"`’.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`auth-worker-max-count`
Número máximo de procesos de trabajo dovecot-auth. Se usan para la ejecución de consultas bloqueantes a passdb y userdb (por ejemplo MySQL y PAM). Se crean y destruyen bajo demanda de manera automática. Su valor predeterminado es '`30`'.

`string auth-gssapi-hostname` [parámetro de `dovecot-configuration`]
Nombre de máquina usado en los nombres de GSSAPI principales. Por omisión se usa el nombre devuelto por gethostname(). Use '`$ALL`' (con comillas) para permitir todas las entradas keytab. Su valor predeterminado es '`""`'.

`string auth-krb5-keytab` [parámetro de `dovecot-configuration`]
Keytab de Kerberos usado para el mecanismo GSSAPI. Si no se especifica ninguno, se usa el predeterminado del sistema (habitualmente `/etc/krb5.keytab`). Puede ser necesario cambiar el servicio auth para que se ejecute como root y tenga permisos de lectura sobre este archivo. Su valor predeterminado es '`""`'.

`boolean auth-use-winbind?` [parámetro de `dovecot-configuration`]
Se identifica con NTLM y GSS-SPNEGO mediante el uso del daemon winbind de Samba y la herramienta auxiliar '`ntlm-auth`'. <doc/wiki/Authentication/Mechanisms/Winbind.txt>.
Su valor predeterminado es '`#f`'.

`nombre-archivo` [parámetro de `dovecot-configuration`]
`auth-winbind-helper-path`
Ruta al binario de la herramienta auxiliar '`ntlm-auth`' de Samba. Su valor predeterminado es '`"/usr/bin/ntlm_auth"`'.

`string auth-failure-delay` [parámetro de `dovecot-configuration`]
Tiempo de espera antes de responder a identificaciones fallidas. Su valor predeterminado es '`"2 secs"`'.

`boolean` [parámetro de `dovecot-configuration`]
`auth-ssl-require-client-cert?`
Es necesario un certificado de cliente SSL válido o falla la identificación. Su valor predeterminado es '`#f`'.

`boolean` [parámetro de `dovecot-configuration`]
`auth-ssl-username-from-cert?`
Toma el nombre de usuaria del certificado de cliente SSL, mediante el uso de `X509_NAME_get_text_by_NID()`, que devuelve el nombre común (CommonName) del nombre de dominio (DN) del sujeto del certificado. Su valor predeterminado es '`#f`'.

`lista-cadenas-separada-espacios` [parámetro de `dovecot-configuration`]
`auth-mechanisms`
Lista de mecanismos de identificación deseados. Los mecanismos permitidos son: '`plain`', '`login`', '`digest-md5`', '`cram-md5`', '`ntlm`', '`rpa`', '`apop`', '`anonymous`', '`gssapi`', '`otp`', '`key`', and '`gss-spnego`'. Véase también la opción de configuración '`disable-plaintext-auth`'.

`lista-cadenas-separada-espacios` [parámetro de `dovecot-configuration`]
`director-servers`
Lista de IP o nombres de máquina de los servidores directores, incluyendo este mismo. Los puertos se pueden especificar como ip:puerto. El puerto predeterminado es el mismo que el usado por el servicio director '`inet-listener`'. Su valor predeterminado es '`'()`'.

`lista-cadenas-separada-espacios` [parámetro de `dovecot-configuration`]
`director-mail-servers`
Lista de IP o nombres de máquina de los servidores motores de correo. Se permiten también rangos, como 10.0.0.10-10.0.0.30. Su valor predeterminado es '`'()`'.

`string director-user-expire` [parámetro de `dovecot-configuration`]
Por cuanto tiempo se redirige a las usuarias a un servidor específico tras pasar ese tiempo sin conexiones. Su valor predeterminado es '`"15 min"`'.

`string director-username-hash` [parámetro de `dovecot-configuration`]
Cómo se traduce el nombre de usuaria antes de aplicar el hash. Entre los valores útiles se incluye %Ln si la usuaria puede ingresar en el sistema con o sin @dominio, %Ld si las bandejas de correo se comparten dentro del dominio. Su valor predeterminado es '`"%Lu"`'.

`string log-path` [parámetro de `dovecot-configuration`]
Archivo de registro usado para los mensajes de error. '`syslog`' los envía a syslog, '`"/dev/stderr"`' a la salida de error estándar. Su valor predeterminado es '`"syslog"`'.

`string info-log-path` [parámetro de `dovecot-configuration`]
Archivo de registro usado para los mensajes informativos. Por omisión se usa '`log-path`'. Su valor predeterminado es '`""`'.

`string debug-log-path` [parámetro de `dovecot-configuration`]
Archivo de registro usado para los mensajes de depuración. Por omisión se usa '`info-log-path`'. Su valor predeterminado es '`""`'.

`string syslog-facility` [parámetro de `dovecot-configuration`]
Subsistema de syslog (facility) usado si se envía el registro a syslog. De manera habitual, si desea que no se use '`mail`', se usará local0..local7. Otros subsistemas estándar también están implementados. Su valor predeterminado es '`"mail"`'.

`boolean auth-verbose?` [parámetro de `dovecot-configuration`]
Registra los intentos de identificación infructuosos y las razones por los que fallaron. Su valor predeterminado es '`#f`'.

`string auth-verbose-passwords` [parámetro de `dovecot-configuration`]
En caso de no coincidir la contraseña, registra la contraseña que se intentó. Los valores aceptados son “no”, “plain” y “sha1”. “sha1” puede ser útil para diferenciar intentos de descubrir la contraseña por fuerza bruta frente a una usuaria simplemente intentando la misma contraseña una y otra vez. También puede recortar el valor a n caracteres mediante la adición de `":n"` (por ejemplo “sha1:6”). Su valor predeterminado es '`"no"`'.

`boolean auth-debug?` [parámetro de `dovecot-configuration`]
Registros aún más detallados para facilitar la depuración. Muestra, por ejemplo, las consultas SQL. Su valor predeterminado es '`#f`'.

`boolean auth-debug-passwords?` [parámetro de `dovecot-configuration`]
En caso de no coincidir la contraseña, registra las contraseñas y esquema usadas de manera que el problema se pueda depurar. La activación de este valor también provoca la activación de '`auth-debug`'. Su valor predeterminado es '`#f`'.

`boolean mail-debug?` [parámetro de `dovecot-configuration`]
Permite la depuración de los procesos de correo. Puede ayudarle a comprender los motivos en caso de que Dovecot no encuentre sus correos. Su valor predeterminado es '`#f`'.

`boolean verbose-ssl?` [parámetro de `dovecot-configuration`]
Muestra los errores a nivel de protocolo SSL. Su valor predeterminado es '`#f`'.

`string log-timestamp` [parámetro de `dovecot-configuration`]
Prefijo de cada línea registrada en el archivo. Los códigos % tienen el formato de strftime(3). Su valor predeterminado es '`"\"%b %d %H:%M:%S \""`'.

`lista-cadenas-separada-espacios` [parámetro de `dovecot-configuration`]
`login-log-format-elements`
Lista de elementos que se desea registrar. Los elementos que tengan valor de variable no-vacía se unen para la formación de una cadena separada por comas.

`string login-log-format` [parámetro de `dovecot-configuration`]
Formato del registro de ingresos al sistema. %s contiene la cadena '`login-log-format-elements`', %$ contiene los datos que se desean registrar. Su valor predeterminado es '`"%$: %s"`'.

`string mail-log-prefix` [parámetro de `dovecot-configuration`]
Prefijo de los registros de procesos de correo. Véase doc/wiki/Variables.txt para obtener una lista de variables que puede usar. Su valor predeterminado es '`"\"%s(%u)<%{pid}><%{session}>: \""`'.

`string deliver-log-format` [parámetro de `dovecot-configuration`]
Formato usado para el registro de las entregas de correo. Puede usar las variables:

| | |
|---|---|
| `%$` | Mensaje de estado de entrega (por ejemplo: '`saved to INBOX`') |
| `%m` | Message-ID |
| `%s` | Asunto |
| `%f` | De (dirección) |
| `%p` | Tamaño físico |
| `%w` | Tamaño virtual. |

El valor predeterminado es '`"msgid=%m: %$"`'.

`string mail-location` [parámetro de `dovecot-configuration`]
Localización de las bandejas de correo de las usuarias. El valor predeterminado está vacío, lo que significa que Dovecot intenta encontrar las bandejas de correo de manera automática. Esto no funciona si la usuaria no tiene todavía ningún correo, por lo que debe proporcionarle a Dovecot la ruta completa.

Si usa mbox, proporcionar una ruta al archivo de la bandeja entrada "INBOX" (por ejemplo `/var/mail/%u`) no es suficiente. También debe proporcionarle a Dovecot la ruta de otras bandejas de correo. Este es el llamado *directorio de correo raíz*, y debe ser la primera ruta proporcionada en la opción '`mail-location`'.

Existen algunas variables especiales que puede usar, por ejemplo:

'`%u`' nombre de usuaria

'`%n`' parte de la usuaria en usuaria@dominio, idéntica a %u si no existe dominio

'`%d`' parte del dominio en usuaria@dominio, vacía si no existe dominio

'`%h`' directorio de la usuaria

Véase doc/wiki/Variables.txt para obtener una lista completa. Algunos ejemplos:

'`maildir:~/Correo`'
'`mbox:~/correo:INBOX=/var/mail/%u`'
'`mbox:/var/mail/%d/%1n/%n:INDEX=/var/indexes/%d/%1n/%`'

El valor predeterminado es '`""`'.

`string mail-uid` [parámetro de `dovecot-configuration`]
Usuaria del sistema y grupo que realizan el acceso al correo. Si se usan varias, userdb puede forzar su valor al devolver campos uid o gid. Se pueden usar tanto identificadores numéricos como nominales. <doc/wiki/UserIds.txt>. Su valor predeterminado es '`""`'.

`string mail-gid` [parámetro de `dovecot-configuration`]
El valor predeterminado es '`""`'.

`string mail-privileged-group` [parámetro de `dovecot-configuration`]
Grupo para permitir de forma temporal operaciones privilegiadas. Actualmente esto se usa únicamente con la bandeja de entrada (INBOX), cuando su creación inicial o el bloqueo con archivo `.lock` falle. Habitualmente se le proporciona el valor '`"mail"`' para tener acceso a `/var/mail`. Su valor predeterminado es '`""`'.

`string mail-access-groups` [parámetro de `dovecot-configuration`]
Proporciona acceso a estos grupos suplementarios a los procesos de correo. Habitualmente se usan para configurar el acceso a bandejas de correo compartidas. Tenga en cuenta que puede ser peligroso establecerlos si las usuarias pueden crear enlaces simbólicos (por ejemplo si se configura el grupo '`mail`' aquí, `ln -s /var/mail ~/mail/var` puede permitir a una usuaria borrar las bandejas de correo de otras usuarias, o `ln -s /bandeja/compartida/secreta ~/mail/mibandeja` permitiría su lectura). Su valor predeterminado es '`""`'.

`string mail-attribute-dict` [`dovecot-configuration` parameter]
The location of a dictionary used to store `IMAP METADATA` as defined by RFC 5464 (`https://tools.ietf.org/html/rfc5464`).

The IMAP METADATA commands are available only if the "imap" protocol configuration's `imap-metadata?` field is '`#t`'.

El valor predeterminado es '`""`'.

`boolean mail-full-filesystem-access?` [parámetro de `dovecot-configuration`]
Allow full file system access to clients. There's no access checks other than what the operating system does for the active UID/GID. It works with both maildir and mboxes, allowing you to prefix mailboxes names with e.g. `/path/` or `~user/`. Defaults to '`#f`'.

`boolean mmap-disable?` [parámetro de `dovecot-configuration`]
No usa `mmap()` en absoluto. Es necesario si almacena índices en sistemas de archivos compartidos (NFS o sistemas de archivos en cluster). Su valor predeterminado es '`#f`'.

`boolean dotlock-use-excl?` [parámetro de `dovecot-configuration`]
Confía en el correcto funcionamiento de '`O_EXCL`' para la creación de archivos de bloqueo dotlock. NFS implementa '`O_EXCL`' desde la versión 3, por lo que debería ser seguro usarlo hoy en día de manera predeterminada. Su valor predeterminado es '`#t`'.

`string mail-fsync` [parámetro de `dovecot-configuration`]
Cuando se usarán las llamadas fsync() o fdatasync():

`optimized`
: Cuando sea necesario para evitar la perdida de datos importantes

`always`
: Útil con, por ejemplo, NFS, donde las escrituras con `write()` se aplazan

`never`
: Nunca se usa (mejor rendimiento, pero los fallos pueden producir pérdida de datos)

Su valor predeterminado es '`"optimized"`'.

`boolean mail-nfs-storage?` [parámetro de `dovecot-configuration`]
Mail storage exists in NFS. Set this to yes to make Dovecot flush NFS caches whenever needed. If you're using only a single mail server this isn't needed. Defaults to '`#f`'.

`boolean mail-nfs-index?` [parámetro de `dovecot-configuration`]
Mail index files also exist in NFS. Setting this to yes requires '`mmap-disable? #t`' and '`fsync-disable? #f`'. Defaults to '`#f`'.

`string lock-method` [parámetro de `dovecot-configuration`]
Método de bloqueo para los archivos de índice. Las alternativas son fcntl, flock y dotlock. Dotlock utiliza técnicas que pueden provocar un consumo de E/S mayor que otros métodos de bloqueo. Usuarias de NFS: flock no funciona, recuerde que se debe cambiar '`mmap-disable`'. Su valor predeterminado es '`"fcntl"`'.

`nombre-archivo mail-temp-dir` [parámetro de `dovecot-configuration`]
Directorio en el que LDA/LMTP almacena de manera temporal correos entrantes de de más de 128 kB. Su valor predeterminado es '`"/tmp"`'.

`entero-no-negativo first-valid-uid` [parámetro de `dovecot-configuration`]
Rango de UID aceptado para las usuarias. Principalmente es para asegurarse de que las usuarias no pueden ingresar en el sistema como un daemon u otras usuarias del sistema. Tenga en cuenta que el binario de dovecot tiene código que impide el ingreso al sistema como root y no puede llevarse a cabo incluso aunque '`first-valid-uid`' tenga el valor 0. Su valor predeterminado es '`500`'.

`entero-no-negativo last-valid-uid` [parámetro de `dovecot-configuration`]
El valor predeterminado es '`0`'.

`entero-no-negativo first-valid-gid` [parámetro de `dovecot-configuration`]
Rango de GID aceptado para las usuarias. Las usuarias que no posean GID válido como ID primario de grupo no tienen permitido el ingreso al sistema. Si la usuaria es miembro de grupos suplementarios con GID no válido, no se activan dichos grupos. Su valor predeterminado es '`1`'.

`entero-no-negativo last-valid-gid` [parámetro de `dovecot-configuration`]
El valor predeterminado es '`0`'.

`entero-no-negativo mail-max-keyword-length` [parámetro de `dovecot-configuration`]
Longitud máxima permitida para nombres de palabras clave del correo. El límite actúa únicamente en la creación de nuevas palabras clave. Su valor predeterminado es '`50`'.

`lista-archivos-sep-dos-puntos valid-chroot-dirs` [parámetro de `dovecot-configuration`]
Lista de directorios bajo los que se permite realizar la llamada al sistema chroot a procesos de correo (es decir, `/var/mail` permitiría realizar la llamada también en `/var/mail/sub/directorio`). Esta configuración no afecta a la configuración de '`login-chroot`', '`mail-chroot`' o la de chroot para identificación. En caso de estar vacía, se ignora '`"/./"`' en los directorios de usuarias. AVISO: Nunca añada directorios que las usuarias locales puedan modificar, puesto que podría permitir vulnerar el control de acceso como "root". Habitualmente esta opción se activa únicamente si no permite a las usuarias acceder a un intérprete de órdenes. <doc/wiki/Chrooting.txt>. Su valor predeterminado es '`'()`'.

`string mail-chroot` [parámetro de `dovecot-configuration`]
Directorio predeterminado de la llamada al sistema chroot para los procesos de correo. El valor puede forzarse para usuarias específicas en la base de datos proporcionando '`/./`' en el directorio de la usuaria (por ejemplo, '`/home/./usuaria`' llama a chroot con `/home`). Tenga en cuenta que habitualmente no es necesario llamar a chroot, Dovecot no permite a las usuarias el acceso a archivos más allá de su directorio de correo en cualquier caso. Si sus directorios de usuaria contienen el directorio de chroot como prefijo, agregue '`/.`' al final de '`mail-chroot`'. <doc/wiki/Chrooting.txt>. Su valor predeterminado es '`""`'.

`nombre-archivo auth-socket-path` [parámetro de `dovecot-configuration`]
Ruta al socket de UNIX al servidor maestro de identificación para la búsqueda de usuarias. Se usa por parte de imap (para usuarias compartidas) y lda. Su valor predeterminado es '`"/var/run/dovecot/auth-userdb"`'.

`nombre-archivo mail-plugin-dir` [parámetro de `dovecot-configuration`]
Directorio en el que se buscarán módulos de correo. Su valor predeterminado es '`"/usr/lib/dovecot"`'.

`lista-cadena-separada-espacios mail-plugins` [parámetro de `dovecot-configuration`]
Lista de módulos cargados en todos los servicios. Los módulos específicos para IMAP, LDA, etc. se añaden en esta lista en sus propios archivos .conf. Su valor predeterminado es '`'()`'.

`entero-no-negativo mail-cache-min-mail-count` [parámetro de `dovecot-configuration`]
El número mínimo de correos en una bandeja antes de que las actualizaciones se realicen en un archivo de caché. Permite optimizar el comportamiento de Dovecot para reducir la tasa de escritura en el disco, lo que produce un aumento de la tasa de lectura. Su valor predeterminado es '`0`'.

`string mailbox-idle-check-interval` [parámetro de `dovecot-configuration`]
Cuando se ejecute la orden IDLE, la bandeja de correo se comprueba de vez en cuando para comprobar si existen nuevos correos o se han producido otros cambios. Esta configuración define el tiempo mínimo entre dichas comprobaciones. Dovecot también puede usar dnotify, inotify y kqueue para recibir información inmediata sobre los cambios que ocurran. Su valor predeterminado es '`"30 secs"`'.

`boolean mail-save-crlf?` [parámetro de `dovecot-configuration`]
Save mails with CR+LF instead of plain LF. This makes sending those mails take less CPU, especially with sendfile() syscall with Linux and FreeBSD. But it also creates a bit more disk I/O which may just make it slower. Also note that if other software reads the mboxes/maildirs, they may handle the extra CRs wrong and cause problems. Defaults to '`#f`'.

`boolean maildir-stat-dirs?` [parámetro de `dovecot-configuration`]
De manera predeterminada la orden LIST devuelve todas las entradas en maildir cuyo nombre empiece en punto. La activación de esta opción hace que Dovecot únicamente devuelva las entradas que sean directorios. Esto se lleva a cabo llamando a stat() con cada entrada, lo que causa mayor E/S del disco. (En sistemas que proporcionen valor a '`dirent->d_type`' esta comprobación no tiene coste alguno y se realiza siempre independientemente del valor configurado aquí). Su valor predeterminado es '`#f`'.

`boolean maildir-copy-with-hardlinks?` [parámetro de `dovecot-configuration`]
Cuando se copia un mensaje, se usan enlaces duros cuando sea posible. Esto mejora mucho el rendimiento, y es difícil que produzca algún efecto secundario. Su valor predeterminado es '`#t`'.

`boolean maildir-very-dirty-syncs?` [parámetro de `dovecot-configuration`]
Asume que Dovecot es el único MUA que accede a maildir: Recorre el directorio cur/ únicamente cuando cambie su mtime de manera inesperada o cuando no se pueda encontrar el correo de otra manera. Su valor predeterminado es '`#f`'.

`lista-cadena-separada-espacios mbox-read-locks` [parámetro de `dovecot-configuration`]
Qué métodos de bloqueo deben usarse para el bloque de mbox. Hay cuatro disponibles:

`dotlock` Crea un archivo <bandeja>.lock. Esta es la solución más antigua y más segura con NFS. Si desea usar un directorio como /var/mail/, las usuarias necesitarán acceso de escritura a dicho directorio.

`dotlock-try`
Lo mismo que dotlock, pero si se produce un fallo debido a los permisos o a que no existe suficiente espacio en disco, simplemente se omite el bloqueo.

`fcntl` Use este a ser posible. Funciona también con NFS si se usa lockd.

`flock` Puede no existir en todos los sistemas. No funciona con NFS.

`lockf` Puede no existir en todos los sistemas. No funciona con NFS.

Puede usar múltiples métodos de bloqueo; en ese caso el orden en el que se declaran es importante para evitar situaciones de bloqueo mutuo en caso de que otros MTA/MUA usen también múltiples métodos de bloqueo. Algunos sistemas operativos no permiten el uso de varios de ellos de manera simultánea.

`lista-cadena-separada-espacios mbox-write-locks` [parámetro de `dovecot-configuration`]

`string mbox-lock-timeout` [parámetro de `dovecot-configuration`]
Tiempo máximo esperado hasta el bloqueo (de todos los archivos) antes de interrumpir la operación. Su valor predeterminado es '`"5 mins"`'.

`string mbox-dotlock-change-timeout` [parámetro de `dovecot-configuration`]
Si existe el archivo dotlock pero la bandeja no se ha modificado de ninguna manera, ignora el archivo de bloqueo tras este tiempo. Su valor predeterminado es '`"2 mins"`'.

`boolean mbox-dirty-syncs?` [parámetro de `dovecot-configuration`]
Cuando el archivo mbox cambie de manera inesperada, se tiene que leer completamente para encontrar los cambios. Si es grande puede tardar bastante tiempo. Deido a que el cambio habitualmente es un nuevo correo añadido al final, sería más rápido únicamente leer los correos nuevos. Si se activa esta opción, Dovecot hace esto pero de todos modos vuelve a leer el archivo mbox al completo cuando algo en la bandeja no se encuentra como se esperaba. La única desventaja real de esta configuración es que si otro MUA cambia las opciones de los mensajes, Dovecot no tiene constancia de ello de manera inmediata. Tenga en cuenta que una sincronización completa se lleva a cabo con las órdenes SELECT, EXAMINE, EXPUNGE y CHECK. Su valor predeterminado es '`#t`'.

`boolean mbox-very-dirty-syncs?` [parámetro de `dovecot-configuration`]
Como '`mbox-dirty-syncs`', pero no realiza sincronizaciones completas tampoco con las órdenes SELECT, EXAMINE, EXPUNGE o CHECK. Si se configura este valor, '`mbox-dirty-syncs`' se ignora. Su valor predeterminado es '`#f`'.

`boolean mbox-lazy-writes?` [parámetro de `dovecot-configuration`]
Retrasa la escritura de las cabeceras de mbox hasta que se realice una escritura completa sincronizada (con las órdenes EXPUNGE y CHECK, y al cerrar la bandeja de correo). Es útil especialmente con POP3, donde el cliente habitualmente borra todos los correos. La desventaja es que nuestros cambios no son visibles para otros MUA de manera inmediata. Su valor predeterminado es '`#t`'.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`mbox-min-index-size`
Si el tamaño del archivo mbox es menor que este valor (por ejemplo, 100k), no escribe archivos de índice. Si el archivo de índice ya existe, todavía se usa para la lectura, pero no se actualiza. Su valor predeterminado es '`0`'.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`mdbox-rotate-size`
Tamaño máximo del archivo dbox hasta su rotación. Su valor predeterminado es '`10000000`'.

`string mdbox-rotate-interval` [parámetro de `dovecot-configuration`]
Antigüedad máxima del archivo dbox hasta que se produce su rotación. Habitualmente en días. El día comienza a medianoche, por lo que 1d = hoy, 2d = ayer, etcétera. 0 = comprobación desactivada. Su valor predeterminado es '`"1d"`'.

`boolean mdbox-preallocate-space?` [parámetro de `dovecot-configuration`]
Cuando se crean nuevos archivos mdbox, reserva inmediatamente un tamaño de '`mbox-rotate-size`'. Actualmente esta configuración funciona únicamente en Linux con determinados sistemas de archivos (ext4, xfs). Su valor predeterminado es '`#f`'.

`string mail-attachment-dir` [parámetro de `dovecot-configuration`]
sdbox y mdbox permiten el almacenamiento de adjuntos en archivos externos, lo que permite también su almacenamiento único. Otros motores no lo permiten por el momento.

AVISO: Esta característica todavía no se ha probado mucho. Su uso queda bajo su propia responsabilidad.

Directorio raíz donde almacenar los adjuntos de los correos. Se desactiva en caso de estar vacío. Su valor predeterminado es '`""`'.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`mail-attachment-min-size`
Los adjuntos de menor tamaño que este valor no se almacenan externamente. También es posible la escritura de un módulo que deshabilite el almacenamiento externo de adjuntos específicos. Su valor predeterminado es '`128000`'.

`string mail-attachment-fs` [parámetro de `dovecot-configuration`]
Motor del sistema de archivos usado para el almacenamiento de adjuntos:

`posix` Dovecot no lleva a cabo el SiS (aunque esto puede ayudar a la deduplicación del propio sistema de archivos)

`sis posix` SiS con comparación inmediata byte-por-byte durante el almacenamiento.

`sis-queue posix`
SiS mediante comparación retrasada y deduplicación.

El valor predeterminado es '`"sis posix"`'.

`string mail-attachment-hash` [parámetro de `dovecot-configuration`]
Formato del hash usado en los archivos adjuntos. Puede añadir cualquier texto y variables: `%{md4}`, `%{md5}`, `%{sha1}`, `%{sha256}`, `%{sha512}`, `%{size}`. Las variables pueden reducirse, por ejemplo `%{sha256:80}` devuelve únicamente los primeros 80 bits. Su valor predeterminado es '`"%{sha1}"`'.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`default-process-limit`
El valor predeterminado es '`100`'.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`default-client-limit`
El valor predeterminado es '`1000`'.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`default-vsz-limit`
Límite predeterminado del tamaño de memoria virtual (VSZ) para procesos del servicio. Esta principalmente orientado a la captura y parada de procesos que pierden memoria antes de que utilicen toda la disponible. Su valor predeterminado es '`256000000`'.

`string default-login-user` [parámetro de `dovecot-configuration`]
Usuaria de ingreso al sistema para los procesos de ingreso al sistema. Es la usuaria en la que menos se confía en el sistema Dovecot. No debería tener acceso a nada en absoluto. Su valor predeterminado es '`"dovenull"`'.

`string default-internal-user` [parámetro de `dovecot-configuration`]
Usuaria interna usada por procesos sin privilegios. Debería ser distinta a la usuaria de ingreso, de modo que los procesos de ingreso al sistema no interfieran con otros procesos. Su valor predeterminado es '`"dovecot"`'.

`string ssl?` [parámetro de `dovecot-configuration`]
Si se permite SSL/TLS: '`yes`' (sí), '`no`', '`required`' (necesario). <doc/wiki/SSL.txt>. Su valor predeterminado es '`"required"`'.

`string ssl-cert` [parámetro de `dovecot-configuration`]
Certificado X.509 de SSL/TLS codificado con PEM (clave pública). Su valor predeterminado es '`"</etc/dovecot/default.pem"`'.

`string ssl-key` [parámetro de `dovecot-configuration`]
Clave privada de SSL/TLS codificada con PEM. La clave se abre antes de renunciar a los privilegios de root, por lo que debe mantenerse legible únicamente para root. Su valor predeterminado es '`"</etc/dovecot/private/default.pem"`'.

`string ssl-key-password` [parámetro de `dovecot-configuration`]
Si el archivo de la clave está protegido por contraseña, introduzca dicha contraseña aquí. De manera alternativa, puede proporcionarla al iniciar dovecot con el parámetro -p. Como este archivo es habitualmente legible por todo el mundo, puede que desee desplazar esta opción a un archivo diferente. Su valor predeterminado es '`""`'.

`string ssl-ca` [parámetro de `dovecot-configuration`]
Certificado usado como autoridad de certificación de confianza codificado en PEM. Configure este valor únicamente si tiene intención de usar '`ssl-verify-client-cert? #t`'. El archivo debe contener el archivo de la o las AC seguido de las CRL correspondientes (por ejemplo, '`ssl-ca </etc/ssl/certs/ca.pem`'). Su valor predeterminado es '`""`'.

`boolean ssl-require-crl?` [parámetro de `dovecot-configuration`]
Es necesario que la comprobación de CRL sea satisfactoria para certificados de clientes.. Su valor predeterminado es '`#t`'.

`boolean ssl-verify-client-cert?` [parámetro de `dovecot-configuration`]
Solicita al cliente el envío de un certificado. Si también desea que sea un requisito, proporcione '`auth-ssl-require-client-cert? #t`' en la sección de identificación. Su valor predeterminado es '`#f`'.

`string ssl-cert-username-field` [parámetro de `dovecot-configuration`]
Cual es el campo del certificado que determina el nombre de usuaria. "commonName" y "x500UniqueIdentifier" son las opciones habituales. También tendrá proporcionar '`auth-ssl-username-from-cert? #t`'. Su valor predeterminado es '`"commonName"`'.

`string ssl-min-protocol` [parámetro de `dovecot-configuration`]
Versión mínima aceptada del protocolo SSL. Su valor predeterminado es '`"TLSv1"`'.

`string ssl-cipher-list` [parámetro de `dovecot-configuration`]
Protocolos de cifrado de SSL usados. Su valor predeterminado es '`"ALL:!kRSA:!SRP:!kDHd:!DSS:!aNULL:!eNULL:!EXPORT:!DES:!3DES:!MD5:!PSK:!RC4:!ADH:!LOW@S`

`string ssl-crypto-device` [parámetro de `dovecot-configuration`]
Dispositivo de cifrado de SSL usado, ejecute "openssl engine" para obtener los valores aceptados. Su valor predeterminado es '`""`'.

`string postmaster-address` [parámetro de `dovecot-configuration`]
Dirección usada cuando se notifiquen correos rechazados. %d expande al dominio receptor. Su valor predeterminado es '`"postmaster@%d"`'.

`string hostname` [parámetro de `dovecot-configuration`]
Nombre de máquina usado en diversas partes de los correos enviados (por ejemplo, en Message-Id) y en las respuestas LMTP. Su valor predeterminado es <el nombre real de la máquina>@dominio. Su valor predeterminado es '`""`'.

`boolean quota-full-tempfail?` [parámetro de `dovecot-configuration`]
Si la usuaria supera la cuota, devuelve un fallo temporal en vez de rechazar el correo. Su valor predeterminado es '`#f`'.

`nombre-archivo sendmail-path` [parámetro de `dovecot-configuration`]
Binario usado para el envío de correos. Su valor predeterminado es '`"/usr/sbin/sendmail"`'.

`string submission-host` [parámetro de `dovecot-configuration`]
Si no está vacío, envía el correo a través de esta máquina[:puerto] SMTP en vez de usar sendmail Su valor predeterminado es '`""`'.

`string rejection-subject` [parámetro de `dovecot-configuration`]
Asunto: cabecera usada en el rechazo de correos. Puede usar las mismas variables que las indicadas en '`rejection-reason`' a continuación. Su valor predeterminado es '`"Rejected: %s"`'.

`string rejection-reason` [parámetro de `dovecot-configuration`]
Mensaje de error legible por personas para el rechazo de correos. Puede usar variables:

`%n` CRLF

`%r` razón

`%s` asunto original

`%t` receptora

El valor predeterminado es '`"Your message to <%t> was automatically rejected:%n%r"`'.

`string recipient-delimiter` [parámetro de `dovecot-configuration`]
Carácter delimitador entre la parte local y el detalle en las direcciones de correo. Su valor predeterminado es '`"+"`'.

`string` [parámetro de `dovecot-configuration`]
`lda-original-recipient-header`
Cabecera de donde se obtiene la dirección receptora original (la dirección de SMTP RCPT TO:) en caso de no estar disponible en otro lugar. El parámetro -a de dovecot-lda reemplaza este valor. Una cabecera usada para esto de manera común es X-Original-To. Su valor predeterminado es '`""`'.

`boolean lda-mailbox-autocreate?` [parámetro de `dovecot-configuration`]
¿Se debe crear una bandeja de correo no existente de manera automática al almacenar un correo? Su valor predeterminado es '`#f`'.

`boolean lda-mailbox-autosubscribe?` [parámetro de `dovecot-configuration`]
¿También Se deben crear suscripciones de manera automática a las bandejas de correo creadas? Su valor predeterminado es '`#f`'.

`entero-no-negativo` [parámetro de `dovecot-configuration`]
`imap-max-line-length`

Longitud máxima de la línea de órdenes de IMAP. Algunos clientes generan líneas de órdenes muy largas con bandejas de correo enormes, por lo que debe incrementarlo si recibe los errores “Too long argument” (parámetro demasiado largo) o "IMAP command line too large" (línea de órdenes de IMAP demasiado grande). Su valor predeterminado es ‘64000’.

`string imap-logout-format` [parámetro de `dovecot-configuration`]

Formato de la cadena de IMAP de salida del sistema:

`%i` número total de bytes leídos del cliente

`%o` número total de bytes enviados al cliente.

Véase `doc/wiki/Variables.txt` para obtener una lista completa de todas las variables que puede usar. Su valor predeterminado es ‘`"in=%i out=%o deleted=%{deleted} expunged=%{expunged} trashed=%{trashed} hdr_count=%{fetch_hdr_count} hdr_bytes=%{fetch_hdr_bytes} body_count=%{fetch_body_count} body_bytes=%{fetch_body_bytes}"`’.

`string imap-capability` [parámetro de `dovecot-configuration`]

Fuerza el valor de la respuesta de IMAP CAPABILITY. Si el valor comienza con ’+’, añade las capacidades especificadas sobre las predeterminadas (por ejemplo, +XFOO XBAR). Su valor predeterminado es ‘""’.

`string imap-idle-notify-interval` [parámetro de `dovecot-configuration`]

Durante cuanto tiempo se espera entre notificaciones "OK Still here" cuando el cliente se encuentre en estado IDLE. Su valor predeterminado es ‘`"2 mins"`’.

`string imap-id-send` [parámetro de `dovecot-configuration`]

Nombres y valores de campos de identificación (ID) que se enviarán a los clientes. El uso de * como un valor hace que Dovecot utilice el valor predeterminado. Los siguientes campos tienen actualmente valores predeterminados: name, version, os, os-version, support-url, support-email. Su valor predeterminado es ‘""’.

`string imap-id-log` [parámetro de `dovecot-configuration`]

Campos de identificación (ID) enviados para su registro por cliente. `*` significa todos. Su valor predeterminado es ‘""’.

`lista-cadena-separada-espacios` [parámetro de `dovecot-configuration`]
`imap-client-workarounds`

Soluciones temporales para varios errores de clientes:

`delay-newmail`

Envía notificaciones de nuevo correo EXISTS/RECENT únicamente en respuesta a ordenes NOOP o CHECK. Algunos clientes las ignoran en otro caso, por ejemplo OSX Mail (<v2.1). Outlook Express tiene problemas mayores no obstante, sin esto puede mostrar a la usuaria errores "Message no longer in server". Tenga en cuenta que OE6 también falla con esta solución temporal si la sincronización se establece como "Headers Only".

`tb-extra-mailbox-sep`
: Thunderbird se confunde de algún modo con LAYOUT=fs (mbox y dbox) y añade sufijos '`/`' adicionales a los nombres de las bandejas de correo. Esta opción hace que Dovecot ignore el carácter '`/`' adicional en vez de tratarlo como un nombre de bandeja de correo no válido.

`tb-lsub-flags`
: Muestra las opciones \Noselect para respuestas LSUB con LAYOUT=fs (por ejemplo mbox). Esto permite a Thunderbird ser consciente de que no se pueden seleccionar y mostrarlas en gris, en vez de únicamente mostrar después el mensaje de error "no seleccionable".

El valor predeterminado es '`'()`'.

`string imap-urlauth-host` [parámetro de `dovecot-configuration`]
: Máquina permitida en las URL URLAUTH enviadas por el cliente. "*" permite todas. Su valor predeterminado es '`""`'.

¡Miau! Muchas opciones de configuración. Lo bueno es que Guix tiene una interfaz completa al lenguage de configuración de Dovecot. Esto no permite únicamente declarar configuraciones de forma bonita, sino que también ofrece capacidades reflexivas: las usuarias pueden escribir código en Scheme para inspeccionar y transformar configuraciones.

No obstante, puede ser que únicamente desee usar un archivo `dovecot.conf` existente. En ese caso, puede proporcionar un objeto `opaque-dovecot-configuration` como parámetro `#:config` a `dovecot-service`. Como su nombre en inglés indica, una configuración opaca no tiene gran capacidad reflexiva.

Los campos disponibles de `opaque-dovecot-configuration` son:

`package dovecot` [parámetro de `opaque-dovecot-configuration`]
: El paquete dovecot.

`string string` [parámetro de `opaque-dovecot-configuration`]
: El contenido de `dovecot.conf`, como una cadena.

Por ejemplo, si su `dovecot.conf` fuese simplemente la cadena vacía, podría instanciar un servicio dovecot de esta manera:

```
(dovecot-service #:config
                 (opaque-dovecot-configuration
                  (string "")))
```

## Servicio OpenSMTPD

`opensmtpd-service-type` [Variable]
: Es el tipo del servicio OpenSMTPD (`https://www.opensmtpd.org`), cuyo valor debe ser un objeto `opensmtpd-configuration` como en este ejemplo:

```
(service opensmtpd-service-type
         (opensmtpd-configuration
           (config-file (local-file "./mi-smtpd.conf"))))
```

`opensmtpd-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de opensmtpd.

`package` (predeterminado: *opensmtpd*)
El objeto paquete del servidor SMTP OpenSMTPD.

`shepherd-requirement` (default: `'()`)
This option can be used to provide a list of symbols naming Shepherd services that this service will depend on, such as `'networking` if you want to configure OpenSMTPD to listen on non-loopback interfaces.

`config-file` (default: `%default-opensmtpd-config-file`)
Objeto "tipo-archivo" del archivo de configuración de OpenSMTPD usado. De manera predeterminada escucha en la interfaz de red local, y pone a disposición de usuarias y daemon de la máquina local el servicio de correo, así como el envío de correo a servidores remotos. Ejecute `man smtpd.conf` para obtener más información.

`setgid-commands?` (default: `#t`)
Make the following commands setgid to `smtpq` so they can be executed: `smtpctl`, `sendmail`, `send-mail`, `makemap`, `mailq`, and `newaliases`. See Section 11.11 [Programas con setuid], page 620, for more information on setgid programs.

## Servicio Exim

`exim-service-type` [Variable]

Este es el tipo del agente de transferencia de correo (MTA) Exim (`https://exim.org`), cuyo valor debe ser un objeto `exim-configuration` como en este ejemplo:

```
(service exim-service-type
         (exim-configuration
           (config-file (local-file "./mi-exim.conf"))))
```

Para usar un servicio `exim-service-type` debe tener también un servicio `mail-aliases-service-type` presente en su declaración `operating-system` (incluso aunque no exista ningún alias).

`exim-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de exim.

`package` (predeterminado: *exim*)
El objeto paquete del servidor Exim.

`config-file` (predeterminado: `#f`)
File-like object of the Exim configuration file to use. If its value is `#f` then use the default configuration file from the package provided in `package`. The resulting configuration file is loaded after setting the `exim_user` and `exim_group` configuration variables.

## Servicio Getmail

`getmail-service-type` [Variable]
This is the type of the Getmail (`http://pyropus.ca/software/getmail/`) mail retriever, whose value should be a `getmail-configuration`.

Los campos disponibles de `getmail-configuration` son:

`símbolo name` [parámetro de `getmail-configuration`]
Un símbolo que identifique el servicio getmail.

El valor predeterminado es '`"unset"`'.

`package package` [parámetro de `getmail-configuration`]
El paquete getmail usado.

`string user` [parámetro de `getmail-configuration`]
Usuaria que ejecuta getmail.

El valor predeterminado es '`"getmail"`'.

`string group` [parámetro de `getmail-configuration`]
Grupo que ejecuta getmail.

El valor predeterminado es '`"getmail"`'.

`string directory` [parámetro de `getmail-configuration`]
El directorio usado para getmail.

El valor predeterminado es '`"/var/lib/getmail/default"`'.

`getmail-configuration-file rcfile` [parámetro de `getmail-configuration`]
El archivo de configuración de getmail usado.

Los campos disponibles de `getmail-configuration-file` son:

`getmail-retriever-configuration retriever` [parámetro de `getmail-configuration-file`]
De qué cuenta de correo obtener el correo, y cómo acceder a dicha cuenta.

Los campos disponibles de `getmail-retriever-configuration` son:

`string type` [parámetro de `getmail-retriever-configuration`]
El controlador que userdb debe usar. Entre los valores aceptados se incluye '`passwd`' y '`static`'.

El valor predeterminado es '`"SimpleIMAPSSLRetriever"`'.

`string server` [parámetro de `getmail-retriever-configuration`]
Usuaria que ejecutará el servidor de correo.

El valor predeterminado es '`unset`'.

`string username` [parámetro de `getmail-retriever-configuration`]
Usuaria que ejecutará el servidor de correo.

El valor predeterminado es '`unset`'.

`entero-no-negativo` [parámetro de `getmail-retriever-configuration`]
`port`
Número de puerto al que conectarse.

El valor predeterminado es '`#f`'

`string password` [parámetro de `getmail-retriever-configuration`]
Sustituye los valores de campos de passwd.

El valor predeterminado es '`""`'.

`list` [parámetro de `getmail-retriever-configuration`]
`password-command`
Sustituye los valores de campos de passwd.

El valor predeterminado es '`'()`'.

`string keyfile` [parámetro de `getmail-retriever-configuration`]
Archivo de claves con formato PEM usado para la negociación TLS.

El valor predeterminado es '`""`'.

`string certfile` [parámetro de `getmail-retriever-configuration`]
Archivo de certificado con formato PEM usado para la negociación TLS.

El valor predeterminado es '`""`'.

`string ca-certs` [parámetro de `getmail-retriever-configuration`]
Certificados de autoridad de certificación (CA) usados.

El valor predeterminado es '`""`'.

`parameter-alist` [parámetro de `getmail-retriever-configuration`]
`extra-parameters`
Parámetros adicionales del receptor de correo.

El valor predeterminado es '`'()`'.

`getmail-destination-configuration` [parámetro de `getmail-configuration-file`]
`destination`
Qué hacer con los mensajes obtenidos.

Los campos disponibles de `getmail-destination-configuration` son:

`string type` [parámetro de `getmail-destination-configuration`]
Tipo de destino del correo. Entre los valores válidos se incluye '`Maildir`', '`Mboxrd`' y '`MDA_external`'.

El valor predeterminado es '`unset`'.

`string-or-filelike` [parámetro de `getmail-destination-configuration`]
`path`
Opción de ruta para el destino del correo. El comportamiento depende del tipo seleccionado.

El valor predeterminado es '`""`'.

`parameter-alist` [parámetro de `getmail-destination-configuration`]
`extra-parameters`

Parámetros adicionales del destino.

El valor predeterminado es '`'()`'.

`getmail-options-configuration` [parámetro de `getmail-configuration-file`]
`options`

Configuración de getmail.

Los campos disponibles de `getmail-options-configuration` son:

`entero-no-negativo` [parámetro de `getmail-options-configuration`]
`verbose`

If set to '`0`', getmail will only print warnings and errors. A value of '`1`' means that messages will be printed about retrieving and deleting messages. If set to '`2`', getmail will print messages about each of its actions.

El valor predeterminado es '`1`'.

`boolean read-all` [parámetro de `getmail-options-configuration`]

Si es verdadero, getmail obtendrá todos los mensajes disponibles. En otro caso, únicamente recupera mensajes que no se hayan visto previamente.

El valor predeterminado es '`#t`'

`boolean delete` [parámetro de `getmail-options-configuration`]

Si se proporciona un valor verdadero, los mensajes se borrarán del servidor tras su recuperación y entrega posterior satisfactoria. En otro caso, los mensajes permanecerán en el servidor.

El valor predeterminado es '`#f`'

`entero-no-negativo` [parámetro de `getmail-options-configuration`]
`delete-after`

Getmail borrará los mensajes tras este número de días después de haberlos visto, si han sido entregados. Esto significa que los mensajes se mantendrán en el servidor este número de días tras entregarlos. El valor '`0`' desactiva esta característica.

El valor predeterminado es '`0`'.

`entero-no-negativo` [parámetro de `getmail-options-configuration`]
`delete-bigger-than`

Borra los mensajes de tamaño mayor que estos bytes tras recibirlos, incluso si las opciones "delete" y "delete-after" están desactivadas. El valor '`0`' desactiva esta característica.

El valor predeterminado es '`0`'.

`entero-no-negativo` [parámetro de `getmail-options-configuration`]
`max-bytes-per-session`

Obtiene mensajes hasta este número de bytes en total antes de cerrar la sesión con el servidor. El valor '`0`' desactiva esta característica.

El valor predeterminado es '`0`'.

`entero-no-negativo` [parámetro de `getmail-options-configuration`]
`max-message-size`
No obtiene mensajes con mayor tamaño que este número de bytes. El valor '`0`' desactiva esta característica.

El valor predeterminado es '`0`'.

`boolean` [parámetro de `getmail-options-configuration`]
`delivered-to`
Si es verdadero, getmail añadirá una cabecera "Delivered-To" a los mensajes.

El valor predeterminado es '`#t`'

`boolean received` [parámetro de `getmail-options-configuration`]
Si es verdadero, getmail añadirá una cabecera "Received" a los mensajes.

El valor predeterminado es '`#t`'

`string message-log` [parámetro de `getmail-options-configuration`]
Getmail generará un registro de sus acciones en el archivo nombrado. El valor '`""`' desactiva esta característica.

El valor predeterminado es '`""`'.

`boolean` [parámetro de `getmail-options-configuration`]
`message-log-syslog`
Si es verdadero, getmail registrará sus acciones a través del registro del sistema.

El valor predeterminado es '`#f`'

`boolean` [parámetro de `getmail-options-configuration`]
`message-log-verbose`
Si es verdadero, getmail registrará información sobre los mensajes que no se hayan podido recuperar y la razón para no hacerlo, así como líneas de inicio y fin informativas.

El valor predeterminado es '`#f`'

`parameter-alist` [parámetro de `getmail-options-configuration`]
`extra-parameters`
Opciones adicionales a incluir.

El valor predeterminado es '`'()`'.

`lista idle` [parámetro de `getmail-configuration`]
Una lista de bandejas de correo en las que getmail debe esperar en el servidor nuevas notificaciones de correo. Esto depende de que el servidor implemente la extensión IDLE.

El valor predeterminado es '`'()`'.

`lista environment-variables` [parámetro de `getmail-configuration`]
Variables de entorno proporcionadas a getmail.

El valor predeterminado es '`'()`'.

## Servicios de alias de correo

`mail-aliases-service-type` [Variable]

Este es el tipo del servicio que proporciona `/etc/aliases`, donde se especifica cómo entregar el correo a las usuarias de este sistema.

```
(service mail-aliases-service-type
         '(("postmaster" "rober")
           ("rober" "rober@example.com" "rober@example2.com")))
```

The configuration for a `mail-aliases-service-type` service is an association list denoting how to deliver mail that comes to this system. Each entry is of the form `(alias addresses ...)`, with `alias` specifying the local alias and `addresses` specifying where to deliver this user's mail.

The aliases aren't required to exist as users on the local system. In the above example, there doesn't need to be a `postmaster` entry in the `operating-system`'s `user-accounts` in order to deliver the `postmaster` mail to `bob` (which subsequently would deliver mail to `bob@example.com` and `bob@example2.com`).

## daemon de IMAP3 de GNU Mailutils

`imap4d-service-type` [Variable]

Es el tipo del daemon IMAP4 de GNU Mailutils (see Section "imap4d" in *GNU Mailutils Manual*), cuyo valor debe ser un objeto `imap4d-configuration` como en este ejemplo:

```
(service imap4d-service-type
         (imap4d-configuration
           (config-file (local-file "imap4d.conf"))))
```

`imap4d-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `imap4d`.

`package` (predeterminado: `mailutils`)
: El paquete que proporciona `imap4d`.

`config-file` (predeterminado: `%default-imap4d-config-file`)
: Objeto "tipo-archivo" con el archivo de configuración usado, de manera predeterminada escucha en el puerto TCP 143 de `localhost`. See Section "Conf-imap4d" in *GNU Mailutils Manual*, para más detalles.

## Radicale Service

`radicale-service-type` [Variable]

This is the type of the Radicale (`https://radicale.org`) CalDAV/CardDAV server whose value should be a `radicale-configuration`.

`radicale-configuration` [Data Type]

Data type representing the configuration of `radicale`.

`package` (default: `radicale`)
: The package that provides `radicale`.

`config-file` (default: `%default-radicale-config-file`)
: File-like object of the configuration file to use, by default it will listen on TCP port 5232 of `localhost` and use the `htpasswd` file at `/var/lib/radicale/users` with no (`plain`) encryption.

### Rspamd Service

`rspamd-service-type` [Variable]
: This is the type of the Rspamd (`https://rspamd.com/`) filtering system whose value should be a `rspamd-configuration`.

`rspamd-configuration` [Data Type]
: Available `rspamd-configuration` fields are:

`package` (default: `rspamd`) (type: file-like)
: The package that provides rspamd.

`config-file` (default: `%default-rspamd-config-file`) (type: file-like)
: File-like object of the configuration file to use. By default all workers are enabled except fuzzy and they are binded to their usual ports, e.g localhost:11334, localhost:11333 and so on

`local.d-files` (default: `()`) (type: directory-tree)
: Configuration files in local.d, provided as a list of two element lists where the first element is the filename and the second one is a file-like object. Settings in these files will be merged with the defaults.

`override.d-files` (default: `()`) (type: directory-tree)
: Configuration files in override.d, provided as a list of two element lists where the first element is the filename and the second one is a file-like object. Settings in these files will override the defaults.

`user` (default: `%default-rspamd-account`) (type: user-account)
: The user to run rspamd as.

`group` (default: `%default-rspamd-group`) (type: user-group)
: The group to run rspamd as.

`debug?` (default: `#f`) (type: boolean)
: Force debug output.

`insecure?` (default: `#f`) (type: boolean)
: Ignore running workers as privileged users.

`skip-template?` (default: `#f`) (type: boolean)
: Do not apply Jinja templates.

`shepherd-requirements` (default: `(loopback)`) (type: list-of-symbols)
: This is a list of symbols naming Shepherd services that this service will depend on.

## 11.10.14 Servicios de mensajería

El módulo `(gnu services messaging)` proporciona definiciones de servicios Guix para servicios de mensajería. Actualmente proporciona los siguientes servicios:

### Servicio Prosody

`prosody-service-type` [Variable]

Este es el tipo para el servidor de comunicaciones XMPP Prosody (`https://prosody.im`). Su valor debe ser un registro `prosody-configuration` como en este ejemplo:

```
(service prosody-service-type
         (prosody-configuration
          (modules-enabled (cons* "groups" "mam" %default-modules-enabled))
          (int-components
           (list
            (int-component-configuration
             (hostname "conference.ejemplo.net")
             (plugin "muc")
             (mod-muc (mod-muc-configuration)))))
          (virtualhosts
           (list
            (virtualhost-configuration
             (domain "ejemplo.net"))))))
```

Véase a continuación detalles acerca de `prosody-configuration`.

De manera predeterminada, Prosody no necesita demasiada configuración. Únicamente un campo `virtualhost` es necesario: especifica el dominio en el que se desea que Prosody proporcione el servicio.

Puede realizar varias comprobaciones preliminares sobre la configuración generada con la orden `prosodyctl check`.

Prosodyctl también le ayudará con la importación de certificados del directorio `letsencrypt` de modo que la usuaria `prosody` puede acceder a ellos. Véase `https://prosody.im/doc/letsencrypt`.

```
prosodyctl --root cert import /etc/certs
```

The available configuration parameters follow. Each parameter definition is preceded by its type; for example, '`string-list foo`' indicates that the `foo` parameter should be specified as a list of strings. Types starting with `maybe-` denote parameters that won't show up in `prosody.cfg.lua` when their value is left unspecified.

También existe la posibilidad de especificar la configuración como una cadena, por si tiene un archivo `prosody.cfg.lua` antiguo que desea transportar desde otro sistema; véase más detalles al final.

El tipo `file-object` designa o bien un objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167) o un nombre de archivo.

Los campos disponibles de `prosody-configuration` son:

`package prosody` [parámetro de `prosody-configuration`]

El paquete Prosody.

`nombre-archivo data-path` [parámetro de `prosody-configuration`]

Ruta del directorio de almacenamiento de datos de Prosody. Véase `https://prosody.im/doc/configure`. Su valor predeterminado es '`"/var/lib/prosody"`'.

`lista-file-object plugin-paths` [parámetro de `prosody-configuration`]
Additional plugin directories. They are searched in all the specified paths in order. See `https://prosody.im/doc/plugins_directory`. Defaults to ‘`'()`’.

`nombre-archivo certificates` [parámetro de `prosody-configuration`]
Cada máquina virtual y componente necesitan un certificado de manera que los clientes y servidores puedan verificar su identidad de manera segura. Prosody cargará de manera automática certificados/claves del directorio especificado aquí. Su valor predeterminado es ‘`"/etc/prosody/certs"`’.

`lista-string admins` [parámetro de `prosody-configuration`]
Es una lista de cuentas con permisos de administración en el servidor. Tenga en cuenta que debe crear las cuentas de manera separada. Véase `https://prosody.im/doc/admins` y `https://prosody.im/doc/creating_accounts`. Ejemplo: `(admins '("usuaria1@example.com" "usuaria2@example.net"))` Su valor predeterminado es ‘`'()`’.

`boolean use-libevent?` [parámetro de `prosody-configuration`]
Activa el uso de libevent para mejorar el rendimiento bajo altas cargas de trabajo. Véase `https://prosody.im/doc/libevent`. Su valor predeterminado es ‘`#f`’.

`lista-módulos modules-enabled` [parámetro de `prosody-configuration`]
La lista de módulos de Prosody cargada durante el arranque. Busca en el archivo de módulos `mod_nombremodulo.lua`, por lo que asegúrese de que también exista. La documentación de los módulos puede encontrarse en: `https://prosody.im/doc/modules`. Su valor predeterminado es ‘`'("roster" "saslauth" "tls" "dialback" "disco" "carbons" "private" "blocklist" "vcard" "version" "uptime" "time" "ping" "pep" "register" "admin_adhoc")`’.

`lista-string modules-disabled` [parámetro de `prosody-configuration`]
‘`"offline"`’, ‘`"c2s"`’ y ‘`"s2s"`’ se cargan de manera automática, pero puede desactivarlos si los añade a esta lista. Su valor predeterminado es ‘`'()`’.

`file-object groups-file` [parámetro de `prosody-configuration`]
Ruta a un archivo de texto donde se definan los grupos compartidos. Si esta ruta está vacía, ‘`mod_groups`’ no hace nada. Véase `https://prosody.im/doc/modules/mod_groups`. Su valor predeterminado es ‘`"/var/lib/prosody/sharedgroups.txt"`’.

`boolean allow-registration?` [parámetro de `prosody-configuration`]
Desactiva la creación de cuentas de manera predeterminada, por seguridad. Véase `https://prosody.im/doc/creating_accounts`. Su valor predeterminado es ‘`#f`’.

`maybe-ssl-configuration ssl` [parámetro de `prosody-configuration`]
Estas opciones de configuración están relacionadas con SSL/TLS. La mayor parte no se proporcionan para usar los valores predeterminados de Prosody. Si no entiende completamente estas opciones, no las añada a su configuración, es fácil aumentar la vulnerabilidad de su servidor si las usa. Véase `https://prosody.im/doc/advanced_ssl_config`.

Los campos disponibles de `ssl-configuration` son:

`maybe-string protocol` [parámetro de `ssl-configuration`]
Determina el inicio del protocolo usado (handshake).

`maybe-nombre-archivo key` [parámetro de `ssl-configuration`]
Ruta a su archivo de clave privada.

`maybe-nombre-archivo certificate` [parámetro de `ssl-configuration`]
Ruta al archivo de su certificado.

`file-object capath` [parámetro de `ssl-configuration`]
Ruta al directorio que contiene los certificados raíz en los que desea que Prosody confíe al verificar los certificados de servidores remotos. Su valor predeterminado es '`"/etc/ssl/certs"`'.

`maybe-file-object cafile` [parámetro de `ssl-configuration`]
Ruta al archivo que contiene los certificados raíz en los que desea que Prosody confíe. Es similar a `capath` pero con todos los certificados concatenados en el mismo archivo.

`maybe-lista-string verify` [parámetro de `ssl-configuration`]
Una lista de opciones de verificación (son de manera prácticamente directa las opciones de `set_verify()` de OpenSSL).

`maybe-lista-string options` [parámetro de `ssl-configuration`]
A list of general options relating to SSL/TLS. These map to OpenSSL's `set_options()`. For a full list of options available in LuaSec, see the LuaSec source.

`maybe-entero-no-negativo depth` [parámetro de `ssl-configuration`]
Cómo de larga puede ser la cadena de autoridades de certificación a comprobar cuando se busque un certificado de confianza.

`maybe-string ciphers` [parámetro de `ssl-configuration`]
Una cadena de algoritmos de cifrado de OpenSSL. Selecciona que algoritmos ofrecerá Prosody a los clientes, y en qué orden.

`maybe-nombre-archivo dhparam` [parámetro de `ssl-configuration`]
Una ruta a un archivo que contenga parámetros para el intercambio de claves Diffie-Hellman. Puede crear un archivo de este tipo con: `openssl dhparam -out /etc/prosody/certs/dh-2048.pem 2048`

`maybe-string curve` [parámetro de `ssl-configuration`]
Curva para el protocolo Diffie-Hellman de curva elíptica. El valor predeterminado de Prosody es '`"secp384r1"`'.

`maybe-string-list verifyext` [parámetro de `ssl-configuration`]
Una lista de opciones de verificación adicionales.

`maybe-string password` [parámetro de `ssl-configuration`]
Contraseña para claves privadas cifradas.

`boolean c2s-require-encryption?` [parámetro de `prosody-configuration`]
Determina si se fuerza que todas las conexiones cliente-servidor vayan cifradas o no. Véase `https://prosody.im/doc/modules/mod_tls`. Su valor predeterminado es '`#f`'.

`lista-string` [parámetro de `prosody-configuration`]
`disable-sasl-mechanisms`
Conjunto de mecanismos que no se ofrecerán nunca. Véase `https://prosody.im/doc/modules/mod_saslauth`. Su valor predeterminado es '`'("DIGEST-MD5")`'.

`string-list insecure-sasl-mechanisms` [`prosody-configuration` parameter]
Set of mechanisms that will not be offered on unencrypted connections. See `https://prosody.im/doc/modules/mod_saslauth`. Defaults to '`'("PLAIN" "LOGIN")`'.

`boolean s2s-require-encryption?` [parámetro de `prosody-configuration`]
Determina si se fuerza que todas las conexiones servidor-servidor vayan cifradas o no. Véase `https://prosody.im/doc/modules/mod_tls`. Su valor predeterminado es '`#f`'.

`boolean s2s-secure-auth?` [parámetro de `prosody-configuration`]
Determina si el cifrado y la identificación mediante certificado son necesarias. Esto proporciona una seguridad ideal, pero necesita que los servidores con los que se comunique permitan cifrado y tengan presentes certificados válidos en los que se tenga confianza. Véase `https://prosody.im/doc/s2s#security`. Su valor predeterminado es '`#f`'.

`lista-string s2s-insecure-domains` [parámetro de `prosody-configuration`]
Muchos servidores no permiten el cifrado o tienen certificados auto-firmados. Puede proporcionar aquí una lista de dominios que no necesitarán la identificación mediante certificado. Se identificarán mediante DNS. Véase `https://prosody.im/doc/s2s#security`. Su valor predeterminado es '`'()`'.

`lista-string s2s-secure-domains` [parámetro de `prosody-configuration`]
Aún en el caso de mantener `s2s-secure-auth?`, puede exigir certificados válidos para algunos dominios especificando una lista aquí. Véase `https://prosody.im/doc/s2s#security`. Su valor predeterminado es '`'()`'.

`string authentication` [parámetro de `prosody-configuration`]
Selecciona el motor de identificación usado. La implementación predeterminada almacena las contraseñas en texto claro y usa el almacenamiento de datos configurado en Prosody para los datos de identificación. Si no confía en su servidor le recomendamos que visite `https://prosody.im/doc/modules/mod_auth_internal_hashed` para obtener información sobre el motor de almacenamiento tras hash. Véase también `https://prosody.im/doc/authentication`. Su valor predeterminado es '`'"internal_plain"`'.

`maybe-string log` [parámetro de `prosody-configuration`]
Determina las opciones del registro. La configuración avanzada del registro no está implementada todavía para el servicio Prosody. Véase `https://prosody.im/doc/logging`. Su valor predeterminado es '`'"*syslog"`'.

`nombre-archivo pidfile` [parámetro de `prosody-configuration`]
Archivo en el que se escribirá el PID. Véase `https://prosody.im/doc/modules/mod_posix`. Su valor predeterminado es '`"/var/run/prosody/prosody.pid"`'.

`maybe-entero-no-negativo http-max-content-size` [parámetro de `prosody-configuration`]
Tamaño máximo permitido del cuerpo (body) HTTP (en bytes)

`maybe-string http-external-url` [parámetro de `prosody-configuration`]
Algunos módulos exponen sus propias URL de diversas maneras. Esta URL se construye en base al protocolo, máquina y puerto usados. Si Prosody se encuentra tras un proxy, se usara `http-external-url` como URL pública. Véase `https://prosody.im/doc/http#external_url`.

`lista-virtualhost-configuration virtualhosts` [parámetro de `prosody-configuration`]
Una máquina (host) en Prosody es un dominio en el que se pueden crear cuentas de usuaria. Por ejemplo, si desea que sus usuarias tengan direcciones como '`"juan.herrero@example.com"`' necesitará añadir una máquina '`"example.com"`'. Todas las opciones en esta lista son efectivas únicamente en esa máquina.

> **Nota:** The name *virtual* host is used in configuration to avoid confusion with the actual physical host that Prosody is installed on. A single Prosody instance can serve many domains, each one defined as a VirtualHost entry in Prosody's configuration. Conversely a server that hosts a single domain would have just one VirtualHost entry.
>
> Véase `https://prosody.im/doc/configure#virtual_host_settings`.

Los campos disponibles de `virtualhost-configuration` son:

all these `prosody-configuration` fields: `admins`, `use-libevent?`, `modules-enabled`, `modules-disabled`, `groups-file`, `allow-registration?`, `ssl`, `c2s-require-encryption?`, `disable-sasl-mechanisms`, `insecure-sasl-mechanisms`, `s2s-require-encryption?`, `s2s-secure-auth?`, `s2s-insecure-domains`, `s2s-secure-domains`, `authentication`, `log`, `http-max-content-size`, `http-external-url`, `raw-content`, plus:

`string domain` [parámetro de `virtualhost-configuration`]
Dominio en el que desea que Prosody proporcione servicio.

`lista-int-component-configuration int-components` [parámetro de `prosody-configuration`]
Los componentes son servicios adicionales en un servidor que están disponibles a los clientes, habitualmente en un subdominio del servidor principal (como por ejemplo '`"micomponente.example.com"`'). Algunos ejemplos de componentes pueden ser los servidores de salas de conversación, los directorios de usuarias o las pasarelas a otros protocolos.

Los componentes internos se implementan con módulos específicos de Prosody. Para añadir un componente interno, simplemente rellene el campo del nombre de máquina, y el módulo que desea usar para el componente.

Véase `https://prosody.im/doc/components`. Su valor predeterminado es '`'()`'.

Los campos disponibles de `int-component-configuration` son:

all these `prosody-configuration` fields: `admins`, `use-libevent?`, `modules-enabled`, `modules-disabled`, `groups-file`, `allow-registration?`, `ssl`, `c2s-require-encryption?`, `disable-sasl-mechanisms`, `insecure-sasl-mechanisms`, `s2s-require-encryption?`, `s2s-secure-auth?`, `s2s-insecure-domains`, `s2s-secure-domains`, `authentication`, `log`, `http-max-content-size`, `http-external-url`, `raw-content`, plus:

`string hostname` [parámetro de `int-component-configuration`]
Nombre de máquina del componente.

`string plugin` [parámetro de `int-component-configuration`]
Módulo que desea usar para el componente.

`maybe-mod-muc-configuration` [parámetro de `int-component-configuration`]
`mod-muc`
Multi-user chat (MUC) es el módulo de Prosody que permite la creación de salas de conversación/conferencias para usuarias XMPP.

Información general sobre la configuración y el uso de las salas de conversación multi-usuaria puede encontrarse en la documentación "Chatrooms" (`https://prosody.im/doc/chatrooms`), que debería leer si no conoce las salas de conversación de XMPP.

Véase también `https://prosody.im/doc/modules/mod_muc`.

Los campos disponibles de `mod-muc-configuration` son:

`string name` [parámetro de `mod-muc-configuration`]
El nombre devuelto en las respuestas de descubrimiento de servicios. Su valor predeterminado es '`"Prosody Chatrooms"`'.

`string-o-boolean` [parámetro de `mod-muc-configuration`]
`restrict-room-creation`
Si es '`#t`', únicamente se permitirá a las administradoras la creación de nuevas salas de conversación. En otro caso cualquiera puede crear una sala. El valor '`"local"`' restringe la creación a usuarias en el dominio superior del servicio. Por ejemplo '`usuaria@example.com`' puede crear grupos en '`rooms.example.com`'. El valor '`"admin"`' restringe el servicio a las administradoras únicamente. Su valor predeterminado es '`#f`'.

`entero-no-negativo` [parámetro de `mod-muc-configuration`]
`max-history-messages`
Número máximo de mensajes históricos que se enviarán a quien se acabe de unir a la sala. Su valor predeterminado es '`20`'.

`lista-ext-component-configuration` [parámetro de `prosody-configuration`]
`ext-components`
Los componentes externos usan XEP-0114, el cual se implementa en la mayor parte de componentes independientes. Para añadir un componente externo, simplemente

rellene el campo de nombre de máquina (hostname). Véase `httos://prosody.im/doc/components`. Su valor predeterminado es ‘`'()`’.

Los campos disponibles de `ext-component-configuration` son:

all these `prosody-configuration` fields: `admins`, `use-libevent?`, `modules-enabled`, `modules-disabled`, `groups-file`, `allow-registration?`, `ssl`, `c2s-require-encryption?`, `disable-sasl-mechanisms`, `insecure-sasl-mechanisms`, `s2s-require-encryption?`, `s2s-secure-auth?`, `s2s-insecure-domains`, `s2s-secure-domains`, `authentication`, `log`, `http-max-content-size`, `http-external-url`, `raw-content`, plus:

`string component-secret` [parámetro de `ext-component-configuration`]
Contraseña usada por el componente para el ingreso al sistema.

`string hostname` [parámetro de `ext-component-configuration`]
Nombre de máquina del componente.

`lista-entero-no-negativo component-ports` [parámetro de `prosody-configuration`]
Puerto o puertos en los que prosody escucha conexiones de componentes. Su valor predeterminado es ‘`'(5347)`’.

`string component-interface` [parámetro de `prosody-configuration`]
Interfaz en la que Prosody escucha conexiones de componentes. Su valor predeterminado es ‘`"127.0.0.1"`’.

`maybe-raw-content raw-content` [parámetro de `prosody-configuration`]
Contenido que se añadirá directamente al archivo de configuración.

Puede ser que únicamente desee usar un archivo `prosody.cfg.lua` ya creado. En ese caso, puede proporcionar un registro `opaque-prosody-configuration` como el valor de `prosody-service-type`. Como su nombre en inglés indica, una configuración opaca no tiene gran capacidad reflexiva. Los campos disponibles de `opaque-prosody-configuration` son:

`package prosody` [parámetro de `opaque-prosody-configuration`]
El paquete prosody.

`string prosody.cfg.lua` [parámetro de `opaque-prosody-configuration`]
El contenido usado para `prosody.cfg.lua`.

Por ejemplo, si su `prosody.cfg.lua` es simplemente la cadena vacía, podría instanciar el servicio de Prosody de esta manera:

```
(service prosody-service-type
         (opaque-prosody-configuration
          (prosody.cfg.lua "")))
```

## Servicio BitlBee

BitlBee (`https://bitlbee.org`) es una pasarela que proporciona una interfaz IRC a una variedad de protocolos como XMPP.

`bitlbee-service-type` [Variable]

Este es el tipo de servicio para el daemon de pasarela IRC BitlBee (`https://bitlbee.org`). Su valor es un `bitlbee-configuration` (véase a continuación).

Para que BitlBee escuche en el puerto 6667 de localhost, añada esta línea a sus servicios:

```
(service bitlbee-service-type)
```

`bitlbee-configuration` [Tipo de datos]

Esta es la configuración para BitlBee, con los siguientes campos:

`interface` (predeterminada: `"127.0.0.1"`)
`port` (predeterminado: `6667`)
: Escucha en la interfaz de red correspondiente a la dirección IP especificada en *interface*, en el puerto *port*.

  Cuando *interface* es `127.0.0.1`, únicamente se permite la conexión de clientes locales; cuando es `0.0.0.0`, las conexiones pueden venir de cualquier interfaz de red.

`bitlbee` (predeterminado: `bitlbee`)
: El paquete BitlBee usado.

`plugins` (predeterminados: `'()`)
: Lista de paquetes de módulos usados—por ejemplo, `bitlbee-discord`.

`extra-settings` (predeterminado: `""`)
: Fragmento de configuración añadido tal cual al archivo de configuración de BitlBee.

## Servicio Quassel

Quassel (`https://quassel-irc.org/`) es un cliente IRC distribuido, lo que significa que uno o más clientes se pueden conectar y desconectar del núcleo central.

`quassel-service-type` [Variable]

Es el tipo de servicio del daemon del motor IRC de Quassel (`https://quassel-irc.org/`). Su valor es un `quassel-configuration` (véase a continuación).

`quassel-configuration` [Tipo de datos]

Es la configuración para Quassel, con los siguientes campos:

`quassel` (predeterminado: `quassel`)
: El paquete Quassel usado.

`interface` (predeterminada: `"::,0.0.0.0"`)
`port` (predeterminado: `4242`)
: Escucha en la o las interfaces de red que correspondan con las direcciones IPv4 o IPv6 delimitadas por comas especificadas en *interface*, en el puerto *port*.

`loglevel` (predeterminado: `"Info"`)
: El nivel de registro deseado. Los valores aceptados son Debug, Info, Warning y Error.

### 11.10.15 Servicios de telefonía

The `(gnu services telephony)` module contains Guix service definitions for telephony services. Currently it provides the following services:

## Jami

`jami-service-type` [Variable]

The service type for running Jami as a service. It takes a `jami-configuration` object as a value, documented below. This section describes how to configure a Jami server that can be used to host video (or audio) conferences, among other uses. The following example demonstrates how to specify Jami account archives (backups) to be provisioned automatically:

```
(service jami-service-type
         (jami-configuration
          (accounts
           (list (jami-account
                  (archive "/etc/jami/unencrypted-account-1.gz"))
                 (jami-account
                  (archive "/etc/jami/unencrypted-account-2.gz"))))))
```

When the accounts field is specified, the Jami account files of the service found under `/var/lib/jami` are recreated every time the service starts.

Jami accounts and their corresponding backup archives can be generated using the `jami` or `jami-gnome` Jami clients. The accounts should not be password-protected, but it is wise to ensure their files are only readable by '`root`'.

The next example shows how to declare that only some contacts should be allowed to communicate with a given account:

```
(service jami-service-type
         (jami-configuration
          (accounts
           (list (jami-account
                  (archive "/etc/jami/unencrypted-account-1.gz")
                  (peer-discovery? #t)
                  (rendezvous-point? #t)
                  (allowed-contacts
                   '("1dbcb0f5f37324228235564b79f2b9737e9a008f"
                     "2dbcb0f5f37324228235564b79f2b9737e9a008f")))))))
```

In this mode, only the declared `allowed-contacts` can initiate communication with the Jami account. This can be used, for example, with rendezvous point accounts to create a private video conferencing space.

To put the system administrator in full control of the conferences hosted on their system, the Jami service supports the following actions:

```
# herd doc jami list-actions
```

```
(list-accounts
 list-account-details
 list-banned-contacts
 list-contacts
 list-moderators
 add-moderator
 ban-contact
 enable-account
 disable-account)
```

The above actions aim to provide the most valuable actions for moderation purposes, not to cover the whole Jami API. Users wanting to interact with the Jami daemon from Guile may be interested in experimenting with the `(gnu build jami-service)` module, which powers the above Shepherd actions.

The `add-moderator` and `ban-contact` actions accept a contact *fingerprint* (40 characters long hash) as first argument and an account fingerprint or username as second argument:

```
# herd add-moderator jami 1dbcb0f5f37324228235564b79f2b9737e9a008f \
  f3345f2775ddfe07a4b0d95daea111d15fbc1199

# herd list-moderators jami
Moderators for account f3345f2775ddfe07a4b0d95daea111d15fbc1199:
  - 1dbcb0f5f37324228235564b79f2b9737e9a008f
```

In the case of `ban-contact`, the second username argument is optional; when omitted, the account is banned from all Jami accounts:

```
# herd ban-contact jami 1dbcb0f5f37324228235564b79f2b9737e9a008f

# herd list-banned-contacts jami
Banned contacts for account f3345f2775ddfe07a4b0d95daea111d15fbc1199:
  - 1dbcb0f5f37324228235564b79f2b9737e9a008f
```

Banned contacts are also stripped from their moderation privileges.

The `disable-account` action allows to completely disconnect an account from the network, making it unreachable, while `enable-account` does the inverse. They accept a single account username or fingerprint as first argument:

```
# herd disable-account jami f3345f2775ddfe07a4b0d95daea111d15fbc1199

# herd list-accounts jami
The following Jami accounts are available:
  - f3345f2775ddfe07a4b0d95daea111d15fbc1199 (dummy) [disabled]
```

The `list-account-details` action prints the detailed parameters of each accounts in the Recutils format, which means the `recsel` command can be used to select accounts of interest (see Section “Selection Expressions” in *GNU recutils manual*). Note that period characters (‘`.`’) found in the account parameter keys are mapped

to underscores (‘_’) in the output, to meet the requirements of the Recutils format. The following example shows how to print the account fingerprints for all accounts operating in the rendezvous point mode:

```
# herd list-account-details jami | \
  recsel -p Account.username -e 'Account.rendezVous ~ "true"'
Account_username: f3345f2775ddfe07a4b0d95daea111d15fbc1199
```

The remaining actions should be self-explanatory.

The complete set of available configuration options is detailed below.

`jami-configuration` [Data Type]

Available `jami-configuration` fields are:

`libjami` (default: `libjami`) (type: package)
The Jami daemon package to use.

`dbus` (default: `dbus-for-jami`) (type: package)
The D-Bus package to use to start the required D-Bus session.

`nss-certs` (default: `nss-certs`) (type: package)
The nss-certs package to use to provide TLS certificates.

`enable-logging?` (default: `#t`) (type: boolean)
Whether to enable logging to syslog.

`debug?` (default: `#f`) (type: boolean)
Whether to enable debug level messages.

`auto-answer?` (default: `#f`) (type: boolean)
Whether to force automatic answer to incoming calls.

`accounts` (type: maybe-jami-account-list)
A list of Jami accounts to be (re-)provisioned every time the Jami daemon service starts. When providing this field, the account directories under `/var/lib/jami/` are recreated every time the service starts, ensuring a consistent state.

`jami-account` [Data Type]

Available `jami-account` fields are:

`archive` (type: string-or-computed-file)
The account archive (backup) file name of the account. This is used to provision the account when the service starts. The account archive should *not* be encrypted. It is highly recommended to make it readable only to the ‘`root`’ user (i.e., not in the store), to guard against leaking the secret key material of the Jami account it contains.

`allowed-contacts` (type: maybe-account-fingerprint-list)
The list of allowed contacts for the account, entered as their 40 characters long fingerprint. Messages or calls from accounts not in that list will be rejected. When left specified, the configuration of the account archive is used as-is with respect to contacts and public inbound calls/messaging allowance, which typically defaults to allow any contact to communicate with the account.

`moderators` (type: maybe-account-fingerprint-list)
: The list of contacts that should have moderation privileges (to ban, mute, etc. other users) in rendezvous conferences, entered as their 40 characters long fingerprint. When left unspecified, the configuration of the account archive is used as-is with respect to moderation, which typically defaults to allow anyone to moderate.

`rendezvous-point?` (type: maybe-boolean)
: Whether the account should operate in the rendezvous mode. In this mode, all the incoming audio/video calls are mixed into a conference. When left unspecified, the value from the account archive prevails.

`peer-discovery?` (type: maybe-boolean)
: Whether peer discovery should be enabled. Peer discovery is used to discover other OpenDHT nodes on the local network, which can be useful to maintain communication between devices on such network even when the connection to the Internet has been lost. When left unspecified, the value from the account archive prevails.

`bootstrap-hostnames` (type: maybe-list-of-strings)
: A list of hostnames or IPs pointing to OpenDHT nodes, that should be used to initially join the OpenDHT network. When left unspecified, the value from the account archive prevails.

`name-server-uri` (type: maybe-string)
: The URI of the name server to use, that can be used to retrieve the account fingerprint for a registered username.

## Mumble server

This section describes how to set up and run a Mumble (`https://mumble.info`) server (formerly known as Murmur).

`mumble-server-service-type` [Variable]
: This is the service to run a Mumble server. It takes a `mumble-server-configuration` object as its value, defined below.

`mumble-server-configuration` [Data Type]
: The service type for the Mumble server. An example configuration can look like this:

```
(service murmur-service-type
         (murmur-configuration
          (welcome-text
            "¡Bienvenida a este servidor Murmur que se ejecuta en Guix!")
          (cert-required? #t) ;no permite ingresos con una contraseña en texto
          (ssl-cert "/etc/certs/mumble.example.com/fullchain.pem")
          (ssl-key "/etc/certs/mumble.example.com/privkey.pem")))
```

: After reconfiguring your system, you can manually set the mumble-server `SuperUser` password with the command that is printed during the activation phase.

: Se recomienda el registro de una cuenta de usuaria normal de Mumble y la concesión de permisos de administración o moderación. Puede usar el cliente `mumble` para ingresar como una nueva usuaria normal, registrarse usted misma, y salir del sistema.

En el siguiente paso ingrese en el sistema con el nombre `SuperUser`, use la contraseña de `SuperUser` que fue establecida con anterioridad, y conceda los permisos de administración o moderación a su usuaria de nombre creada anteriormente y cree algunos canales.

Available `mumble-server-configuration` fields are:

`package` (predeterminado: `mumble`)
: Package that contains `bin/mumble-server`.

`user` (default: `"mumble-server"`)
: User who will run the Mumble-Server server.

`group` (default: `"mumble-server"`)
: Group of the user who will run the mumble-server server.

`port` (predeterminado: `64738`)
: Puerto en el que escucha el servidor.

`welcome-text` (predeterminado: `""`)
: Mensaje de bienvenida enviado a clientes tras su conexión.

`server-password` (predeterminada: `""`)
: Contraseña que debe introducirse para poder conectarse.

`max-users` (predeterminados: `100`)
: Número máximo de usuarias que pueden estar conectadas a la vez al servidor.

`max-user-bandwidth` (predeterminado: `#f`)
: Tráfico de voz máximo que una usuaria puede mandar por segundo.

`database-file` (default: `"/var/lib/mumble-server/db.sqlite"`)
: Nombre de archivo de la base de datos sqlite. La usuaria del servicio se convertirá en propietaria del directorio.

`log-file` (default: `"/var/log/mumble-server/mumble-server.log"`)
: Nombre de archivo del archivo de registro. La usuaria del servicio se convertirá en propietaria del directorio.

`autoban-attempts` (predeterminados: `10`)
: Número máximo de ingresos al sistema que una usuaria puede llevar a cabo en `autoban-timeframe` sin bloquearse su acceso durante `autoban-time`.

`autoban-timeframe` (predeterminado: `120`)
: Marco de tiempo del bloqueo automático en segundos.

`autoban-time` (predeterminado: `300`)
: Duración en segundos del periodo que permanecerá bloqueado un cliente cuando viole los límites de bloqueo automático.

`opus-threshold` (predeterminado: `100`)
: Porcentaje de clientes que tienen que permitir opus antes de cambiar al algoritmo de sonido opus.

`channel-nesting-limit` (predeterminado: `10`)
: Cual puede ser el nivel de recursión de los canales.

`channelname-regex` (predeterminado: `#f`)
: Una cadena en forma de expresión regular Qt que deben cumplir los nombres de canal.

`username-regex` (predeterminado: `#f`)
: Una cadena en forma de expresión regular Qt que deben cumplir los nombres de usuaria.

`text-message-length` (predeterminado: `5000`)
: Número máximo de bytes que una usuaria puede enviar en un mensaje de texto.

`image-message-length` (predeterminado: `(* 128 1024)`)
: Número máximo de bytes que una usuaria puede enviar en un mensaje de imagen.

`cert-required?` (predeterminado: `#f`)
: If it is set to `#t` clients that use weak password authentication will not be accepted. Users must have completed the certificate wizard to join.

`remember-channel?` (predeterminado: `#f`)
: Should mumble-server remember the last channel each user was in when they disconnected and put them into the remembered channel when they rejoin.

`allow-html?` (predeterminado: `#f`)
: Si se permite html en mensajes de texto, comentarios de usuaria y descripciones de canal.

`allow-ping?` (predeterminado: `#f`)
: Setting to true exposes the current user count, the maximum user count, and the server's maximum bandwidth per client to unauthenticated users. In the Mumble client, this information is shown in the Connect dialog.

  Desactivar esta opción impedirá la escucha pública en el servidor.

`bonjour?` (predeterminado: `#f`)
: Si el servidor debe anunciarse a sí mismo en la red local a través del protocolo "bonjour".

`send-version?` (predeterminado: `#f`)
: Should the mumble-server server version be exposed in ping requests.

`log-days` (predeterminado: `31`)
: Mumble also stores logs in the database, which are accessible via RPC. The default is 31 days of months, but you can set this setting to 0 to keep logs forever, or -1 to disable logging to the database.

`obfuscate-ips?` (predeterminado: `#t`)
: Si las IP registradas deben ofuscarse para proteger la privacidad de las usuarias.

`ssl-cert` (predeterminado: `#f`)
: Nombre del archivo del certificado SSL/TLS usado para conexiones cifradas.

```
(ssl-cert "/etc/certs/example.com/fullchain.pem")
```

`ssl-key` (predeterminada: `#f`)
: Ruta de archivo de la clave privada de ssl usada para las conexiones cifradas.

```
(ssl-key "/etc/certs/example.com/privkey.pem")
```

`ssl-dh-params` (predeterminado: `#f`)
: Nombre del archivo codificado con PEM con parámetros Diffie-Hellman para el cifrado SSL/TLS. De manera alternativa puede establecer su valor a `"@ffdhe2048"`, `"@ffdhe3072"`, `"@ffdhe4096"`, `"@ffdhe6144"` o `"@ffdhe8192"` para usar los parámetros contenidos en el RFC 7919.

`ssl-ciphers` (predeterminado: `#f`)
: La opción `ssl-ciphers` selecciona los protocolos de cifrado disponibles para su uso en SSL/TLS.

  Esta opción se especifica mediante el uso de la notación de listas de prot. de cifrado de OpenSSL (`https://www.openssl.org/docs/apps/ciphers.html#CIPHER-LIST-FORMAT`).

  It is recommended that you try your cipher string using 'openssl ciphers <string>' before setting it here, to get a feel for which cipher suites you will get. After setting this option, it is recommend that you inspect your Mumble server log to ensure that Mumble is using the cipher suites that you expected it to.

  > **Nota:** Changing this option may impact the backwards compatibility of your Mumble-Server server, and can remove the ability for older Mumble clients to be able to connect to it.

`public-registration` (predeterminado: `#f`)
: Must be a `<mumble-server-public-registration-configuration>` record or `#f`.

  Puede registrar de manera opcional su servidor en la lista pública de servidores que el cliente `mumble` muestra al inicio. No puede registrar su servidor si tiene establecida una contraseña para el servidor (`server-password`), o establece `allow-ping` como `#f`.

  Puede tomar algunas horas hasta que se muestre en la lista pública.

`file` (predeterminado: `#f`)
: Forma opcional alternativa de forzar el valor de esta configuración.

`mumble-server-public-registration-configuration` [Data Type]
Configuration for public registration of a mumble-server service.

`name` This is a display name for your server. Not to be confused with the hostname.

`password` A password to identify your registration. Subsequent updates will need the same password. Don't lose your password.

`url` Debe ser un enlace `http://` o `https://` a su página web.

`hostname` (predeterminado: `#f`)
De manera predeterminada su servidor se enumerará por sus direcciones IP. Si se usa esta opción, en vez de eso se enlazará a través de este nombre de máquina.

**Deprecation notice:** Due to historical reasons, all of the above `mumble-server-` procedures are also exported with the `murmur-` prefix. It is recommended that you switch to using `mumble-server-` going forward.

### 11.10.16 File-Sharing Services

The `(gnu services file-sharing)` module provides services that assist with transferring files over peer-to-peer file-sharing networks.

#### Transmission Daemon Service

Transmission (`https://transmissionbt.com/`) is a flexible BitTorrent client that offers a variety of graphical and command-line interfaces. A `transmission-daemon-service-type` service provides Transmission's headless variant, `transmission-daemon`, as a system service, allowing users to share files via BitTorrent even when they are not logged in.

`transmission-daemon-service-type` [Variable]
The service type for the Transmission Daemon BitTorrent client. Its value must be a `transmission-daemon-configuration` object as in this example:

```
(service transmission-daemon-service-type
         (transmission-daemon-configuration
          ;; Restrict access to the RPC ("control") interface
          (rpc-authentication-required? #t)
          (rpc-username "transmission")
          (rpc-password
           (transmission-password-hash
            "transmission" ; desired password
            "uKd1uMs9"))   ; arbitrary salt value

          ;; Accept requests from this and other hosts on the
          ;; local network
          (rpc-whitelist-enabled? #t)
          (rpc-whitelist '("::1" "127.0.0.1" "192.168.0.*"))

          ;; Limit bandwidth use during work hours
          (alt-speed-down (* 1024 2)) ;   2 MB/s
          (alt-speed-up 512)          ; 512 kB/s

          (alt-speed-time-enabled? #t)
          (alt-speed-time-day 'weekdays)
```

```
(alt-speed-time-begin
 (+ (* 60 8) 30))           ; 8:30 am
(alt-speed-time-end
 (+ (* 60 (+ 12 5)) 30))))  ; 5:30 pm
```

Once the service is started, users can interact with the daemon through its Web interface (at `http://localhost:9091/`) or by using the `transmission-remote` command-line tool, available in the `transmission` package. (Emacs users may want to also consider the `emacs-transmission` package.) Both communicate with the daemon through its remote procedure call (RPC) interface, which by default is available to all users on the system; you may wish to change this by assigning values to the `rpc-authentication-required?`, `rpc-username` and `rpc-password` settings, as shown in the example above and documented further below.

The value for `rpc-password` must be a password hash of the type generated and used by Transmission clients. This can be copied verbatim from an existing `settings.json` file, if another Transmission client is already being used. Otherwise, the `transmission-password-hash` and `transmission-random-salt` procedures provided by this module can be used to obtain a suitable hash value.

`transmission-password-hash` *password salt* [Procedure]

Returns a string containing the result of hashing *password* together with *salt*, in the format recognized by Transmission clients for their `rpc-password` configuration setting.

*salt* must be an eight-character string. The `transmission-random-salt` procedure can be used to generate a suitable salt value at random.

`transmission-random-salt` [Procedure]

Returns a string containing a random, eight-character salt value of the type generated and used by Transmission clients, suitable for passing to the `transmission-password-hash` procedure.

These procedures are accessible from within a Guile REPL started with the `guix repl` command (see Section 8.13 [Invocación de guix repl], page 177). This is useful for obtaining a random salt value to provide as the second parameter to ‘transmission-password-hash‘, as in this example session:

```
$ guix repl
scheme@(guix-user)> ,use (gnu services file-sharing)
scheme@(guix-user)> (transmission-random-salt)
$1 = "uKd1uMs9"
```

Alternatively, a complete password hash can generated in a single step:

```
scheme@(guix-user)> (transmission-password-hash "transmission"
(transmission-random-salt))
$2 = "{c8bbc6d1740cd8dc819a6e25563b67812c1c19c9VtFPfdsX"
```

The resulting string can be used as-is for the value of `rpc-password`, allowing the password to be kept hidden even in the operating-system configuration.

Torrent files downloaded by the daemon are directly accessible only to users in the “transmission” user group, who receive read-only access to the directory specified by the `download-dir` configuration setting (and also the directory specified by `incomplete-dir`, if

`incomplete-dir-enabled?` is `#t`). Downloaded files can be moved to another directory or deleted altogether using `transmission-remote` with its `--move` and `--remove-and-delete` options.

If the `watch-dir-enabled?` setting is set to `#t`, users in the "transmission" group are able also to place `.torrent` files in the directory specified by `watch-dir` to have the corresponding torrents added by the daemon. (The `trash-original-torrent-files?` setting controls whether the daemon deletes these files after processing them.)

Some of the daemon's configuration settings can be changed temporarily by `transmission-remote` and similar tools. To undo these changes, use the service's `reload` action to have the daemon reload its settings from disk:

```
# herd reload transmission-daemon
```

The full set of available configuration settings is defined by the `transmission-daemon-configuration` data type.

`transmission-daemon-configuration` [Data Type]
The data type representing configuration settings for Transmission Daemon. These correspond directly to the settings recognized by Transmission clients in their `settings.json` file.

Available `transmission-daemon-configuration` fields are:

`package transmission` [`transmission-daemon-configuration` parameter]
The Transmission package to use.

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`stop-wait-period`
The period, in seconds, to wait when stopping the service for `transmission-daemon` to exit before killing its process. This allows the daemon time to complete its housekeeping and send a final update to trackers as it shuts down. On slow hosts, or hosts with a slow network connection, this value may need to be increased.

El valor predeterminado es '`10`'.

`string download-dir` [`transmission-daemon-configuration` parameter]
The directory to which torrent files are downloaded.

Defaults to '`"/var/lib/transmission-daemon/downloads"`'.

`boolean` [`transmission-daemon-configuration` parameter]
`incomplete-dir-enabled?`
If `#t`, files will be held in `incomplete-dir` while their torrent is being downloaded, then moved to `download-dir` once the torrent is complete. Otherwise, files for all torrents (including those still being downloaded) will be placed in `download-dir`.

El valor predeterminado es '`#f`'

`maybe-string` [`transmission-daemon-configuration` parameter]
`incomplete-dir`
The directory in which files from incompletely downloaded torrents will be held when `incomplete-dir-enabled?` is `#t`.

El valor predeterminado es '`disabled`'.

`umask umask` [`transmission-daemon-configuration` parameter]
The file mode creation mask used for downloaded files. (See the `umask` man page for more information.)

Defaults to '`18`'.

`boolean rename-partial-files?` [`transmission-daemon-configuration` parameter]
When `#t`, ".part" is appended to the name of partially downloaded files.

El valor predeterminado es '`#t`'

`preallocation-mode preallocation` [`transmission-daemon-configuration` parameter]
The mode by which space should be preallocated for downloaded files, one of `none`, `fast` (or `sparse`) and `full`. Specifying `full` will minimize disk fragmentation at a cost to file-creation speed.

Defaults to '`fast`'.

`boolean watch-dir-enabled?` [`transmission-daemon-configuration` parameter]
If `#t`, the directory specified by `watch-dir` will be watched for new `.torrent` files and the torrents they describe added automatically (and the original files removed, if `trash-original-torrent-files?` is `#t`).

El valor predeterminado es '`#f`'

`maybe-string watch-dir` [`transmission-daemon-configuration` parameter]
The directory to be watched for `.torrent` files indicating new torrents to be added, when `watch-dir-enabled` is `#t`.

El valor predeterminado es '`disabled`'.

`boolean trash-original-torrent-files?` [`transmission-daemon-configuration` parameter]
When `#t`, `.torrent` files will be deleted from the watch directory once their torrent has been added (see `watch-directory-enabled?`).

El valor predeterminado es '`#f`'

`boolean speed-limit-down-enabled?` [`transmission-daemon-configuration` parameter]
When `#t`, the daemon's download speed will be limited to the rate specified by `speed-limit-down`.

El valor predeterminado es '`#f`'

`non-negative-integer speed-limit-down` [`transmission-daemon-configuration` parameter]
The default global-maximum download speed, in kilobytes per second.

El valor predeterminado es '`100`'.

`boolean speed-limit-up-enabled?` [`transmission-daemon-configuration` parameter]
When `#t`, the daemon's upload speed will be limited to the rate specified by `speed-limit-up`.

El valor predeterminado es '`#f`'

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`speed-limit-up`

The default global-maximum upload speed, in kilobytes per second.

El valor predeterminado es '`100`'.

`boolean alt-speed-enabled?` [`transmission-daemon-configuration` parameter]

When `#t`, the alternate speed limits `alt-speed-down` and `alt-speed-up` are used (in place of `speed-limit-down` and `speed-limit-up`, if they are enabled) to constrain the daemon's bandwidth usage. This can be scheduled to occur automatically at certain times during the week; see `alt-speed-time-enabled?`.

El valor predeterminado es '`#f`'

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`alt-speed-down`

The alternate global-maximum download speed, in kilobytes per second.

El valor predeterminado es '`50`'.

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`alt-speed-up`

The alternate global-maximum upload speed, in kilobytes per second.

El valor predeterminado es '`50`'.

`boolean` [`transmission-daemon-configuration` parameter]
`alt-speed-time-enabled?`

When `#t`, the alternate speed limits `alt-speed-down` and `alt-speed-up` will be enabled automatically during the periods specified by `alt-speed-time-day`, `alt-speed-time-begin` and `alt-time-speed-end`.

El valor predeterminado es '`#f`'

`day-list` [`transmission-daemon-configuration` parameter]
`alt-speed-time-day`

The days of the week on which the alternate-speed schedule should be used, specified either as a list of days (`sunday`, `monday`, and so on) or using one of the symbols `weekdays`, `weekends` or `all`.

Defaults to '`all`'.

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`alt-speed-time-begin`

The time of day at which to enable the alternate speed limits, expressed as a number of minutes since midnight.

Defaults to '`540`'.

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`alt-speed-time-end`

The time of day at which to disable the alternate speed limits, expressed as a number of minutes since midnight.

Defaults to '`1020`'.

`string bind-address-ipv4` [`transmission-daemon-configuration` parameter]
The IP address at which to listen for peer connections, or "0.0.0.0" to listen at all available IP addresses.

El valor predeterminado es '`"0.0.0.0"`'.

`string bind-address-ipv6` [`transmission-daemon-configuration` parameter]
The IPv6 address at which to listen for peer connections, or "::" to listen at all available IPv6 addresses.

Defaults to '`"::"`'.

`boolean peer-port-random-on-start?` [`transmission-daemon-configuration` parameter]
If `#t`, when the daemon starts it will select a port at random on which to listen for peer connections, from the range specified (inclusively) by `peer-port-random-low` and `peer-port-random-high`. Otherwise, it listens on the port specified by `peer-port`.

El valor predeterminado es '`#f`'

`port-number peer-port-random-low` [`transmission-daemon-configuration` parameter]
The lowest selectable port number when `peer-port-random-on-start?` is `#t`.

Defaults to '`49152`'.

`port-number peer-port-random-high` [`transmission-daemon-configuration` parameter]
The highest selectable port number when `peer-port-random-on-start` is `#t`.

Defaults to '`65535`'.

`port-number peer-port` [`transmission-daemon-configuration` parameter]
The port on which to listen for peer connections when `peer-port-random-on-start?` is `#f`.

Defaults to '`51413`'.

`boolean port-forwarding-enabled?` [`transmission-daemon-configuration` parameter]
If `#t`, the daemon will attempt to configure port-forwarding on an upstream gateway automatically using UPnP and NAT-PMP.

El valor predeterminado es '`#t`'

`encryption-mode encryption` [`transmission-daemon-configuration` parameter]
The encryption preference for peer connections, one of `prefer-unencrypted-connections`, `prefer-encrypted-connections` or `require-encrypted-connections`.

Defaults to '`prefer-encrypted-connections`'.

`maybe-string` [`transmission-daemon-configuration` parameter]
`peer-congestion-algorithm`

The TCP congestion-control algorithm to use for peer connections, specified using a string recognized by the operating system in calls to `setsockopt`. When left unspecified, the operating-system default is used.

Note that on GNU/Linux systems, the kernel must be configured to allow processes to use a congestion-control algorithm not in the default set; otherwise, it will deny these requests with "Operation not permitted". To see which algorithms are available on your system and which are currently permitted for use, look at the contents of the files `tcp_available_congestion_control` and `tcp_allowed_congestion_control` in the `/proc/sys/net/ipv4` directory.

As an example, to have Transmission Daemon use the TCP Low Priority congestion-control algorithm (`http://www-ece.rice.edu/networks/TCP-LP/`), you'll need to modify your kernel configuration to build in support for the algorithm, then update your operating-system configuration to allow its use by adding a `sysctl-service-type` service (or updating the existing one's configuration) with lines like the following:

```
(service sysctl-service-type
         (sysctl-configuration
          (settings
           ("net.ipv4.tcp_allowed_congestion_control" .
            "reno cubic lp"))))
```

The Transmission Daemon configuration can then be updated with

```
(peer-congestion-algorithm "lp")
```

and the system reconfigured to have the changes take effect.

El valor predeterminado es '`disabled`'.

`tcp-type-of-service` [`transmission-daemon-configuration` parameter]
`peer-socket-tos`

The type of service to request in outgoing TCP packets, one of `default`, `low-cost`, `throughput`, `low-delay` and `reliability`.

Defaults to '`default`'.

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`peer-limit-global`

The global limit on the number of connected peers.

Defaults to '`200`'.

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`peer-limit-per-torrent`

The per-torrent limit on the number of connected peers.

El valor predeterminado es '`50`'.

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`upload-slots-per-torrent`

The maximum number of peers to which the daemon will upload data simultaneously for each torrent.

Defaults to '`14`'.

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`peer-id-ttl-hours`

The maximum lifespan, in hours, of the peer ID associated with each public torrent before it is regenerated.

Defaults to '`6`'.

`boolean blocklist-enabled?` [`transmission-daemon-configuration` parameter]

When `#t`, the daemon will ignore peers mentioned in the blocklist it has most recently downloaded from `blocklist-url`.

El valor predeterminado es '`#f`'

`maybe-string blocklist-url` [`transmission-daemon-configuration` parameter]

The URL of a peer blocklist (in P2P-plaintext or eMule `.dat` format) to be periodically downloaded and applied when `blocklist-enabled?` is `#t`.

El valor predeterminado es '`disabled`'.

`boolean` [`transmission-daemon-configuration` parameter]
`download-queue-enabled?`

If `#t`, the daemon will be limited to downloading at most `download-queue-size` non-stalled torrents simultaneously.

El valor predeterminado es '`#t`'

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`download-queue-size`

The size of the daemon's download queue, which limits the number of non-stalled torrents it will download at any one time when `download-queue-enabled?` is `#t`.

El valor predeterminado es '`5`'.

`boolean` [`transmission-daemon-configuration` parameter]
`seed-queue-enabled?`

If `#t`, the daemon will be limited to seeding at most `seed-queue-size` non-stalled torrents simultaneously.

El valor predeterminado es '`#f`'

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`seed-queue-size`

The size of the daemon's seed queue, which limits the number of non-stalled torrents it will seed at any one time when `seed-queue-enabled?` is `#t`.

El valor predeterminado es '`10`'.

`boolean` [`transmission-daemon-configuration` parameter]
`queue-stalled-enabled?`

When `#t`, the daemon will consider torrents for which it has not shared data in the past `queue-stalled-minutes` minutes to be stalled and not count them against its `download-queue-size` and `seed-queue-size` limits.

El valor predeterminado es '`#t`'

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`queue-stalled-minutes`

The maximum period, in minutes, a torrent may be idle before it is considered to be stalled, when `queue-stalled-enabled?` is `#t`.

El valor predeterminado es '`30`'.

`boolean` [`transmission-daemon-configuration` parameter]
`ratio-limit-enabled?`

When `#t`, a torrent being seeded will automatically be paused once it reaches the ratio specified by `ratio-limit`.

El valor predeterminado es '`#f`'

`non-negative-rational` [`transmission-daemon-configuration` parameter]
`ratio-limit`

The ratio at which a torrent being seeded will be paused, when `ratio-limit-enabled?` is `#t`.

Defaults to '`2.0`'.

`boolean` [`transmission-daemon-configuration` parameter]
`idle-seeding-limit-enabled?`

When `#t`, a torrent being seeded will automatically be paused once it has been idle for `idle-seeding-limit` minutes.

El valor predeterminado es '`#f`'

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`idle-seeding-limit`

The maximum period, in minutes, a torrent being seeded may be idle before it is paused, when `idle-seeding-limit-enabled?` is `#t`.

El valor predeterminado es '`30`'.

`boolean dht-enabled?` [`transmission-daemon-configuration` parameter]

Enable the distributed hash table (DHT) protocol (`http://bittorrent.org/beps/bep_0005.html`), which supports the use of trackerless torrents.

El valor predeterminado es '`#t`'

`boolean lpd-enabled?` [`transmission-daemon-configuration` parameter]

Enable local peer discovery (`https://en.wikipedia.org/wiki/Local_Peer_Discovery`) (LPD), which allows the discovery of peers on the local network and may reduce the amount of data sent over the public Internet.

El valor predeterminado es '`#f`'

`boolean pex-enabled?` [`transmission-daemon-configuration` parameter]

Enable peer exchange (`https://en.wikipedia.org/wiki/Peer_exchange`) (PEX), which reduces the daemon's reliance on external trackers and may improve its performance.

El valor predeterminado es '`#t`'

`boolean utp-enabled?` [`transmission-daemon-configuration` parameter]
Enable the micro transport protocol (`http://bittorrent.org/beps/bep_0029.html`) (uTP), which aims to reduce the impact of BitTorrent traffic on other users of the local network while maintaining full utilization of the available bandwidth.

El valor predeterminado es '`#t`'

`boolean rpc-enabled?` [`transmission-daemon-configuration` parameter]
If `#t`, enable the remote procedure call (RPC) interface, which allows remote control of the daemon via its Web interface, the `transmission-remote` command-line client, and similar tools.

El valor predeterminado es '`#t`'

`string rpc-bind-address` [`transmission-daemon-configuration` parameter]
The IP address at which to listen for RPC connections, or "0.0.0.0" to listen at all available IP addresses.

El valor predeterminado es '`"0.0.0.0"`'.

`port-number rpc-port` [`transmission-daemon-configuration` parameter]
The port on which to listen for RPC connections.

Defaults to '`9091`'.

`string rpc-url` [`transmission-daemon-configuration` parameter]
The path prefix to use in the RPC-endpoint URL.

Defaults to '`"/transmission/"`'.

`boolean rpc-authentication-required?` [`transmission-daemon-configuration` parameter]
When `#t`, clients must authenticate (see `rpc-username` and `rpc-password`) when using the RPC interface. Note this has the side effect of disabling host-name whitelisting (see `rpc-host-whitelist-enabled?`.

El valor predeterminado es '`#f`'

`maybe-string rpc-username` [`transmission-daemon-configuration` parameter]
The username required by clients to access the RPC interface when `rpc-authentication-required?` is `#t`.

El valor predeterminado es '`disabled`'.

`maybe-transmission-password-hash rpc-password` [`transmission-daemon-configuration` parameter]
The password required by clients to access the RPC interface when `rpc-authentication-required?` is `#t`. This must be specified using a password hash in the format recognized by Transmission clients, either copied from an existing `settings.json` file or generated using the `transmission-password-hash` procedure.

El valor predeterminado es '`disabled`'.

`boolean` [`transmission-daemon-configuration` parameter]
`rpc-whitelist-enabled?`

When `#t`, RPC requests will be accepted only when they originate from an address specified in `rpc-whitelist`.

El valor predeterminado es '`#t`'

`string-list rpc-whitelist` [`transmission-daemon-configuration` parameter]

The list of IP and IPv6 addresses from which RPC requests will be accepted when `rpc-whitelist-enabled?` is `#t`. Wildcards may be specified using '`*`'.

Defaults to '`'("127.0.0.1" "::1")`'.

`boolean` [`transmission-daemon-configuration` parameter]
`rpc-host-whitelist-enabled?`

When `#t`, RPC requests will be accepted only when they are addressed to a host named in `rpc-host-whitelist`. Note that requests to "localhost" or "localhost.", or to a numeric address, are always accepted regardless of these settings.

Note also this functionality is disabled when `rpc-authentication-required?` is `#t`.

El valor predeterminado es '`#t`'

`string-list` [`transmission-daemon-configuration` parameter]
`rpc-host-whitelist`

The list of host names recognized by the RPC server when `rpc-host-whitelist-enabled?` is `#t`.

El valor predeterminado es '`'()`'.

`message-level` [`transmission-daemon-configuration` parameter]
`message-level`

The minimum severity level of messages to be logged (to `/var/log/transmission.log`) by the daemon, one of `none` (no logging), `error`, `info` and `debug`.

El valor predeterminado es '`info`'

`boolean` [`transmission-daemon-configuration` parameter]
`start-added-torrents?`

When `#t`, torrents are started as soon as they are added; otherwise, they are added in "paused" state.

El valor predeterminado es '`#t`'

`boolean` [`transmission-daemon-configuration` parameter]
`script-torrent-done-enabled?`

When `#t`, the script specified by `script-torrent-done-filename` will be invoked each time a torrent completes.

El valor predeterminado es '`#f`'

`maybe-file-object` [`transmission-daemon-configuration` parameter]
`script-torrent-done-filename`

A file name or file-like object specifying a script to run each time a torrent completes, when `script-torrent-done-enabled?` is `#t`.

El valor predeterminado es '`disabled`'.

`boolean` [`transmission-daemon-configuration` parameter]
`scrape-paused-torrents-enabled?`

When `#t`, the daemon will scrape trackers for a torrent even when the torrent is paused.

El valor predeterminado es '`#t`'

`non-negative-integer` [`transmission-daemon-configuration` parameter]
`cache-size-mb`

The amount of memory, in megabytes, to allocate for the daemon's in-memory cache. A larger value may increase performance by reducing the frequency of disk I/O.

Defaults to '`4`'.

`boolean prefetch-enabled?` [`transmission-daemon-configuration` parameter]

When `#t`, the daemon will try to improve I/O performance by hinting to the operating system which data is likely to be read next from disk to satisfy requests from peers.

El valor predeterminado es '`#t`'

### 11.10.17 Servicios de monitorización

#### Servicio Tailon

Tailon (`https://tailon.readthedocs.io/`) es una aplicación web para la visualización y búsqueda en archivos de registro.

El ejemplo siguiente configura el servicio con los valores predeterminados. Por omisión, se puede acceder a Tailon en el puerto 8080 (`http://localhost:8080`).

```
(service tailon-service-type)
```

El ejemplo siguiente personaliza más la configuración de Tailon, añadiendo `sed` a la lista de órdenes permitidas.

```
(service tailon-service-type
         (tailon-configuration
           (config-file
             (tailon-configuration-file
               (allowed-commands '("tail" "grep" "awk" "sed"))))))
```

`tailon-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de Tailon. Este tipo tiene los siguientes parámetros:

`config-file` (predeterminado: `(tailon-configuration-file)`)

The configuration file to use for Tailon. This can be set to a *tailon-configuration-file* record value, or any gexp (see Section 8.12 [Expresiones-G], page 167).

Por ejemplo, para usar un archivo local, se puede usar la función `local-file`:

```
(service tailon-service-type
         (tailon-configuration
           (config-file (local-file "./mi-tailon.conf"))))
```

`package` (predeterminado: `tailon`)
: El paquete tailon usado.

`tailon-configuration-file` [Tipo de datos]
Tipo de datos que representa las opciones de configuración de Tailon. Este tipo tiene los siguientes parámetros:

`files` (predeterminados: `(list "/var/log")`)
: List of files to display. The list can include strings for a single file or directory, or a list, where the first item is the name of a subsection, and the remaining items are the files or directories in that subsection.

`bind` (predeterminado: `"localhost:8080"`)
: Dirección y puerto al que Tailon debe asociarse.

`relative-root` (predeterminado: `#f`)
: Ruta URL usada por Tailon, use `#f` para no usar una ruta.

`allow-transfers?` (predeterminado: `#t`)
: Permite la descarga de archivos de registro en la interfaz web.

`follow-names?` (predeterminado: `#t`)
: Permite la lectura de archivos todavía no existentes.

`tail-lines` (predeterminado: `200`)
: Número de líneas a leer inicialmente de cada archivo.

`allowed-commands` (predeterminadas: `(list "tail" "grep" "awk")`)
: Commands to allow running. By default, `sed` is disabled.

`debug?` (predeterminado: `#f`)
: Proporcione el valor `#t` en `debug?` para mostrar mensajes de depuración.

`wrap-lines` (predeterminado: `#t`)
: Initial line wrapping state in the web interface. Set to `#t` to initially wrap lines (the default), or to `#f` to initially not wrap lines.

`http-auth` (predeterminado: `#f`)
: HTTP authentication type to use. Set to `#f` to disable authentication (the default). Supported values are `"digest"` or `"basic"`.

`users` (predeterminado: `#f`)
: If HTTP authentication is enabled (see `http-auth`), access will be restricted to the credentials provided here. To configure users, use a list of pairs, where the first element of the pair is the username, and the 2nd element of the pair is the password.

```
(tailon-configuration-file
  (http-auth "basic")
  (users     '(("usuaria1" . "contraseña1")
               ("usuaria2" . "contraseña2"))))
```

### Servicio Darkstat

Darkstat es un programa de interceptación de paquetes que captura el tráfico de la red, calcula estadísticas sobre su uso y proporciona informes a través de HTTP.

`darkstat-service-type` [Variable]

Este es el tipo de servicio del servicio darkstat (`https://unix4lyfe.org/darkstat/`), su valor debe ser un registro `darkstat-configuration` como en este ejemplo:

```
(service darkstat-service-type
         (darkstat-configuration
           (interface "eno1")))
```

`darkstat-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `darkstat`.

`package` (predeterminado: `darkstat`)
El paquete darkstat usado.

`interface`
Captura el tráfico en la interfaz de red especificada.

`port` (predeterminado: `"667"`)
Asocia la interfaz web al puerto especificado.

`bind-address` (predeterminada: `"127.0.0.1"`)
Asocia la interfaz web a la dirección especificada.

`base` (predeterminada: `"/"`)
Specify the path of the base URL. This can be useful if `darkstat` is accessed via a reverse proxy.

### Servicio del exportador de nodos Prometheus

El "exportador de nodos" Prometheus pone a disposición del sistema de monitorización Prometheus las estadísticas de hardware y el sistema operativo proporcionadas por el núcleo Linux. Este servicio debe desplegarse en todos los nodos físicos y máquinas virtuales, donde la monitorización de estas estadísticas sea deseable.

`prometheus-node-exporter-service-type` [Variable]

This is the service type for the prometheus-node-exporter (`https://github.com/prometheus/node_exporter/`) service, its value must be a `prometheus-node-exporter-configuration`.

```
(service prometheus-node-exporter-service-type)
```

`prometheus-node-exporter-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `node_exporter`.

`package` (predeterminado: `go-github-com-prometheus-node-exporter`)
El paquete prometheus-node-exporter usado.

`web-listen-address` (predeterminada: `":9100"`)
Asocia la interfaz web a la dirección especificada.

`textfile-directory` (default: `"/var/lib/prometheus/node-exporter"`)
: This directory can be used to export metrics specific to this machine. Files containing metrics in the text format, with the filename ending in `.prom` should be placed in this directory.

`extra-options` (predeterminadas: `'()`)
: Extra options to pass to the Prometheus node exporter.

## vnStat Network Traffic Monitor

vnStat is a network traffic monitor that uses interface statistics provided by the kernel rather than traffic sniffing. This makes it a light resource monitor, regardless of network traffic rate.

`vnstat-service-type` [Variable]

This is the service type for the vnStat (`https://humdi.net/vnstat/`) daemon and accepts a `vnstat-configuration` value.

The following example will configure the service with default values:

```
(service vnstat-service-type)
```

`vnstat-configuration` [Data Type]

Available `vnstat-configuration` fields are:

`package` (default: `vnstat`) (type: file-like)
: The vnstat package.

`database-directory` (default: `"/var/lib/vnstat"`) (type: string)
: Specifies the directory where the database is to be stored. A full path must be given and a leading '/' isn't required.

`5-minute-hours` (default: `48`) (type: maybe-integer)
: Data retention duration for the 5 minute resolution entries. The configuration defines for how many past hours entries will be stored. Set to `-1` for unlimited entries or to `0` to disable the data collection of this resolution.

`64bit-interface-counters` (default: `-2`) (type: maybe-integer)
: Select interface counter handling. Set to `1` for defining that all interfaces use 64-bit counters on the kernel side and `0` for defining 32-bit counter. Set to `-1` for using the old style logic used in earlier versions where counter values within 32-bits are assumed to be 32-bit and anything larger is assumed to be a 64-bit counter. This may produce false results if a 64-bit counter is reset within the 32-bits. Set to `-2` for using automatic detection based on available kernel datastructures.

`always-add-new-interfaces?` (default: `#t`) (type: maybe-boolean)
: Enable or disable automatic creation of new database entries for interfaces not currently in the database even if the database file already exists when the daemon is started. New database entries will also get created for new interfaces seen while the daemon is running. Pseudo interfaces '`lo`', '`lo0`' and '`sit0`' are always excluded from getting added.

`bandwidth-detection?` (default: `#t`) (type: maybe-boolean)
: Try to automatically detect *max-bandwidth* value for each monitored interface. Mostly only ethernet interfaces support this feature. *max-bandwidth* will be used as fallback value if detection fails. Any interface specific *max-BW* configuration will disable the detection for the specified interface. In Linux, the detection is disabled for tun interfaces due to the Linux kernel always reporting 10 Mbit regardless of the used real interface.

`bandwidth-detection-interval` (default: `5`) (type: maybe-integer)
: How often in minutes interface specific detection of *max-bandwidth* is done for detecting possible changes when *bandwidth-detection* is enabled. Can be disabled by setting to `0`. Value range: '`0`'..'`30`'

`boot-variation` (default: `15`) (type: maybe-integer)
: Time in seconds how much the boot time reported by system kernel can variate between updates. Value range: '`0`'..'`300`'

`check-disk-space?` (default: `#t`) (type: maybe-boolean)
: Enable or disable the availability check of at least some free disk space before a database write.

`create-directories?` (default: `#t`) (type: maybe-boolean)
: Enable or disable the creation of directories when a configured path doesn't exist. This includes *database-directory*.

`daemon-group` (type: maybe-user-group)
: Specify the group to which the daemon process should switch during startup. Set to `%unset-value` to disable group switching.

`daemon-user` (type: maybe-user-account)
: Specify the user to which the daemon process should switch during startup. Set to `%unset-value` to disable user switching.

`daily-days` (default: `62`) (type: maybe-integer)
: Data retention duration for the one day resolution entries. The configuration defines for how many past days entries will be stored. Set to `-1` for unlimited entries or to `0` to disable the data collection of this resolution.

`database-synchronous` (default: `-1`) (type: maybe-integer)
: Change the setting of the SQLite `"synchronous"` flag which controls how much care is taken to ensure disk writes have fully completed when writing data to the database before continuing other actions. Higher values take extra steps to ensure data safety at the cost of slower performance. A value of `0` will result in all handling being left to the filesystem itself. Set to `-1` to select the default value according to database mode controlled by *database-write-ahead-logging* setting. See SQLite documentation for more details regarding values from `1` to `3`. Value range: '`-1`'..'`3`'

`database-write-ahead-logging?` (default: `#f`) (type: maybe-boolean)
: Enable or disable SQLite Write-Ahead Logging mode for the database. See SQLite documentation for more details and note that support for read-only operations isn't available in older SQLite versions.

`hourly-days` (default: `4`) (type: maybe-integer)
Data retention duration for the one hour resolution entries. The configuration defines for how many past days entries will be stored. Set to `-1` for unlimited entries or to `0` to disable the data collection of this resolution.

`log-file` (type: maybe-string)
Specify log file path and name to be used if *use-logging* is set to `1`.

`max-bandwidth` (type: maybe-integer)
Maximum bandwidth for all interfaces. If the interface specific traffic exceeds the given value then the data is assumed to be invalid and rejected. Set to 0 in order to disable the feature. Value range: '`0`'..'`50000`'

`max-bw` (type: maybe-alist)
Same as *max-bandwidth* but can be used for setting individual limits for selected interfaces. This is an association list of interfaces as strings to integer values. For example,

```
(max-bw `(("eth0" .  15000)
          ("ppp0" .  10000)))
```

*bandwidth-detection* is disabled on an interface specific level for each *max-bw* configuration. Value range: '`0`'..'`50000`'

`monthly-months` (default: `25`) (type: maybe-integer)
Data retention duration for the one month resolution entries. The configuration defines for how many past months entries will be stored. Set to `-1` for unlimited entries or to `0` to disable the data collection of this resolution.

`month-rotate` (default: `1`) (type: maybe-integer)
Day of month that months are expected to change. Usually set to 1 but can be set to alternative values for example for tracking monthly billed traffic where the billing period doesn't start on the first day. For example, if set to 7, days of February up to and including the 6th will count for January. Changing this option will not cause existing data to be recalculated. Value range: '`1`'..'`28`'

`month-rotate-affects-years?` (default: `#f`) (type: maybe-boolean)
Enable or disable *month-rotate* also affecting yearly data. Applicable only when *month-rotate* has a value greater than one.

`offline-save-interval` (default: `30`) (type: maybe-integer)
How often in minutes cached interface data is saved to file when all monitored interfaces are offline. Value range: *save-interval*..'`60`'

`pid-file` (default: `"/var/run/vnstatd.pid"`) (type: maybe-string)
Specify pid file path and name to be used.

`poll-interval` (default: `5`) (type: maybe-integer)
How often in seconds interfaces are checked for status changes. Value range: '`2`'..'`60`'

`rescan-database-on-save?` (type: maybe-boolean)
: Automatically discover added interfaces from the database and start monitoring. The rescan is done every *save-interval* or *offline-save-interval* minutes depending on the current activity state.

`save-interval` (default: `5`) (type: maybe-integer)
: How often in minutes cached interface data is saved to file. Value range: ( *update-interval* / 60 )..'`60`'

`save-on-status-change?` (default: `#t`) (type: maybe-boolean)
: Enable or disable the additional saving to file of cached interface data when the availability of an interface changes, i.e., when an interface goes offline or comes online.

`time-sync-wait` (default: `5`) (type: maybe-integer)
: How many minutes to wait during daemon startup for system clock to sync if most recent database update appears to be in the future. This may be needed in systems without a real-time clock (RTC) which require some time after boot to query and set the correct time. `0` = wait disabled. Value range: '`0`'..'`60`'

`top-day-entries` (default: `20`) (type: maybe-integer)
: Data retention duration for the top day entries. The configuration defines how many of the past top day entries will be stored. Set to `-1` for unlimited entries or to `0` to disable the data collection of this resolution.

`trafficless-entries?` (default: `#t`) (type: maybe-boolean)
: Create database entries even when there is no traffic during the entry's time period.

`update-file-owner?` (default: `#t`) (type: maybe-boolean)
: Enable or disable the update of file ownership during daemon process startup. During daemon startup, only database, log and pid files will be modified if the user or group change feature ( *daemon-user* or *daemon-group* ) is enabled and the files don't match the requested user or group. During manual database creation, this option will cause file ownership to be inherited from the database directory if the directory already exists. This option only has effect when the process is started as root or via sudo.

`update-interval` (default: `20`) (type: maybe-integer)
: How often in seconds the interface data is updated. Value range: *poll-interval*..'`300`'

`use-logging` (default: `2`) (type: maybe-integer)
: Enable or disable logging. Accepted values are: `0` = disabled, `1` = logfile and `2` = syslog.

`use-utc?` (type: maybe-boolean)
: Enable or disable using UTC as timezone in the database for all entries. When enabled, all entries added to the database will use UTC regardless of the configured system timezone. When disabled, the configured system timezone will be used. Changing this setting will not result in already existing data to be modified.

`yearly-years` (default: `-1`) (type: maybe-integer)
: Data retention duration for the one year resolution entries. The configuration defines for how many past years entries will be stored. Set to `-1` for unlimited entries or to `0` to disable the data collection of this resolution.

## Servidor Zabbix

Zabbix is a high performance monitoring system that can collect data from a variety of sources and provide the results in a web-based interface. Alerting and reporting is built-in, as well as *templates* for common operating system metrics such as network utilization, CPU load, and disk space consumption.

This service provides the central Zabbix monitoring service; you also need [zabbix-frontend], page 456, to configure Zabbix and display results, and optionally [zabbix-agent], page 455, on machines that should be monitored (other data sources are supported, such as [prometheus-node-exporter], page 449).

`zabbix-server-service-type` [Variable]
: This is the service type for the Zabbix server service. Its value must be a `zabbix-server-configuration` record, shown below.

`zabbix-server-configuration` [Data Type]
: Los campos disponibles de `zabbix-server-configuration` son:

  `zabbix-server` (default: `zabbix-server`) (type: file-like)
  : El paquete zabbix-server.

  `user` (default: `"zabbix"`) (type: string)
  : Usuaria que ejecutará el servidor Zabbix.

  `group` (default: `"zabbix"`) (type: string)
  : Grupo que ejecutará el servidor Zabbix.

  `db-host` (default: `"127.0.0.1"`) (type: string)
  : El nombre de máquina de la base de datos.

  `db-name` (default: `"zabbix"`) (type: string)
  : Nombre de la base de datos.

  `db-user` (default: `"zabbix"`) (type: string)
  : Usuaria de la base de datos.

  `db-password` (default: `""`) (type: string)
  : Contraseña de la base de datos. Por favor, en vez de esto use `include-files` con `DBPassword=SECRET` dentro de un archivo especificado.

  `db-port` (default: `5432`) (type: number)
  : Puerto de la base de datos.

  `log-type` (default: `""`) (type: string)
  : Especifica donde se escriben los mensajes de registro:

    - `system` - syslog.
    - `file` - archivo especificado con el parámetro `log-file`.
    - `console` - salida estándar.

`log-file` (default: `"/var/log/zabbix/server.log"`) (type: string)
Nombre del archivo de registro para el parámetro `file` de `log-type`.

`pid-file` (default: `"/var/run/zabbix/zabbix_server.pid"`) (type: string)
Nombre del archivo de PID.

`ssl-ca-location` (default: `"/etc/ssl/certs/ca-certificates.crt"`) (type: string)
La localización de los archivos de autoridades de certificación (CA) para la verificación de certificados SSL de los servidores.

`ssl-cert-location` (default: `"/etc/ssl/certs"`) (type: string)
Localización de los certificados SSL de los clientes.

`extra-options` (default: `""`) (type: extra-options)
Opciones adicionales que se añadirán al final del archivo de configuración de Zabbix.

`include-files` (default: `'()`) (type: include-files)
Puede incluir archivos individuales o todos los archivos en un directorio en el archivo de configuración.

## Agente Zabbix

The Zabbix agent gathers information about the running system for the Zabbix monitoring server. It has a variety of built-in checks, and can be extended with custom *user parameters* (`https://www.zabbix.com/documentation/current/en/manual/config/items/userparameters`).

`zabbix-agent-service-type` [Variable]
This is the service type for the Zabbix agent service. Its value must be a `zabbix-agent-configuration` record, shown below.

`zabbix-agent-configuration` [Data Type]
Los campos disponibles de `zabbix-agent-configuration` son:

`zabbix-agent` (default: `zabbix-agentd`) (type: file-like)
El paquete zabbix-agent.

`user` (default: `"zabbix"`) (type: string)
Usuaria que ejecutará el agente Zabbix.

`group` (default: `"zabbix"`) (type: string)
Grupo que ejecutará el agente Zabbix.

`hostname` (default: `""`) (type: string)
Nombre de máquina único y sensible a mayúsculas que es necesario para la comprobaciones activas y debe corresponder con el nombre de máquina configurado en el servidor.

`log-type` (default: `""`) (type: string)
Especifica donde se escriben los mensajes de registro:

- `system` - syslog.
- `file` - file specified with `log-file` parameter.

- `console` - salida estándar.

`log-file` (default: `"/var/log/zabbix/agent.log"`) (type: string)
Nombre del archivo de registro para el parámetro `file` de `log-type`.

`pid-file` (default: `"/var/run/zabbix/zabbix_agent.pid"`) (type: string)
Nombre del archivo de PID.

`server` (default: `'("127.0.0.1")`) (type: list)
Lista de direcciones IP, opcionalmente en notación CIDR, o nombres de máquina de servidores y proxy Zabbix. Se aceptarán conexiones entrantes únicamente desde las máquinas proporcionadas aquí.

`server-active` (default: `'("127.0.0.1")`) (type: list)
Lista de pares IP:puerto (o máquina:puerto) de servidores Zabbix y servidores Zabbix para las comprobaciones activas. Si no se especifica un puerto, se usa el puerto predeterminado. Si no se especifica este parámetro, las comprobaciones activas se desactivan.

`extra-options` (default: `""`) (type: extra-options)
Opciones adicionales que se añadirán al final del archivo de configuración de Zabbix.

`include-files` (default: `'()`) (type: include-files)
Puede incluir archivos individuales o todos los archivos en un directorio en el archivo de configuración.

## Motor de visualización de Zabbix

The Zabbix front-end provides a web interface to Zabbix. It does not need to run on the same machine as the Zabbix server. This service works by extending the [PHP-FPM], page 483, and [NGINX], page 472, services with the configuration necessary for loading the Zabbix user interface.

`zabbix-front-end-service-type` [Variable]
This is the service type for the Zabbix web frontend. Its value must be a `zabbix-front-end-configuration` record, shown below.

`zabbix-front-end-configuration` [Data Type]
Los campos disponibles de `zabbix-front-end-configuration` son:

`zabbix-server` (default: `zabbix-server`) (type: file-like)
The Zabbix server package to use.

`nginx` (default: `'()`) (type: list)
List of [nginx-server-configuration], page 475, blocks for the Zabbix front-end. When empty, a default that listens on port 80 is used.

`db-host` (default: `"localhost"`) (type: string)
El nombre de máquina de la base de datos.

`db-port` (default: `5432`) (type: number)
Puerto de la base de datos.

`db-name` (default: `"zabbix"`) (type: string)
: Nombre de la base de datos.

`db-user` (default: `"zabbix"`) (type: string)
: Usuaria de la base de datos.

`db-password` (default: `""`) (type: string)
: Contraseña de la base de datos. Por favor, en vez de esto use `db-secret-file`.

`db-secret-file` (default: `""`) (type: string)
: Archivo secreto que se añadirá al final del archivo `zabbix.conf.php`. Este archivo contiene las credenciales usadas por el motor de visualización de Zabbix. Se espera que usted lo cree manualmente.

`zabbix-host` (default: `"localhost"`) (type: string)
: Nombre de máquina del servidor Zabbix.

`zabbix-port` (default: `10051`) (type: number)
: Puerto del servidor Zabbix.

### 11.10.18 Servicios Kerberos

El módulo `(gnu services kerberos)` proporciona servicios relacionados con el protocolo de identificación *Kerberos*.

#### Servicio Krb5

Los programas que usan una biblioteca cliente de Kerberos habitualmente esperan un archivo de configuración en la ruta `/etc/krb5.conf`. Este servicio genera dicho archivo desde una definición proporcionada en la declaración de sistema operativo. Esto no causa el inicio de ningún daemon.

Este servicio no crea ningún archivo "keytab"—debe crearlos explícitamente usted. Se ha comprobado que este servicio funciona con la biblioteca de cliente `mit-krb5` del MIT. No se han probado otras implementaciones.

`krb5-service-type` [Variable]
: Un tipo de servicio para clientes Kerberos 5.

Este es un ejemplo de su uso:

```
(service krb5-service-type
         (krb5-configuration
          (default-realm "EXAMPLE.COM")
          (allow-weak-crypto? #t)
          (realms (list
                   (krb5-realm
                    (name "EXAMPLE.COM")
                    (admin-server "groucho.example.com")
                    (kdc "karl.example.com"))
                   (krb5-realm
                    (name "ARGRX.EDU")
                    (admin-server "kerb-admin.argrx.edu")
```

```
(kdc "keys.argrx.edu"))))))
```

Este ejemplo proporciona una configuración de cliente Kerberos 5 que:

- Reconoce dos dominios, *sean:* "EXAMPLE.COM" y "ARGRX.EDU", los cuales tienen distintos servidores administrativos y centros de distribución de claves;
- El valor predeterminado será "EXAMPLE.COM" si no se especifica el dominio explícitamente por parte del cliente.
- Acepta servicios cuyos únicos tipos de cifrado implementados se sabe que son débiles.

Los tipos `krb5-realm` y `krb5-configuration` contienen muchos campos. Aquí se describen únicamente los más habitualmente usados. Para obtener una lista complete y una explicación detallada de cada campo, véase la documentación de krb5.conf.

`krb5-realm` [Tipo de datos]

`name` Este campo es una cadena que identifica el nombre del dominio. Una convención habitual es el uso del nombre completo de DNS de su organización, convertido a mayúsculas.

`admin-server`
Este campo es una cadena que identifica la máquina donde se ejecuta el servidor administrativo.

`kdc` Este campo es una cadena que identifica el centro de distribución de claves para el dominio.

`krb5-configuration` [Tipo de datos]

`allow-weak-crypto?` (predeterminado: `#f`)
Si esta opción es `#t` se aceptarán los servicios que únicamente ofrezcan algoritmos de cifrado que se conozca que son débiles.

`default-realm` (predeterminado: `#f`)
Este campo debe ser una cadena que identifique el dominio predeterminado de Kerberos para los clientes. Debería proporcionar el nombre de su dominio Kerberos. Si este valor es `#f`, el dominio debe especificarse en cada principal de Kerberos cuando se invoquen programas como `kinit`.

`realms` Debe ser una lista no vacía de objetos `krb5-realm`, accesibles por los clientes. Normalmente, uno de ellos tendrá un campo `name` que corresponda con el campo `default-realm`.

## Servicio krb5 de PAM

El servicio `pam-krb5` le permite la identificación para el ingreso al sistema y la gestión de contraseñas mediante Kerberos. Este servicio es necesario si desea que aplicaciones que permiten PAM lleven a cabo la identificación de usuarias mediante el uso de Kerberos.

`pam-krb5-service-type` [Variable]
Un tipo de servicio para el módulo PAM de Kerberos 5.

`pam-krb5-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del módulo PAM de Kerberos 5. Este tipo tiene los siguientes parámetros:

`pam-krb5` (predeterminado: `pam-krb5`)
: El paquete pam-krb5 usado.

`minimum-uid` (predeterminado: `1000`)
: El ID de usuaria mínimo con el que se permitirán los intentos de identificación con Kerberos. El proceso de identificación de las cuentas locales con valores menores fallará de manera silenciosa.

### 11.10.19 Servicios LDAP

#### Authentication against LDAP with nslcd

El módulo (`gnu services authentication`) proporciona el tipo `nslcd-service-type`, que puede usarse para la identificación a través de un servidor LDAP. Además de la configuración del servicio en sí, puede desear añadir `ldap` como servicio de nombres en el selector de servicios de nombres (NSS). See Section 11.13 [Selector de servicios de nombres], page 622, para información detallada.

Aquí se encuentra una declaración simple de sistema operativo con la configuración predeterminada de `nslcd-service-type` y una configuración del selector de servicios de nombre que consulta en último lugar al servicios de nombres `ldap`:

```
(use-service-modules authentication)
(use-modules (gnu system nss))
...
(operating-system
  ...
  (services
    (cons*
      (service nslcd-service-type)
      (service dhcp-client-service-type)
      %base-services))
  (name-service-switch
   (let ((services (list (name-service (name "db"))
                         (name-service (name "files"))
                         (name-service (name "ldap")))))
     (name-service-switch
      (inherit %mdns-host-lookup-nss)
      (password services)
      (shadow   services)
      (group    services)
      (netgroup services)
      (gshadow  services)))))
```

Los campos disponibles de `nslcd-configuration` son:

`package nss-pam-ldapd` [parámetro de `nslcd-configuration`]
: El paquete `nss-pam-ldapd` usado.

`maybe-number threads` [parámetro de `nslcd-configuration`]
El número de hilos a iniciar que pueden gestionar peticiones y realizar consultas en LDAP. Cada hilo abre una conexión separada al servidor LDAP. Se inician 5 hilos de manera predeterminada.

El valor predeterminado es '`disabled`'.

`string uid` [parámetro de `nslcd-configuration`]
Especifica el id de usuaria con el que debe ejecutarse el daemon.

El valor predeterminado es '`"nslcd"`'.

`string gid` [parámetro de `nslcd-configuration`]
Especifica el id de grupo con el que debe ejecutarse el daemon.

El valor predeterminado es '`"nslcd"`'.

`opción-registro log` [parámetro de `nslcd-configuration`]
This option controls the way logging is done via a list containing SCHEME and LEVEL. The SCHEME argument may either be the symbols '`none`' or '`syslog`', or an absolute file name. The LEVEL argument is optional and specifies the log level. The log level may be one of the following symbols: '`crit`', '`error`', '`warning`', '`notice`', '`info`' or '`debug`'. All messages with the specified log level or higher are logged.

El valor predeterminado es '`'("/var/log/nslcd" info)`'.

`lista uri` [parámetro de `nslcd-configuration`]
La lista de URI de servidores LDAP. Normalmente, únicamente se usará el primer servidor y los siguientes se usan en caso de fallo.

Defaults to '`'("ldap://localhost:389/")`'.

`maybe-string ldap-version` [parámetro de `nslcd-configuration`]
La versión del protocolo LDAP usada. El valor predeterminado usa la versión máxima implementada por la biblioteca LDAP.

El valor predeterminado es '`disabled`'.

`maybe-string binddn` [parámetro de `nslcd-configuration`]
Especifica el nombre distinguido con el que enlazarse en el servidor de directorio para las búsquedas. El valor predeterminado se enlaza de forma anónima.

El valor predeterminado es '`disabled`'.

`maybe-string bindpw` [parámetro de `nslcd-configuration`]
Especifica las credenciales usadas para el enlace. Esta opción tiene utilidad únicamente cuando se usa con binddn.

El valor predeterminado es '`disabled`'.

`maybe-string rootpwmoddn` [parámetro de `nslcd-configuration`]
Especifica el nombre distinguido usado cuando la usuaria root intenta modificar la contraseña de una usuaria mediante el módulo de PAM.

El valor predeterminado es '`disabled`'.

`maybe-string rootpwmodpw` [parámetro de `nslcd-configuration`]
Especifica las credenciales con las que enlazarse si la usuaria root intenta cambiar la contraseña de una usuaria. Esta opción tiene utilidad únicamente cuando se usa con rootpwmoddn.

El valor predeterminado es '`disabled`'.

`maybe-string sasl-mech` [parámetro de `nslcd-configuration`]
Especifica el mecanismo de SASL usado cuando se realice la identificación con SASL.

El valor predeterminado es '`disabled`'.

`maybe-string sasl-realm` [parámetro de `nslcd-configuration`]
Especifica el dominio de SASL usado cuando se realice la identificación con SASL.

El valor predeterminado es '`disabled`'.

`maybe-string sasl-authcid` [parámetro de `nslcd-configuration`]
Especifica la identidad de verificación usada cuando se realice la identificación con SASL.

El valor predeterminado es '`disabled`'.

`maybe-string sasl-authzid` [parámetro de `nslcd-configuration`]
Especifica la identidad de autorización usada cuando se realice la identificación con SASL.

El valor predeterminado es '`disabled`'.

`maybe-boolean sasl-canonicalize?` [parámetro de `nslcd-configuration`]
Determina si el nombre de máquina del servidor LDAP debe transformarse a su forma canónica. Si se activa, la librería LDAP realizará una búsqueda inversa de nombre de máquina. De manera predeterminada, se delega en la biblioteca la decisión de realizar esta comprobación o no.

El valor predeterminado es '`disabled`'.

`maybe-string krb5-ccname` [parámetro de `nslcd-configuration`]
Establece el nombre para la caché de credenciales GSS-API de Kerberos.

El valor predeterminado es '`disabled`'.

`string base` [parámetro de `nslcd-configuration`]
El directorio de búsqueda base.

El valor predeterminado es '`"dc=example,dc=com"`'.

`opción-de-ámbito scope` [parámetro de `nslcd-configuration`]
Especifica el ámbito de búsqueda (subtree, oneleve, base o children). El ámbito predeterminado es subtree; el ámbito base casi nunca es útil para búsquedas del servicio de nombres; el ámbito children no está implementado en todos los servidores.

El valor predeterminado es '`'(subtree)`'.

`maybe-deref-option deref` [parámetro de `nslcd-configuration`]
Especifica la política para seguir las referencias de los alias. La política predeterminada es nunca seguir las referencias de los alias.

El valor predeterminado es '`disabled`'.

`maybe-boolean referrals` [parámetro de `nslcd-configuration`]
Especifica si el seguimiento automático de referencias debe activarse. El seguimiento de referencias es comportamiento predeterminado.

El valor predeterminado es '`disabled`'.

`lista-asociación-entrada maps` [parámetro de `nslcd-configuration`]
Esta opción permite que se busquen atributos personalizados en vez de los atributos predeterminados de RFC 2307. Es una lista de asociaciones, de las que cada una consiste en el nombre de la asociación, el atributo de RFC 2307 al que corresponde y la expresión de búsqueda del atributo en la forma que esté disponible en el directorio.

El valor predeterminado es '`'()`'.

`lista-asociación-entrada filters` [parámetro de `nslcd-configuration`]
Una lista de filtros que consiste en el nombre de una asociación a la que se aplica el filtro y una expresión de filtrado de búsqueda de LDAP.

El valor predeterminado es '`'()`'.

`maybe-number bind-timelimit` [parámetro de `nslcd-configuration`]
Especifica el tiempo límite usado en segundos durante la conexión al servidor de directorio. El valor predeterminado son 10 segundos.

El valor predeterminado es '`disabled`'.

`maybe-number timelimit` [parámetro de `nslcd-configuration`]
Especifica el tiempo límite (en segundos) durante el que se esperará una respuesta del servidor LDAP. Un valor de cero, por omisión, hace que se espere de manera indefinida hasta que las búsquedas se completen.

El valor predeterminado es '`disabled`'.

`maybe-number idle-timelimit` [parámetro de `nslcd-configuration`]
Especifica el periodo de inactividad (en segundos) tras el cual se cerrará la conexión con el servidor LDAP. El valor predeterminado no cierra las conexiones por inactividad.

El valor predeterminado es '`disabled`'.

`maybe-number reconnect-sleeptime` [parámetro de `nslcd-configuration`]
Especifica en número de segundos que se dormirá cuando falle la conexión a todos los servidores LDAP. De manera predeterminada se espera un segundo entre el primer fallo y el primer reintento.

El valor predeterminado es '`disabled`'.

`maybe-number reconnect-retrytime` [parámetro de `nslcd-configuration`]
Especifica el tiempo tras el cual el servidor LDAP se considera no disponible de manera permanente. Una vez se alcance este tiempo, los reintentos se realizarán una vez en cada periodo de tiempo igual al especificado. El valor predeterminado es 10 segundos.

El valor predeterminado es '`disabled`'.

`maybe-ssl-option ssl` [parámetro de `nslcd-configuration`]
Determina si se usa SSL/TLS o no (el comportamiento predeterminado es no hacerlo). Si se especifica 'start-tls, se usa StartTLS en vez de la transmisión del protocolo LDAP en crudo sobre SSL.

El valor predeterminado es '`disabled`'.

`maybe-tls-reqcert-option tls-reqcert` [parámetro de `nslcd-configuration`]
Especifica las comprobaciones que se deben realizar con un certificado proporcionado por el servidor. El significado de los valores se describe en la página de manual de ldap.conf(5).

El valor predeterminado es '`disabled`'.

`maybe-string tls-cacertdir` [parámetro de `nslcd-configuration`]
Especifica el directorio que contiene los certificados X.509 para la identificación de pares. Este parámetro se ignora si se usa GnuTLS.

El valor predeterminado es '`disabled`'.

`maybe-string tls-cacertfile` [parámetro de `nslcd-configuration`]
Especifica la ruta al certificado X.509 para la identificación de pares.

El valor predeterminado es '`disabled`'.

`maybe-string tls-randfile` [parámetro de `nslcd-configuration`]
Especifica la ruta de la fuente de entropía. Este parámetro se ignora si se usa GnuTLS.

El valor predeterminado es '`disabled`'.

`maybe-string tls-ciphers` [parámetro de `nslcd-configuration`]
Especifica como una cadena los algoritmos de cifrado usados para TLS.

El valor predeterminado es '`disabled`'.

`maybe-string tls-cert` [parámetro de `nslcd-configuration`]
Especifica la ruta al archivo que contiene el certificado local para la identificación de clientes con TLS.

El valor predeterminado es '`disabled`'.

`maybe-string tls-key` [parámetro de `nslcd-configuration`]
Especifica la ruta al archivo que contiene la clave privada para la identificación de clientes con TLS.

El valor predeterminado es '`disabled`'.

`maybe-number pagesize` [parámetro de `nslcd-configuration`]
Proporcione un valor superior a 0 para solicitar al servidor LDAP que proporcione los resultados divididos en páginas de acuerdo con el RFC2696. El valor predeterminado (0) no solicita resultados divididos en páginas.

El valor predeterminado es '`disabled`'.

`maybe-ignore-users-option` [parámetro de `nslcd-configuration`]
`nss-initgroups-ignoreusers`

Esta opción previene las búsquedas de pertenencia a grupos a través de LDAP sobre las usuarias especificadas. De manera alternativa, se puede usar el valor 'all-local. Con dicho valor nslcd construye al inicio una lista completa de usuarias que no se encuentren en LDAP.

El valor predeterminado es '`disabled`'.

`maybe-number nss-min-uid` [parámetro de `nslcd-configuration`]

Esta opción hace que se ignoren las usuarias de LDAP con un identificador numérico inferior al valor especificado.

El valor predeterminado es '`disabled`'.

`maybe-number nss-uid-offset` [parámetro de `nslcd-configuration`]

Esta opción especifica un desplazamiento que se añade a todos los identificadores numéricos de usuaria de LDAP. Puede usarse para evitar colisiones de identificadores con usuarias locales.

El valor predeterminado es '`disabled`'.

`maybe-number nss-gid-offset` [parámetro de `nslcd-configuration`]

Esta opción especifica un desplazamiento que se añade a todos los identificadores numéricos de grupos de LDAP. Puede usarse para evitar colisiones de identificadores con grupos locales.

El valor predeterminado es '`disabled`'.

`maybe-boolean nss-nested-groups` [parámetro de `nslcd-configuration`]

Cuando se activa esta opción, un grupo puede contener como atributo la pertenencia a otro grupo. Los miembros de grupos anidados se devuelven en el grupo superior y los grupos superiores se devuelven cuando se busquen los grupos de una usuaria específica. El valor predeterminado determina que no se realicen búsquedas adicionales para grupos anidados.

El valor predeterminado es '`disabled`'.

`maybe-boolean` [parámetro de `nslcd-configuration`]
`nss-getgrent-skipmembers`

Cuando se activa esta opción, la lista de miembros de un grupo no se obtiene en las búsquedas de grupos. Las búsquedas que busquen los grupos de los que una usuaria es miembro continuarán funcionando de manera que probablemente a la usuaria se le asignen los grupos correctos durante el ingreso al sistema.

El valor predeterminado es '`disabled`'.

`maybe-boolean` [parámetro de `nslcd-configuration`]
`nss-disable-enumeration`

Cuando se activa esta opción, las funciones que provocan la carga de todas las entradas usuaria/grupo del directorio no tendrán éxito al realizarlo. Esto puede reducir de forma dramática la carga del servidor LDAP cuando existe un gran número de usuarias y/o grupos. Esta opción no se recomienda para la mayoría de las configuraciones.

El valor predeterminado es '`disabled`'.

`maybe-string validnames` [parámetro de `nslcd-configuration`]
Esta opción puede usarse para especificar cómo se verifican en el sistema los nombres de usuaria y grupo. Este patrón se usa para comprobar todos los nombres de usuarias y grupos que se soliciten y proporcionen a través de LDAP.

El valor predeterminado es '`disabled`'.

`maybe-boolean ignorecase` [parámetro de `nslcd-configuration`]
Especifica si se realizarán las búsquedas sin diferenciar mayúsculas y minúsculas o no. Su activación puede abrir puntos vulnerables que permitan la omisión de las comprobaciones de autorización e introducir vulnerabilidades que permitan el envenenamiento de la caché de nscd, lo que puede provocar la denegación del servicio.

El valor predeterminado es '`disabled`'.

`maybe-boolean pam-authc-ppolicy` [parámetro de `nslcd-configuration`]
Esta opción determina si los controles de la política de contraseñas se solicitan y manejan desde el servidor LDAP cuando se realice la identificación de usuarias.

El valor predeterminado es '`disabled`'.

`maybe-string pam-authc-search` [parámetro de `nslcd-configuration`]
De manera predeterminada nslcd realiza una búsqueda LDAP con las credenciales de la usuaria tras la orden BIND (identificación) para asegurarse de que la opción BIND fue satisfactoria. La búsqueda predeterminada es una simple comprobación de la existencia del DN de la usuaria. Se puede especificar un filtro de búsqueda que se usará en vez de dicha búsqueda. Debe devolver al menos una entrada.

El valor predeterminado es '`disabled`'.

`maybe-string pam-authz-search` [parámetro de `nslcd-configuration`]
Esta opción permite la configuración detallada de las comprobaciones de autorización que deben realizarse. El filtro de búsqueda especificado es ejecutado, y si cualquier entrada corresponde se permite el acceso, el cual se deniega en caso contrario.

El valor predeterminado es '`disabled`'.

`maybe-string pam-password-prohibit-message` [parámetro de `nslcd-configuration`]
Si se proporciona esta opción, se denegará la modificación de contraseñas a través de pam_ldap y en vez de ello el mensaje especificado se presentará a la usuaria. El mensaje puede usarse para redirigir a la usuaria a un medio alternativo para el cambio de su contraseña.

El valor predeterminado es '`disabled`'.

`lista pam-services` [parámetro de `nslcd-configuration`]
Lista de nombres de servicio de PAM para los que la identificación de LDAP debería ser suficiente.

El valor predeterminado es '`'()`'.

## LDAP Directory Server

The `(gnu services ldap)` module provides the `directory-server-service-type`, which can be used to create and launch an LDAP server instance.

Here is an example configuration of the `directory-server-service-type`:

```
(use-service-modules ldap)


...
(operating-system
  ...
  (services
    (cons
      (service directory-server-service-type
               (directory-server-instance-configuration
                (slapd
                 (slapd-configuration
                  (root-password "{PBKDF2_SHA256}AAAgAG...ABSOLUTELYSECRET")))))
      %base-services)))
```

The root password should be generated with the `pwdhash` utility that is provided by the `389-ds-base` package.

Note that changes to the directory server configuration will not be applied to existing instances. You will need to back up and restore server data manually. Only new directory server instances will be created upon system reconfiguration.

`directory-server-instance-configuration` [Data Type]

Available `directory-server-instance-configuration` fields are:

`package` (default: `389-ds-base`) (type: file-like)
: The `389-ds-base` package.

`config-version` (default: `2`) (type: number)
: Sets the format version of the configuration file. To use the INF file with `dscreate`, this parameter must be 2.

`full-machine-name` (default: `"localhost"`) (type: string)
: Sets the fully qualified hostname (FQDN) of this system.

`selinux` (default: `#false`) (type: boolean)
: Enables SELinux detection and integration during the installation of this instance. If set to `#true`, `dscreate` auto-detects whether SELinux is enabled.

`strict-host-checking` (default: `#true`) (type: boolean)
: Sets whether the server verifies the forward and reverse record set in the `full-machine-name` parameter. When installing this instance with GSSAPI authentication behind a load balancer, set this parameter to `#false`.

`systemd` (default: `#false`) (type: boolean)
: Enables systemd platform features. If set to `#true`, `dscreate` auto-detects whether systemd is installed.

`slapd` (type: slapd-configuration)
Configuration of slapd.

`slapd-configuration` [Data Type]
Available `slapd-configuration` fields are:

`instance-name` (default: `"localhost"`) (type: string)
Sets the name of the instance. You can refer to this value in other parameters of this INF file using the `{instance_name}` variable. Note that this name cannot be changed after the installation!

`user` (default: `"dirsrv"`) (type: string)
Sets the user name the ns-slapd process will use after the service started.

`group` (default: `"dirsrv"`) (type: string)
Sets the group name the ns-slapd process will use after the service started.

`port` (default: `389`) (type: number)
Sets the TCP port the instance uses for LDAP connections.

`secure-port` (default: `636`) (type: number)
Sets the TCP port the instance uses for TLS-secured LDAP connections (LDAPS).

`root-dn` (default: `"cn=Directory Manager"`) (type: string)
Sets the *Distinquished Name* (DN) of the administrator account for this instance.

`root-password` (default: `"{invalid}YOU-SHOULD-CHANGE-THIS"`) (type: string)
Sets the password of the account specified in the `root-dn` parameter. You can either set this parameter to a plain text password `dscreate` hashes during the installation or to a "{algorithm}hash" string generated by the `pwdhash` utility. Note that setting a plain text password can be a security risk if unprivileged users can read this INF file!

`self-sign-cert` (default: `#true`) (type: boolean)
Sets whether the setup creates a self-signed certificate and enables TLS encryption during the installation. This is not suitable for production, but it enables administrators to use TLS right after the installation. You can replace the self-signed certificate with a certificate issued by a certificate authority.

`self-sign-cert-valid-months` (default: `24`) (type: number)
Set the number of months the issued self-signed certificate will be valid.

`backup-dir` (default: `"/var/lib/dirsrv/slapd-{instance_name}/bak"`) (type: string)
Set the backup directory of the instance.

`cert-dir` (default: `"/etc/dirsrv/slapd-{instance_name}"`) (type: string)
Sets the directory of the instance's Network Security Services (NSS) database.

`config-dir` (default: `"/etc/dirsrv/slapd-{instance_name}"`) (type: string)
Sets the configuration directory of the instance.

`db-dir` (default: `"/var/lib/dirsrv/slapd-{instance_name}/db"`) (type: string)
Sets the database directory of the instance.

`initconfig-dir` (default: `"/etc/dirsrv/registry"`) (type: string)
Sets the directory of the operating system's rc configuration directory.

`ldif-dir` (default: `"/var/lib/dirsrv/slapd-{instance_name}/ldif"`) (type: string)
Sets the LDIF export and import directory of the instance.

`lock-dir` (default: `"/var/lock/dirsrv/slapd-{instance_name}"`) (type: string)
Sets the lock directory of the instance.

`log-dir` (default: `"/var/log/dirsrv/slapd-{instance_name}"`) (type: string)
Sets the log directory of the instance.

`run-dir` (default: `"/run/dirsrv"`) (type: string)
Sets PID directory of the instance.

`schema-dir` (default: `"/etc/dirsrv/slapd-{instance_name}/schema"`) (type: string)
Sets schema directory of the instance.

`tmp-dir` (default: `"/tmp"`) (type: string)
Sets the temporary directory of the instance.

`backend-userroot` (type: backend-userroot-configuration)
Configuration of the userroot backend.

`backend-userroot-configuration` [Data Type]
Available `backend-userroot-configuration` fields are:

`create-suffix-entry?` (default: `#false`) (type: boolean)
: Set this parameter to `#true` to create a generic root node entry for the suffix in the database.

`require-index?` (default: `#false`) (type: boolean)
: Set this parameter to `#true` to refuse unindexed searches in this database.

`sample-entries` (default: `"no"`) (type: string)
: Set this parameter to `"yes"` to add latest version of sample entries to this database. Or, use `"001003006"` to use the 1.3.6 version sample entries. Use this option, for example, to create a database for testing purposes.

`suffix` (type: maybe-string)
: Sets the root suffix stored in this database. If you do not set the suffix attribute the install process will not create the backend/suffix. You can also create multiple backends/suffixes by duplicating this section.

### 11.10.20 Servicios Web

El módulo `(gnu services web)` proporciona el servidor HTTP Apache, el servidor web nginx y también un recubrimiento del daemon de fastcgi.

### Servidor HTTP Apache

`httpd-service-type` [Variable]

Tipo de servicio para el servidor Apache HTTP (`https://httpd.apache.org/`) (*httpd*). El valor para este tipo de servicio es un registro `httpd-configuration`.

Un ejemplo de configuración simple se proporciona a continuación.

```
(service httpd-service-type
         (httpd-configuration
           (config
             (httpd-config-file
               (server-name "www.example.com")
               (document-root "/srv/http/www.example.com")))))
```

Otros servicios también pueden extender el tipo `httpd-service-type` para añadir su contribución a la configuración.

```
(simple-service 'www.example.com-server httpd-service-type
                (list
                  (httpd-virtualhost
                    "*:80"
                    (list (string-join '("ServerName www.example.com"
                                         "DocumentRoot /srv/http/www.example.com"
                                        "\n")))))
```

Los detalles de los tipos de registro `httpd-configuration`, `httpd-module`, `httpd-config-file` y `httpd-virtualhost` se proporcionan a continuación.

`httpd-configuration` [Tipo de datos]

Este tipo de datos representa la configuración del servicio httpd.

`package` (predeterminado: `httpd`)
El paquete httpd usado.

`pid-file` (predeterminado: `"/var/run/httpd"`)
El archivo pid usado por el servicio de Shepherd.

`config` (predeterminado: `(httpd-config-file)`)
The configuration file to use with the httpd service. The default value is a `httpd-config-file` record, but this can also be a different G-expression that generates a file, for example a `plain-file`. A file outside of the store can also be specified through a string.

`httpd-module` [Tipo de datos]

Este es el tipo de datos que representa un módulo para el servicio httpd.

`name` El nombre del módulo.

`file` The file for the module. This can be relative to the httpd package being used, the absolute location of a file, or a G-expression for a file within the store, for example `(file-append mod-wsgi "/modules/mod_wsgi.so")`.

`%default-httpd-modules` [Variable]

Una lista de objetos `httpd-module` predeterminados.

`httpd-config-file` [Tipo de datos]

Este tipo de datos representa un archivo de configuración para el servicio httpd.

`modules` (predeterminados: `%default-httpd-modules`)
The modules to load. Additional modules can be added here, or loaded by additional configuration.

Por ejemplo, para manejar las peticiones de archivos PHP, puede usar el módulo `mod_proxy_fcgi` de Apache junto con `php-fpm-service-type`:

```
(service httpd-service-type
         (httpd-configuration
          (config
           (httpd-config-file
            (modules (cons*
                      (httpd-module
                       (name "modulo_proxy")
                       (file "modules/mod_proxy.so"))
                      (httpd-module
                       (name "module_proxy_fcgi")
                       (file "modules/mod_proxy_fcgi.so"))
                      %default-httpd-modules))
            (extra-config (list "\
<FilesMatch \\.php$>
    SetHandler \"proxy:unix:/var/run/php-fpm.sock|fcgi://localhost/\"
</FilesMatch>"))))))
```

```
(service php-fpm-service-type
         (php-fpm-configuration
          (socket "/var/run/php-fpm.sock")
          (socket-group "httpd")))
```

`server-root` (predeterminado: `httpd`)
: The `ServerRoot` in the configuration file, defaults to the httpd package. Directives including `Include` and `LoadModule` are taken as relative to the server root.

`server-name` (predeterminado: `#f`)
: El campo `ServerName` (nombre del servidor) en el archivo de configuración, el cual se usa para especificar el esquema de peticiones, nombre de máquina y puerto que el servidor usa para su propia identificación.

  This doesn't need to be set in the server config, and can be specified in virtual hosts. The default is `#f` to not specify a `ServerName`.

`document-root` (predeterminado: `"/srv/http"`)
: La raíz (`DocumentRoot`) desde la que se proporcionan los archivos.

`listen` (predeterminado: `'("80")`)
: The list of values for the `Listen` directives in the config file. The value should be a list of strings, when each string can specify the port number to listen on, and optionally the IP address and protocol to use.

`pid-file` (predeterminado: `"/var/run/httpd"`)
: The `PidFile` to use. This should match the `pid-file` set in the `httpd-configuration` so that the Shepherd service is configured correctly.

`error-log` (predeterminado: `"/var/log/httpd/error_log"`)
: El archivo `ErrorLog` en el que el servidor registrará los errores.

`user` (predeterminada: `"httpd"`)
: La usuaria como la que el servidor responderá a las peticiones.

`group` (predeterminado: `"httpd"`)
: El grupo como el que el servidor responderá a las peticiones.

`extra-config` (predeterminadas: `(list "TypesConfig etc/httpd/mime.types")`)
: Una lista de cadenas y expresiones-G que se añadirán al final del archivo de configuración.

  Los valores con los que se extiende el servicio se añaden al final de esta lista.

`httpd-virtualhost` [Tipo de datos]

Este tipo de datos representa un bloque de configuración de máquina virtual del servicio httpd.

Se deben añadir a la configuración adicional extra-config del servicio httpd-service.

```
(simple-service 'servidor-www.example.com httpd-service-type
                (list
                  (httpd-virtualhost
```

```
                        "*:80"
                        (list (string-join '("ServerName "www.example.com"
                                             "DocumentRoot /srv/http/www.example.com"
                                           "\n")))))
```

`addresses-and-ports`
: Las direcciones y puertos de la directiva `VirtualHost`.

`contents` El contenido de la directiva `VirtualHost`; debe ser una lista de cadenas y expresiones-G.

## NGINX

`nginx-service-type` [Variable]

Tipo de servicio para el servidor web NGinx (`https://nginx.org/`). El valor para este tipo de servicio es un registro `<nginx-configuration>`.

Un ejemplo de configuración simple se proporciona a continuación.

```
(service nginx-service-type
         (nginx-configuration
           (server-blocks
             (list (nginx-server-configuration
                     (server-name '("www.example.com"))
                     (root "/srv/http/www.example.com"))))))
```

Además de añadiendo bloques de servidor a la configuración del servicio de manera directa, este servicio puede extenderse con otros servicios para añadir bloques de servidor, como en este ejemplo:

```
(simple-service 'mi-servidor-adicional nginx-service-type
                (list (nginx-server-configuration
                        (root "/srv/http/sitio-adicional")
                        (try-files (list "$uri" "$uri/index.html")))))
```

Durante su inicio, `nginx` no ha leído todavía su archivo de configuración, por lo que usa un archivo predeterminado para registrar los mensajes de error. Si se produce algún fallo al cargar su archivo de configuración, allí es donde se registran los mensajes de error. Tras la carga del archivo de configuración, el archivo de registro de errores predeterminado cambia al especificado allí. En nuestro caso, los mensajes de error durante el inicio se pueden encontrar en `/var/run/nginx/logs/error.log`, y tras la configuración en `/var/log/nginx/error.log`. La segunda ruta puede cambiarse con las opciones de configuración *log-directory*.

`nginx-configuration` [Tipo de datos]

This data type represents the configuration for NGinx. Some configuration can be done through this and the other provided record types, or alternatively, a config file can be provided.

`nginx` (predeterminado: `nginx`)
: El paquete nginx usado.

`shepherd-requirement` (default: `'()`)
: This is a list of symbols naming Shepherd services the nginx service will depend on.

  This is useful if you would like `nginx` to be started after a back-end web server or a logging service such as Anonip has been started.

`log-directory` (predeterminado: `"/var/log/nginx"`)
: Directorio en el que NGinx escribirá los archivos de registro.

`log-level` (default: `'error`) (type: symbol)
: Logging level, which can be any of the following values: `'debug`, `'info`, `'notice`, `'warn`, `'error`, `'crit`, `'alert`, or `'emerg`.

`run-directory` (predeterminado: `"/var/run/nginx"`)
: Directorio en el que NGinx crea el archivo de PID, y escribe archivos temporales.

`server-blocks` (predeterminados: `'()`)
: Una lista de *bloques de servidor* que se crearán en el archivo de configuración generado; los elementos deben ser del tipo `<nginx-server-configuration>`.

  El ejemplo siguiente configura NGinx para proporcionar `www.example.com` a partir del directorio `/srv/http/www.example.com`, sin usar HTTPS.

```
(service nginx-service-type
         (nginx-configuration
           (server-blocks
             (list (nginx-server-configuration
                     (server-name '("www.example.com"))
                     (root "/srv/http/www.example.com"))))))
```

`upstream-blocks` (predeterminados: `'()`)
: Una lista de *bloques upstream* creada en el archivo de configuración generado, los elementos deben ser del tipo `<nginx-upstream-configuration>`.

  La configuración de proveedores a través de `upstream-blocks` puede ser útil al combinarse con `location` en los registros `<nginx-server-configuration>`. El siguiente ejemplo crea la configuración de un servidor con una configuración de ruta, que hará de intermediaria en las peticiones a la configuración de proveedores, que delegarán las peticiones en dos servidores.

```
(service
  nginx-service-type
  (nginx-configuration
    (server-blocks
      (list (nginx-server-configuration
              (server-name '("www.example.com"))
              (root "/srv/http/www.example.com")
              (locations
```

```
                                (list
                                  (nginx-location-configuration
                                  (uri "/ruta1")
                                  (body '("proxy_pass http://servidor-proxy;")))))))
                    (upstream-blocks
                      (list (nginx-upstream-configuration
                              (name "servidor-proxy")
                              (servers (list "servidor1.example.com"
                                             "servidor2.example.com")))))))
```

`file` (predeterminado: `#f`)
: Si se proporciona un archivo de configuración con *file*, se usará este, en vez de generar un archivo de configuración a partir de los parámetros `log-directory`, `run-directory`, `server-blocks` y `upstream-blocks` proporcionados. Para conseguir un funcionamiento adecuado, estos parámetros deben corresponder con el contenido de *file*, lo que asegura que los directorios se hayan creado durante la activación del servicio.

  Esto puede ser útil si ya dispone de un archivo de configuración, o no es posible hacer lo que necesita con el resto de opciones del registro nginx-configuration.

`server-names-hash-bucket-size` (predeterminado: `#f`)
: Tamaño del cubo para las tablas hash de los nombres de servidor, cuyo valor predeterminado es `#f` para que se use el tamaño de la línea de caché de los procesadores.

`server-names-hash-bucket-max-size` (predeterminado: `#f`)
: Tamaño máximo del cubo para las tablas hash de nombres de servidor.

`modules` (predeterminados: `'()`)
: Lista de módulos dinámicos de nginx cargados. Debe ser una lista de nombres de archivo de módulos cargables, como en este ejemplo:

```
(modules
 (list
  (file-append nginx-accept-language-module "\
/etc/nginx/modules/ngx_http_accept_language_module.so")
  (file-append nginx-lua-module "\
/etc/nginx/modules/ngx_http_lua_module.so")))
```

`lua-package-path` (predeterminada: `'()`)
: Lista de paquetes de lua para nginx cargados. Debe ser una lista de nombres de archivo de módulos cargables, como en este ejemplo:

```
(lua-package-path (list lua-resty-core
                        lua-resty-lrucache
                        lua-resty-signal
                        lua-tablepool
                        lua-resty-shell))
```

`lua-package-cpath` (predeterminada: `'()`)
: Lista de paquetes C de lua para nginx cargados. Debe ser una lista de nombres de archivo de módulos cargables, como en este ejemplo:

```
(lua-package-cpath (list lua-resty-signal))
```

`global-directives` (predeterminadas: `'((events . ()))`)
: Lista asociativa de directivas globales para el nivel superior de la configuración de nginx. Los valores en sí mismos pueden ser listas asociativas.

```
(global-directives
 `((worker_processes . 16)
   (pcre_jit . on)
   (events . ((worker_connections . 1024)))))
```

`extra-content` (predeterminado: `""`)
: Contenido adicional para el bloque `http`. Debe ser una cadena o una expresión-G que evalúe a una cadena.

`nginx-server-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de un bloque de servidor nginx. Este tipo tiene los siguientes parámetros:

`listen` (predeterminadas: `'("80" "443 ssl")`)
: Cada directiva `listen` establece la dirección y el puerto para IP, o la ruta para un socket de dominio de UNIX sobre el que el servidor acepta peticiones. Se puede especificar tanto dirección y puerto como únicamente la dirección o únicamente el puerto. Una dirección puede ser también un nombre de máquina, por ejemplo:

```
'("127.0.0.1:8000" "127.0.0.1" "8000" "*:8000" "localhost:8000")
```

`server-name` (predeterminados: `(list 'default)`)
: Una lista de nombres de servidor que este servidor representa. `'default` representa el servidor predeterminado para conexiones que no correspondan a otro servidor.

`root` (predeterminada: `"/srv/http"`)
: Raíz del sitio web que nginx proporcionará.

`locations` (predeterminado: `'()`)
: Una lista de registros *nginx-location-configuration* o *nginx-named-location-configuration* usados dentro de este bloque de servidor.

`index` (predeterminado: `(list "index.html")`)
: Archivos de índice buscados cuando los clientes solicitan un directorio. Si no se encuentra ninguno, Nginx enviará la lista de archivos del directorio.

`try-files` (predeterminado: `'()`)
: Una lista de archivos cuya existencia se comprueba en el orden especificado. `nginx` usará el primer archivo que encuentre para procesar la petición.

`ssl-certificate` (predeterminado: `#f`)
: Lugar donde se encuentra el certificado para conexiones seguras. Proporcione `#f` si no dispone de un certificado o no desea usar HTTPS.

`ssl-certificate-key` (predeterminado: `#f`)
: Lugar donde se encuentra la clave privada para conexiones seguras. Proporcione `#f` si no dispone de una clave o no desea usar HTTPS.

`server-tokens?` (predeterminado: `#f`)
: Determina si el servidor debe añadir su configuración a las respuestas.

`raw-content` (predeterminado: `'()`)
: Una lista de líneas que se añadirán literalmente al bloque del servidor.

`nginx-upstream-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de un bloque `upstream` de nginx. Este tipo tiene los siguientes parámetros:

`name` Nombre para este grupo de servidores.

`servers` Especifica las direcciones de los servidores en el grupo. Las direcciones se pueden proporcionar mediante direcciones IP (por ejemplo '`127.0.0.1`'), nombres de dominio (por ejemplo '`maquina1.example.com`') o rutas de socket de UNIX mediante el prefijo '`unix:`'. El puerto predeterminado para las direcciones IP o nombres de dominio es el 80, y se puede proporcionar un puerto de manera explícita.

`extra-content`
: A string or list of strings to add to the upstream block.

`nginx-location-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de un bloque `location` de nginx. Este tipo tiene los siguientes parámetros:

`uri` URI a la que corresponde este bloque de location.

`body` Body of the location block, specified as a list of strings. This can contain many configuration directives. For example, to pass requests to a upstream server group defined using an `nginx-upstream-configuration` block, the following directive would be specified in the body '`(list "proxy_pass http://upstream-name;")`'.

`nginx-named-location-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de un bloque de localización con nombre de nginx. Los bloques de localizaciones con nombre se usan para la redirección de peticiones, y no se usan para el procesamiento regular de peticiones. Este tipo tiene los siguientes parámetros:

`name` Nombre que identifica este bloque de dirección `location`.

`body` See [cuerpo de nginx-location-configuration], page 476, como el cuerpo de los bloques de localizaciones con nombre puede usarse de manera similar al `cuerpo de nginx-location-configuration`. Una restricción es que el cuerpo de una localización con nombre no puede contener bloques de localizaciones.

### Caché Varnish

Varnish es un servidor de caché rápida que se coloca entre aplicaciones web y usuarios finales. Redirige peticiones a los clientes y almacena en caché las URL a las que se accede de manera que múltiples peticiones al mismo recurso únicamente creen una petición al motor.

`varnish-service-type` [Variable]

Tipo de servicio para el daemon Varnish.

`varnish-configuration` [Tipo de datos]

Tipo de datos que representa la configuración del servicio `varnish`. Este tipo tiene los siguientes parámetros:

`package` (predeterminado: `varnish`)
: El paquete Varnish usado.

`name` (predeterminado: `"default"`)
: Un nombre para esta instancia de Varnish. Varnish creará un directorio en `/var/varnish` con este nombre y mantendrá allí los archivos temporales. Si el nombre comienza con una barra, se interpreta como un nombre absoluto de directorio.

  Proporcione el parámetro `-n` a otros programas de Varnish para que se conecten a la instancia de dicho nombre, por ejemplo `varnishncsa -n default`.

`backend` (predeterminado: `"localhost:8080"`)
: Motor usado. Esta opción no tiene efecto si se usa `vcl`.

`vcl` (predeterminado: #f)
: El programa *VCL* (lenguaje de configuración de Varnish) ejecutado. Si se proporciona `#f`, Varnish llevará a cabo las redirecciones al motor (`backend`) usando la configuración predeterminada. En otro caso debe ser un objeto "tipo-archivo" con sintaxis válida para VCL.

  Por ejemplo, para proporcionar un espejo de www.gnu.org (`https://www.gnu.org`) con VCL podría escribir algo parecido a esto:

```
(define %espejo-gnu
  (plain-file "gnu.vcl"
              "vcl 4.1;
backend gnu { .host = \"www.gnu.org\"; }"))

(operating-system
  ;; ...
  (services (cons (service varnish-service-type
                           (varnish-configuration
                            (listen '(":80"))
                            (vcl %espejo-gnu)))
                  %base-services)))
```

  La configuración de una instancia de Varnish ya en ejecución se puede inspeccionar y cambiar mediante el uso de la orden `varnishadm`.

Consulte la guía de usuaria de Varnish (`https://varnish-cache.org/docs/`) y el libro de Varnish (`https://book.varnish-software.com/4.0/`) para obtener la documentación completa de Varnish y su lenguaje de configuración.

`listen` (predeterminada: `'("localhost:80")`)
Lista de direcciones en las que Varnish escucha.

`storage` (predeterminado: `'("malloc,128m")`)
Lista de motores de almacenamiento que estarán disponibles en VCL.

`parameters` (predeterminados: `'()`)
Lista de parámetros de tiempo de ejecución con la forma `'(("parámetro" . "valor"))`.

`extra-options` (predeterminadas: `'()`)
Parámetros adicionales a proporcional al proceso `varnishd`.

## Whoogle Search

Whoogle Search (`https://github.com/benbusby/whoogle-search`) is a self-hosted, ad-free, privacy-respecting meta search engine that collects and displays Google search results. By default, you can configure it by adding this line to the `services` field of your operating system declaration:

```
(service whoogle-service-type)
```

As a result, Whoogle Search runs as local Web server, which you can access by opening '`http://localhost:5000`' in your browser. The configuration reference is given below.

`whoogle-service-type` [Variable]
Service type for Whoogle Search. Its value must be a `whoogle-configuration` record—see below.

`whoogle-configuration` [Data Type]
Data type representing Whoogle Search service configuration.

`package` (default: `whoogle-search`)
The Whoogle Search package to use.

`host` (predeterminada: `"127.0.0.1"`)
The host address to run Whoogle on.

`port` (default: `5000`)
The port where Whoogle will be exposed.

`environment-variables` (default: `'()`)
A list of strings with the environment variables to configure Whoogle. You can consult its environment variables template (`https://github.com/benbusby/whoogle-search/blob/main/whoogle.template.env`) for the list of available options.

## Patchwork

Patchwork es un sistema de seguimiento de parches. Puede recolectar parches enviados a listas de correo y mostrarlos en una interfaz web.

`patchwork-service-type` [Variable]
Tipo de servicio para Patchwork.

El siguiente ejemplo muestra un servicio mínimo para Patchwork, para el dominio `patchwork.example.com`.

```
(service patchwork-service-type
         (patchwork-configuration
          (domain "patchwork.example.com")
          (settings-module
           (patchwork-settings-module
            (allowed-hosts (list domain))
            (default-from-email "patchwork@patchwork.example.com")))
          (getmail-retriever-config
           (getmail-retriever-configuration
            (type "SimpleIMAPSSLRetriever")
            (server "imap.example.com")
            (port 993)
            (username "patchwork")
            (password-command
             (list (file-append coreutils "/bin/cat")
                   "/etc/getmail-patchwork-imap-password"))
            (extra-parameters
            '((mailboxes . ("Parches")))))))))
```

Existen tres registros para la configuración del servicio de Patchwork. El registro `<patchwork-configuration>` está relacionado con la configuración de Patchwork dentro del servicio HTTPD.

El campo `settings-module` dentro del registro `<patchwork-configuration>` puede rellenarse con un registro `<patchwork-settings-module>`, que describe un módulo de configuración generado dentro del almacén de Guix.

En el campo `database-configuration` dentro del registro `<patchwork-settings-module>`, debe usarse `<patchwork-database-configuration>`.

`patchwork-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del servicio Patchwok. Este tipo tiene los siguientes parámetros:

`patchwork` (predeterminado: `patchwork`)
El paquete Patchwork usado.

`domain` Dominio usado por Patchwork, se usa en el servicio HTTPD como “virtual host”.

`settings-module`
: The settings module to use for Patchwork. As a Django application, Patchwork is configured with a Python module containing the settings. This can either be an instance of the `<patchwork-settings-module>` record, any other record that represents the settings in the store, or a directory outside of the store.

`static-path` (predeterminada: `"/static/"`)
: Ruta bajo la cual el servicio HTTPD proporciona archivos estáticos.

`getmail-retriever-config`
: The getmail-retriever-configuration record value to use with Patchwork. Getmail will be configured with this value, the messages will be delivered to Patchwork.

`patchwork-settings-module` [Tipo de datos]
Tipo de datos que representa un módulo de configuración de Patchwork. Algunas de estas opciones están directamente relacionadas con Patchwork, pero otras son relativas a Django, el entorno web usado Patchwork, o la biblioteca Django Rest Framework. Este tipo tiene los siguientes parámetros:

`database-configuration` (predeterminada: `(patchwork-database-configuration)`)
: La configuración de la conexión a la base de datos usada para Patchwork. Véase el tipo de registro `<patchwork-database-configuration>` para más información.

`secret-key-file` (predeterminado: `"/etc/patchwork/django-secret-key"`)
: Patchwork, como una aplicación web Django, usa una clave secreta para firmar criptográficamente valores. Este archivo debe contener un valor único e impredecible.

  Si este archivo no existe, el servicio de Shepherd patchwork-setup lo creará y rellenará con un valor aleatorio.

  Esta configuración está relacionada con Django.

`allowed-hosts`
: A list of valid hosts for this Patchwork service. This should at least include the domain specified in the `<patchwork-configuration>` record.

  Esta es una opción de configuración de Django.

`default-from-email`
: La dirección de correo desde de la que Patchwork debe enviar el correo de manera predeterminada.

  Esta es una opción de configuración de Patchwork.

`static-url` (predeterminada: `#f`)
: The URL to use when serving static assets. It can be part of a URL, or a full URL, but must end in a `/`.

  Si se usa el valor predeterminado, se usará el valor de `static-path` del registro `<patchwork-configuration>`.

  Esta es una opción de configuración de Django.

`admins` (predeterminadas: `'()`)
: Direcciones de correo electrónico a las que enviar los detalles de los errores que ocurran. Cada valor debe ser una lista que contenga dos elementos, el nombre y la dirección de correo electrónico en dicho orden.

  Esta es una opción de configuración de Django.

`debug?` (predeterminado: `#f`)
: Determina si se ejecuta Patchwork en modo de depuración. Si se proporciona `#t`, se mostrarán mensajes de error detallados.

  Esta es una opción de configuración de Django.

`enable-rest-api?` (predeterminado: `#t`)
: Determina si se activa el API REST de Patchwork.

  Esta es una opción de configuración de Patchwork.

`enable-xmlrpc?` (predeterminado: `#t`)
: Determina si se activa el API XML RPC.

  Esta es una opción de configuración de Patchwork.

`force-https-links?` (predeterminado: `#t`)
: Determina si se usan enlaces HTTPS en las páginas de Patchwork.

  Esta es una opción de configuración de Patchwork.

`extra-settings` (predeterminado: `""`)
: Código adicional que colocar al final del módulo de configuración de Patchwork.

`patchwork-database-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de base de datos de Patchwork.

`engine` (predeterminado: `"django.db.backends.postgresql_psycopg2"`)
: Motor de base de datos usado.

`name` (predeterminado: `"patchwork"`)
: Nombre de la base de datos usada.

`user` (predeterminada: `"httpd"`)
: Usuaria usada para la conexión a la base de datos.

`password` (predeterminada: `""`)
: Contraseña usada para la conexión a la base de datos.

`host` (predeterminada: `""`)
: Máquina usada para la conexión a la base de datos.

`port` (predeterminado: `""`)
: Puerto en el que se conecta a la base de datos.

## Mumi

Mumi (`https://git.savannah.gnu.org/cgit/guix/mumi.git/`) is a Web interface to the Debbugs bug tracker, by default for the GNU instance (`https://bugs.gnu.org`). Mumi is a Web server, but it also fetches and indexes mail retrieved from Debbugs.

`mumi-service-type` [Variable]
El tipo de servicio para Mumi.

`mumi-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del servicio Mumi. Este tipo tiene los siguientes campos:

`mumi` (predeterminado: `mumi`)
El paquete Mumi usado.

`mailer?` (predeterminado: `#true`)
Determina si se activa o desactiva el componente de correo `mailer`.

`mumi-configuration-sender`
La dirección de correo usada como remitente para los comentarios.

`mumi-configuration-smtp`
Una URI para las opciones de configuración de SMTP de Mailutils. Puede ser algo parecido a `sendmail:///ruta/de/bin/msmtp` o cualquier otra URI implementada por Mailutils. See Section "SMTP Mailboxes" in *GNU Mailutils*.

## FastCGI

FastCGI es una interfaz entre la presentación (front-end) y el motor (back-end) de un servicio web. Es en cierto modo una característica antigua; los nuevos servicios web generalmente únicamente se comunican con HTTP entre ambas partes. No obstante, existe cierto número de servicios de motor como PHP o el acceso HTTP optimizado para repositorios Git que usan FastCGI, por lo que debemos incluirlo en Guix.

Para usar FastCGI debe configurar el servidor web de entrada[10] (por ejemplo, ngnix) para delegar un subconjunto de sus peticiones al motor fastcgi, que escucha en un puerto TCP local o en un socket de UNIX. Existe un programa de intermediación llamado `fcgiwrap` que se posiciona entre el proceso del motor y el servidor web. El servidor indica el programa del motor usado, proporcionando dicha información al proceso `fcgiwrap`.

`fcgiwrap-service-type` [Variable]
El tipo de servicio para la pasarela FastCGI `fcgiwrap`.

`fcgiwrap-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del servicio `fcgiwrap`. Este tipo tiene los siguientes parámetros:

`package` (predeterminado: `fcgiwrap`)
El paquete fcgiwrap usado.

`socket` (predeterminado: `tcp:127.0.0.1:9000`)
El socket donde el proceso `fcgiwrap` deba escuchar, como una cadena. Los valores adecuados para *socket* socket incluyen `unix:`*`/ruta/al/socket/unix`*, `tcp:`*`dirección.ip.con.puntos`*`:`*`puerto`* and `tcp6:[`*`dirección_ipv6`*`]:puerto`.

[10] NdT: Front-end en inglés.

`user` (predeterminado: `fcgiwrap`)
`group` (predeterminado: `fcgiwrap`)
: Los nombres de usuaria y grupo, como cadenas, con los que se ejecutará el proceso `fcgiwrap`. El servicio `fastcgi` se asegura, en caso de solicitar específicamente el uso de nombres de usuaria o grupo `fcgiwrap`, que la usuaria y/o grupo correspondientes se encuentren presentes en el sistema.

  Es posible configurar un servicio web proporcionado por FastCGI para que el servidor de fachada proporcione la información de identificación HTTP al motor, y para permitir que `fcgiwrap` se ejecute en el proceso del motor como la usuaria local correspondiente. Para activar esta funcionalidad en el motor, ejecute `fcgiwrap` mediante la usuaria y grupo `root`. Tenga en cuenta de que esta funcionalidad debe configurarse del mismo modo en el servidor de fachada.

## PHP-FPM

PHP-FPM (FastCGI Process Manager) es una implementación alternativa de FastCGI en PHP con algunas características adicionales útiles para sitios de cualquier tamaño.

Estas características incluyen:

- Lanzamiento adaptativo de procesos
- Estadísticas básicas (similares a mod_status de Apache)
- Gestión avanzada de procesos con parada/arranque coordinados
- Capacidad de iniciar procesos de trbajo con diferentes uid/gid/chroor/entorno y diferentes php.ini (reemplaza a safe_mode)
- Registro a través de la salida estándar y de error
- Reinicio de emergencia en caso de destrucción accidental de la caché de opcode
- Posibilidad de subida acelerada
- Posibilidad de un "slowlog"
- Mejoras a FastCGI, como fastcgi_finish_request() - una función especial para terminar una petición y enviar todos los datos mientras que se continua haciendo una tarea de alto consumo de tiempo (conversión de datos audiovisuales, procesamiento de estadísticas, etcétera).

... y muchas más.

`php-fpm-service-type` [Variable]
: Un tipo de servicio para `php-fpm`.

`php-fpm-configuration` [Tipo de datos]
: Tipo de datos para la configuración del servicio php-fpm.

  `php` (predeterminado: `php`)
  : El paquete php usado.

  `socket` (predeterminado: `(string-append "/var/run/php" (version-major (package-version php)) "-fpm.sock")`)
  : La dirección desde la que FastCGI acepta peticiones. Las sintaxis válidas son:

`"dir.ecc.ión.ip:puerto"`
: Escucha con un socket TCP en la dirección especificada en un puerto específico.

`"puerto"`
: Escucha en un socket TCP en todas las direcciones sobre un puerto específico.

`"/ruta/a/socket/unix"`
: Escucha en un socket Unix.

`user` (predeterminada: `php-fpm`)
: Usuaria que poseerá los procesos de trabajo de php.

`group` (predeterminado: `php-fpm`)
: Grupo de los procesos de trabajo.

`socket-user` (predeterminado: `php-fpm`)
: Usuaria que puede comunicarse con el socket de php-fpm.

`socket-group` (predeterminado: `nginx`)
: Grupo que puede comunicarse con el socket de php-fpm.

`pid-file` (predeterminado: `(string-append "/var/run/php" (version-major (package-version php)) "-fpm.pid")`)
: El identificador de proceso del proceso de php-fpm se escribe en este archivo cuando se ha iniciado el servicio.

`log-file` (predeterminado: `(string-append "/var/log/php" (version-major (package-version php)) "-fpm.log")`)
: Registro del proceso maestro de php-fpm.

`process-manager` (predeterminado: `(php-fpm-dynamic-process-manager-configuration)`)
: Configuración detallada para el gestor de procesos php-fpm. Debe ser uno de los siguientes tipos:

```
<php-fpm-dynamic-process-manager-configuration>
<php-fpm-static-process-manager-configuration>
<php-fpm-on-demand-process-manager-configuration>
```

`display-errors` (predeterminado `#f`)
: Determina si los errores y avisos de php deben enviarse a los clientes para que se muestren en sus navegadores. Esto es útil para la programación local con php, pero un riesgo para la seguridad de sitios públicos, ya que los mensajes de error pueden revelar contraseñas y datos personales.

`timezone` (predeterminado: `#f`)
: Especifica el parámetro `php_admin_value[date.timezone]`.

`workers-logfile` (predeterminado `(string-append "/var/log/php" (version-major (package-version php)) "-fpm.www.log")`)
: Este archivo registrará las salidas por `stderr` de los procesos de trabajo de php. Puede proporcionarse `#f` para desactivar el registro.

`file` (predeterminado `#f`)
: Sustituye opcionalmente la configuración al completo. Puede usar la función `mixed-text-file` o una ruta absoluta de un archivo para hacerlo.

`php-ini-file` (predeterminado: `#f`)
: Sustituye opcionalmente la configuración predeterminada de php. Puede ser cualquier objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167). Puede usar la función `mixed-text-file` o una ruta absoluta de un archivo para hacerlo.

  Para el desarrollo local es útil proporcionar valores mayores para los plazos y límites de memoria de los procesos php lanzados. Esto puede obtenerse con el siguiente fragmento de la configuración de sistema operativo:

```
(define %php-ini-local
  (plain-file "php.ini"
              "memory_limit = 2G
max_execution_time = 1800"))

(operating-system
  ;; ...
  (services (cons (service php-fpm-service-type
                           (php-fpm-configuration
                            (php-ini-file %php-ini-local)))
                  %base-services)))
```

  Consulte las directivas principales de php.ini (`https://www.php.net/manual/en/ini.core.php`) para obtener una documentación extensa de las directivas aceptables en el archivo `php.ini`.

`php-fpm-dynamic-process-manager-configuration` [Tipo de datos]
Data Type for the `dynamic` php-fpm process manager. With the `dynamic` process manager, spare worker processes are kept around based on its configured limits.

`max-children` (predeterminados: `5`)
: Número máximo de procesos de trabajo.

`start-servers` (predeterminados: `2`)
: Cuantos procesos de trabajo deben ejecutarse al inicio.

`min-spare-servers` (predeterminado: `1`)
: Cuantos procesos de trabajo deben mantenerse disponibles como mínimo.

`max-spare-servers` (predeterminados: `3`)
: Cuantos procesos de trabajo deben mantenerse disponibles como máximo.

`php-fpm-static-process-manager-configuration` [Tipo de datos]
Tipo de datos para el gestor de procesos `static` de php-fpm. Con el gestor de procesos `static`, se crea un número fijo de procesos de trabajo.

`max-children` (predeterminados: `5`)
: Número máximo de procesos de trabajo.

`php-fpm-on-demand-process-manager-configuration` [Tipo de datos]
Tipo de datos para el gestor de procesos `on-demand` de php-fpm. Con el gestor de procesos `on-demand`, se crean procesos de trabajo únicamente cuando se reciben peticiones.

`max-children` (predeterminados: `5`)
Número máximo de procesos de trabajo.

`process-idle-timeout` (predeterminado: `10`)
El tiempo en segundos tras el cual un proceso sin peticiones será eliminado.

`nginx-php-location` [*#:nginx-package nginx*] [*socket* (*string-append* "*/var/run/php*" (*version-major* (*package-version* php)) "-fpm.sock")] [Procedimiento]
Función auxiliar para añadir php a una configuración `nginx-server-configuration` rápidamente.

Una configuración simple de servicios para nginx con php puede ser más o menos así:

```
(services (cons* (service dhcp-client-service-type)
                 (service php-fpm-service-type)
                 (service nginx-service-type
                          (nginx-server-configuration
                           (server-name '("example.com"))
                           (root "/srv/http/")
                           (locations
                            (list (nginx-php-location)))
                           (listen '("80"))
                           (ssl-certificate #f)
                           (ssl-certificate-key #f)))
                 %base-services))
```

El generadores de avatares de gato es un servicio simple para demostrar el uso de php-fpm en `Nginx`. Se usa para generar un avatar de gato desde una semilla, por ejemplo el hash de la dirección de correo de la usuaria.

`cat-avatar-generator-service` [*#:cache-dir* "*/var/cache/cat-avatar-generator*"] [*#:package* cat-avatar-generator] [#:configuration (nginx-server-configuration)] [Procedimiento]
Devuelve una configuración de nginx-server-configuration que hereda de `configuration`. Extiende la configuración de nginx para añadir un bloque de servidor que proporciona `package`, una versión de cat-avatar-generator. Durante su ejecución, cat-avatar-generator podrá usar `cache-dir` como su directorio de caché.

Una configuración simple para cat-avatar-generator puede ser más o menos así:

```
(services (cons* (cat-avatar-generator-service
                  #:configuration
                  (nginx-server-configuration
                    (server-name '("example.com"))))
                 ...
                 %base-services))
```

## Hpcguix-web

El programa hpcguix-web (`https://github.com/UMCUGenetics/hpcguix-web/`) es una interfaz web personalizable para buscar paquetes de Guix, diseñado inicialmente para usuarias de clusters de computación de alto rendimiento (HPC).

`hpcguix-web-service-type` [Variable]
El tipo de servicio para `hpcguix-web`.

`hpcguix-web-configuration` [Tipo de datos]
El tipo de datos para la configuración del servicio hpcguix-web.

`specs` (default: `#f`)
Either `#f` or a gexp (see Section 8.12 [Expresiones-G], page 167) specifying the hpcguix-web service configuration as an `hpcguix-web-configuration` record. The main fields of that record type are:

`title-prefix` (predeterminado: `"hpcguix | "`)
El prefijo del título de la página.

`guix-command` (predeterminada: `"guix"`)
The `guix` command to use in examples that appear on HTML pages.

`package-filter-proc` (predeterminado: `(const #t)`)
Un procedimiento que especifica cómo filtrar los paquetes mostrados.

`package-page-extension-proc` (predeterminado: `(const '())`)
Paquete de extensión para `hpcguix-web`.

`menu` (predeterminadas: `'()`)
Entradas adicionales en el menú de la página.

`channels` (predeterminados: `%default-channels`)
Lista de canales desde los que se construye la lista de paquetes (see Chapter 6 [Canales], page 69).

`package-list-expiration` (predeterminado: `(* 12 3600)`)
El tiempo de expiración, en segundos, tras el cual la lista de paquetes se reconstruye desde las últimas instancias de los canales proporcionados.

Véase el repositorio de hpcguix-web para un ejemplo completo (`https://github.com/UMCUGenetics/hpcguix-web/blob/master/hpcweb-configuration.scm`).

`package` (predeterminado: `hpcguix-web`)
El paquete hpcguix-web usado.

`address` (default: `"127.0.0.1"`)
The IP address to listen to.

`port` (default: `5000`)
The port number to listen to.

Una declaración típica del servicio hpcguix-web es más o menos así:

```
(service hpcguix-web-service-type
         (hpcguix-web-configuration
          (specs
           #~(hpcweb-configuration
              (title-prefix "Guix-HPC - ")
              (menu '(("/about" "ABOUT")))))))
```

**Nota:** El servicio hpcguix-web actualiza periódicamente la lista de paquetes que publica obteniendo canales con Git. Para ello, necesita acceder a certificados X.509 de manera que pueda validar los servidores Git durante la comunicación con HTTPS, y asume que `/etc/ssl/certs` contiene dichos certificados.

A certificate package, `nss-certs`, is provided by default as part of `%base-packages`. Section 11.12 [Certificados X.509], page 621, for more information on X.509 certificates.

### gmnisrv

El programa gmnisrv (`https://git.sr.ht/~sircmpwn/gmnisrv`) es un servidor simple del protocolo Gemini (`https://gemini.circumlunar.space/`).

`gmnisrv-service-type` [Variable]

Es el tipo del servicio gmnisrv, cuyo valor debe ser un objeto `gmniserv-configuration` como en este ejemplo:

```
(service gmnisrv-service-type
         (gmnisrv-configuration
           (config-file (local-file "./mi-gmnisrv.ini"))))
```

`gmnisrv-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de gmnisrv.

`package` (predeterminado: *gmnisrv*)
: El objeto paquete del servidor gmnisrv.

`config-file` (predeterminado: `%default-gmnisrv-config-file`)
: Objeto tipo-archivo del archivo de configuración de gmnisrv usado. La configuración predeterminada escucha en el puerto 1965 y proporciona archivos desde `/srv/gemini`. Los certificados se almacenan en `/var/lib/gemini/certs`. Puede ejecutar las ordenes `man gmnisrv` y `man gmnisrv.ini` para obtener más información.

### Agate

The Agate (`gemini://qwertqwefsday.eu/agate.gmi`) (GitHub page over HTTPS (`https://github.com/mbrubeck/agate`)) program is a simple Gemini (`https://gemini.circumlunar.space/`) protocol server written in Rust.

`agate-service-type` [Variable]

This is the type of the agate service, whose value should be an `agate-service-type` object, as in this example:

```
(service agate-service-type
```

```
(agate-configuration
  (content "/srv/gemini")
  (cert "/srv/cert.pem")
  (key "/srv/key.rsa")))
```

The example above represents the minimal tweaking necessary to get Agate up and running. Specifying the path to the certificate and key is always necessary, as the Gemini protocol requires TLS by default.

To obtain a certificate and a key, you could, for example, use OpenSSL, running a command similar to the following example:

```
openssl req -x509 -newkey rsa:4096 -keyout key.rsa -out cert.pem \
    -days 3650 -nodes -subj "/CN=example.com"
```

Of course, you'll have to replace *example.com* with your own domain name, and then point the Agate configuration towards the path of the generated key and certificate.

`agate-configuration` [Data Type]

Data type representing the configuration of Agate.

`package` (default: `agate`)
: The package object of the Agate server.

`content` (default: `"/srv/gemini"`)
: The directory from which Agate will serve files.

`cert` (default: `#f`)
: The path to the TLS certificate PEM file to be used for encrypted connections. Must be filled in with a value from the user.

`key` (predeterminada: `#f`)
: The path to the PKCS8 private key file to be used for encrypted connections. Must be filled in with a value from the user.

`addr` (default: `'("0.0.0.0:1965" "[::]:1965")`)
: A list of the addresses to listen on.

`hostname` (predeterminado: `#f`)
: The domain name of this Gemini server. Optional.

`lang` (default: `#f`)
: RFC 4646 language code(s) for text/gemini documents. Optional.

`silent?` (default: `#f`)
: Set to `#t` to disable logging output.

`serve-secret?` (default: `#f`)
: Set to `#t` to serve secret files (files/directories starting with a dot).

`log-ip?` (default: `#t`)
: Whether or not to output IP addresses when logging.

`user` (default: `"agate"`)
: Owner of the `agate` process.

`group` (default: `"agate"`)
: Owner's group of the `agate` process.

`log-file` (default: `"/var/log/agate.log"`)
The file which should store the logging output of Agate.

### 11.10.21 Servicios de certificados

El módulo `(gnu services certbot)` proporciona un servicio para la obtención automática de un certificado TLS válido de la autoridad de certificación Let's Encrypt. Estos certificados pueden usarse para proporcionar contenido de forma segura sobre HTTPS u otros protocolos basados en TLS, con el conocimiento de que el cliente podrá verificar la autenticidad del servidor.

Let's Encrypt (`https://letsencrypt.org/`) provides the `certbot` tool to automate the certification process. This tool first securely generates a key on the server. It then makes a request to the Let's Encrypt certificate authority (CA) to sign the key. The CA checks that the request originates from the host in question by using a challenge-response protocol, requiring the server to provide its response over HTTP. If that protocol completes successfully, the CA signs the key, resulting in a certificate. That certificate is valid for a limited period of time, and therefore to continue to provide TLS services, the server needs to periodically ask the CA to renew its signature.

El servicio certbot automatiza este proceso: la generación inicial de la clave, la petición inicial de certificación al servicio Let's Encrypt, la integración del desafío/respuesta en el servidor web, la escritura del certificado en disco, las renovaciones periódicas automáticas y el despliegue de tareas asociadas con la renovación (por ejemplo la recarga de servicios y la copia de claves con diferentes permisos).

Certbot se ejecuta dos veces al día, en un minuto aleatorio dentro de la hora. No hará nada hasta que sus certificados estén pendientes de renovación o sean revocados, pero su ejecución regular propociona a su servicio la oportunidad de permanecer en línea en caso de que se produzca una revocación iniciada por Let's Encrypt por alguna razón.

Mediante el uso de este servicio, usted acepta el acuerdo de suscripción ACME, que se puede encontrar aquí: `https://acme-v01.api.letsencrypt.org/directory`.

`certbot-service-type` [Variable]
Un tipo de servicio para el cliente de Let's Encrypt `certbot`. Su valor debe ser un registro `certbot-configuration` como en este ejemplo:

```
(service certbot-service-type
         (certbot-configuration
          (email "foo@example.net")
          (certificates
           (list
            (certificate-configuration
             (domains '("example.net" "www.example.net")))
            (certificate-configuration
             (domains '("bar.example.net")))))))
```

Véase a continuación los detalles de `certbot-configuration`.

`certbot-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del servicio `certbot`. Este tipo tiene los siguientes parámetros:

`package` (predeterminado: `certbot`)
: El paquete certbot usado.

`webroot` (predeterminado: `/var/www`)
: Directorio desde el que se proporcionan los archivos de desafío/respuesta de Let's Encrypt.

`certificates` (predeterminados: `'()`)
: Una lista de configuraciones `certificates-configuration` para los cuales se generan certificados y se solicitan firmas. Cada certificado tiene un nombre (`name`) y varios dominios (`domains`).

`email` (predeterminado: `#f`)
: Dirección de correo electrónico opcional usada para el registro y el contacto de recuperación. Se recomienda que proporcione un valor ya que le permite recibir importantes notificaciones acerca de la cuenta y los certificados emitidos.

`server` (predeterminada: `#f`)
: URL opcional del servidor ACME. Esta configuración cambia el valor predeterminado de certbot, que es el servidor de Let's Encrypt.

`rsa-key-size` (predeterminado: `2048`)
: Tamaño de la clave RSA.

`default-location` (predeterminada: *véase a continuación*)
: La configuración `nginx-location-configuration` predeterminada. Debido a que `certbot` necesita proporcionar desafíos y respuestas, necesita ser capaz de ejecutar un servidor web. Se lleva a cabo extendiendo el servicio web `nginx` con una configuración `nginx-server-configuration` que escucha en los dominios *domains* en el puerto 80, y que contiene una configuración `nginx-location-configuration` para el subespacio de rutas URI `/.well-known/` usado por Let's Encrypt. See Section 11.10.20 [Servicios Web], page 469, para más información sobre estos tipos de datos de configuración de nginx.

  Las peticiones a otras rutas URL se compararán contra la dirección predeterminada `default-location`, la cual, en caso de estar presente, se añade a todas las configuraciones `nginx-server-configuration`.

  De manera predeterminada, la dirección predeterminada `default-location` emitirá una redirección `http://`*`dominio`*`/...` a `https://`*`dominio`*`/...`, lo que le permite definir qué proporcionará en su sitio web a través de `https`.

  Proporcione `#f` para no emitir una dirección predeterminada.

`certificate-configuration` [Tipo de datos]
: Tipo de datos que representa la configuración de un certificado. Este tipo tiene los siguientes parámetros:

`name` (predeterminado: *vea a continuación*)
: Este nombre se usa por Certbot para su mantenimiento interno y en las rutas de archivos; no afecta al contenido del certificado en sí mismo. Para ver los nombres de certificados, ejecute `certbot certificates`.

  Su valor predeterminado es el primer dominio proporcionado.

`domains` (predeterminado: `'()`)
: El primer dominio proporcionado será el sujeto del nombre común (CN) del certificado, y todos los dominios serán nombres alternativos (Subject Alternative Names) en el certificado.

`challenge` (predeterminado: `#f`)
: El tipo de desafío que debe ejecutar certbot. Si se especifica `#f`, el valor por omisión es desafío HTTP. Si se especifica un valor, el valor por omisión es el módulo manual (véase `authentication-hook`, `cleanup-hook` y la documentación en `https://certbot.eff.org/docs/using.html#hooks`), y concede permiso a Let's Encrypt para registrar la IP pública de la máquina que realiza la petición.

`csr` (default: `#f`)
: File name of Certificate Signing Request (CSR) in DER or PEM format. If `#f` is specified, this argument will not be passed to certbot. If a value is specified, certbot will use it to obtain a certificate, instead of using a self-generated CSR. The domain-name(s) mentioned in `domains`, must be consistent with the domain-name(s) mentioned in CSR file.

`authentication-hook` (predeterminado: `#t`)
: Orden ejecutada en un shell una vez por cada desafío de certificado que debe contestarse. Durante su ejecución, la variable del shell `$CERTBOT_DOMAIN` contiene el dominio que se está validando, `$CERTBOT_VALIDATION` contiene la cadena de validación y `$CERTBOT_TOKEN` contiene el nombre de archivo del recurso solicitado cuando se realiza el desafío HTTP-01.

`cleanup-hook` (predeterminado: `#f`)
: Orden ejecutada en un shell una vez por cada desafío de certificado que haya sido contestado por `auth-hook`. Durante su ejecución, las variables del shell disponibles en el script `auth-hook` todavía están disponibles, y adicionalmente `$CERTBOT_AUTH_OUTPUT` contendrá la salida estándar que produjo `auth-hook`.

`deploy-hook` (predeterminado: `#f`)
: Orden ejecutada en un shell una vez por cada certificado emitido satisfactoriamente. Durante su ejecución, la variable del shell `$RENEWED_LINEAGE` apuntará al subdirectorio live de configuración (por ejemplo, ‘`"/etc/letsencrypt/live/example.com"`’) que contiene las nuevas claves y certificados; la variable del shell `$RENEWED_DOMAINS` contendrá una lista delimitada por espacios de certificados de dominio renovados (por ejemplo, ‘`"example.com www.example.com"`’).

`start-self-signed?` (default: `#t`)
: Whether to generate an initial self-signed certificate during system activation. This option is particularly useful to allow `nginx` to start before `certbot` has run, because `certbot` relies on `nginx` running to perform HTTP challenges.

Para cada configuración `certificate-configuration`, el certificado se almacena `/etc/certs/`*name*`/fullchain.pem` y la clave se almacena en `/etc/certs/`*name*`/privkey.pem`.

### 11.10.22 Servicios DNS

El módulo `(gnu services dns)` proporciona servicios relacionados con el *sistema de nombres de dominio* (DNS). Proporciona un servicio de servidor para el alojamiento de un servidor *autorizado* DNS para múltiples zonas, esclavo o maestro. Este servicio usa Knot DNS (`https://www.knot-dns.cz/`). Y también un servidor de caché y reenvío de DNS para la red local, que usa dnsmasq (`http://www.thekelleys.org.uk/dnsmasq/doc.html`).

#### Servicio Knot

Esta es una configuración de ejemplo de un servidor de autoridad para dos zonas, una maestra y otra esclava:

```
(define-zone-entries example.org.zone
;; Name TTL Class Type Data
  ("@"  ""  "IN"  "A"  "127.0.0.1")
  ("@"  ""  "IN"  "NS" "ns")
  ("ns" ""  "IN"  "A"  "127.0.0.1"))

(define master-zone
  (knot-zone-configuration
    (domain "example.org")
    (zone (zone-file
            (origin "example.org")
            (entries example.org.zone)))))

(define slave-zone
  (knot-zone-configuration
    (domain "plop.org")
    (dnssec-policy "default")
    (master (list "plop-master"))))

(define plop-master
  (knot-remote-configuration
    (id "plop-master")
    (address (list "208.76.58.171"))))

(operating-system
  ;; ...
```

```
(services (cons* (service knot-service-type
                  (knot-configuration
                    (remotes (list plop-master))
                    (zones (list master-zone slave-zone))))
                 ;; ...
                 %base-services)))
```

`knot-service-type` [Variable]

Este es el tipo de datos para el servidor DNS Knot.

Knot DNS es un servidor de autoridad de DNS, lo que significa que puede servir múltiples zonas, es decir, nombres de dominio que compraría a una autoridad de registro de nombres. Este servidor no es un resolvedor, lo que significa que sólo puede resolver nombres para los que tiene autoridad. Este servidor puede configurarse para servir zonas como servidor maestro o como servidor esclavo con una granularidad al nivel de zona. Las zonas esclavas obtendrán sus datos de los servidores maestros, y las proporcionarán como un servidor de autoridad. Desde el punto de vista de un resolvedor, no hay diferencia entre servidor maestro y esclavo.

Los siguientes tipos de datos se usan para configurar el servidor DNS Knot:

`knot-key-configuration` [Tipo de datos]

Tipo de datos que representa una clave. Este tipo tiene los siguientes parámetros:

`id` (predeterminado: `""`)
: An identifier for other configuration fields to refer to this key. IDs must be unique and must not be empty.

`algorithm` (predeterminado: `#f`)
: El algoritmo usado. Debe seleccionarse entre `#f`, `'hmac-md5`, `'hmac-sha1`, `'hmac-sha224`, `'hmac-sha256`, `'hmac-sha384` y `'hmac-sha512`.

`secret` (predeterminado: `""`)
: La clave secreta en sí.

`knot-acl-configuration` [Tipo de datos]

Tipo de datos que representa una configuración de lista de control de acceso (ACL). Este tipo tiene los siguientes parámetros:

`id` (predeterminado: `""`)
: An identifier for other configuration fields to refer to this key. IDs must be unique and must not be empty.

`address` (predeterminada: `'()`)
: Lista ordenada de direcciones IP, subredes o rangos de red representadas como cadenas. La búsqueda debe corresponder con alguna. El valor vacío significa que la comprobación de correspondencia de la dirección no es necesaria.

`key` (predeterminada: `'()`)
: Lista ordenada de referencias a claves representadas como cadenas. La cadena debe corresponder con un ID de clave definido en `knot-key-configuration`. Ninguna clave significa que la comprobación de claves no es necesaria para este control de acceso (ACL).

`action` (predeterminada: `'()`)
: An ordered list of actions that are permitted or forbidden by this ACL. Possible values are lists of zero or more elements from `'transfer`, `'notify` and `'update`.

`deny?` (predeterminado: `#f`)
: Cuando es verdadero, este ACL define restricciones. Las acciones enumeradas no se permiten. Cuando es falso, las acciones enumeradas se permiten.

`zone-entry` [Tipo de datos]
Tipo de datos que representa una entrada de registro en un archivo de zona. Este tipo tiene los siguientes parámetros:

`name` (predeterminado: `"@"`)
: El nombre del registro. `"@"` hace referencia al origen de la zona. Los nombres son relativos al origen de la zona. Por ejemplo, en la zona `example.org`, `"ns.example.org"` en realidad hace referencia a `ns.example.org.example.org`. Los nombres que terminan en un punto se consideran absolutos, lo que significa que `"ns.example.org."` hace referencia a `ns.example.org`.

`ttl` (predeterminado: `""`)
: El tiempo de vida (TTL) de este registro. Si no se proporciona, se usa el TTL predeterminado.

`class` (predeterminada: `"IN"`)
: La clase del registro. Actualmente Knot implementa únicamente `"IN"` y parcialmente `"CH"`.

`type` (predeterminado: `"A"`)
: El tipo del registro. Los tipos comunes incluyen A (dirección IPv4), AAAA (dirección IPv6), NS (servidor de nombres[11]) y MX (pasarela de correo[12]). Otros muchos tipos distintos se encuentran definidos.

`data` (predeterminados: `""`)
: Los datos que contiene el registro. Por ejemplo, una dirección IP asociada con un registro A, o un nombre de dominio asociado con un registro NS. Recuerde que los nombres de dominio son relativos al origen a menos que terminen con punto.

`zone-file` [Tipo de datos]
Tipo de datos que representa el contenido de un archivo de zona. Este tipo tiene los siguientes parámetros:

`entries` (predeterminadas: `'()`)
: La lista de entradas. El registro SOA se genera automáticamente, por lo que no necesita ponerlo en la lista de entradas. Esta lista probablemente debería contener una entrada apuntando a su servidor DNS de

[11] Name Server en inglés.
[12] Mail eXchange en inglés

autoridad. En vez de usar una lista de entradas directamente, puede usar `define-zone-entries` para definir un objeto que contenga la lista de entradas más fácilmente, que posteriormente puede proporcionar en el campo `entries` del archivo `zone-file`.

`origin` (predeterminado: `""`)
: El nombre de su zona. Este parámetro no puede estar vacío.

`ns` (predeterminado: `"ns"`)
: El dominio de su servidor DNS primario de autoridad. El nombre es relativo al origen, a menos que termine en punto. Es obligatorio que este servidor DNS primario corresponda con un registro NS en la zona y que esté asociado a una dirección IP en la lista de entradas.

`mail` (predeterminado: `"hostmaster"`)
: Dirección de correo a través de la cual la gente puede contactar con usted, como propietaria de la zona. Se traduce a `<mail>@<origin>`.

`serial` (predeterminado: `1`)
: Número serie de la zona. Como se usa para tener constancia de los cambios tanto en servidores esclavos como en resolvedores, es obligatorio que *nunca* decremente. Incremente su valor siempre que haga cambios en su zona.

`refresh` (predeterminado: `(* 2 24 3600)`)
: La frecuencia con la que los servidores esclavos realizarán una transferencia de zona. Este valor es un número de segundos. Puede calcularse con multiplicaciones o con `(string->duration)`.

`retry` (predeterminado: `(* 15 60)`)
: El periodo tras el cual un servidor esclavo reintentará el contacto con su maestro cuando falle al intentarlo la primera vez.

`expiry` (predeterminado: `(* 14 24 3600)`)
: Tiempo de vida (TTL) predeterminado de los registros. Los registros existentes se consideran correctos durante al menos este periodo de tiempo. Tras este periodo, los resolvedores invalidarán su caché y comprobarán de nuevo que todavía exista.

`nx` (predeterminado: `3600`)
: Default TTL of inexistent records. This delay is usually short because you want your new domains to reach everyone quickly.

`knot-remote-configuration` [Tipo de datos]

Tipo de datos que representa una configuración remota. Este tipo tiene los siguientes parámetros:

`id` (predeterminado: `""`)
: An identifier for other configuration fields to refer to this remote. IDs must be unique and must not be empty.

`address` (predeterminada: `'()`)
: Una lista ordenada de direcciones IP de destino. Las direcciones se prueban en secuencia. Opcionalmente se puede proporcionar el puerto con el

separador @. Por ejemplo: `(list "1.2.3.4" "2.3.4.5@53")`. El puerto predeterminado es el 53.

`via` (predeterminada: `'()`)
An ordered list of source IP addresses. An empty list will have Knot choose an appropriate source IP. An optional port can be given with the @ separator. The default is to choose at random.

`key` (predeterminada: `#f`)
Referencia a una clave, esto es una cadena que contiene el identificador de una clave definida en el campo `knot-key-configuration`.

`knot-keystore-configuration` [Tipo de datos]
Tipo de datos que representa un almacén de claves para alojar claves de dnssec. Este tipo tiene los siguientes parámetros:

`id` (predeterminado: `""`)
El identificador del almacén de claves. No debe estar vacío.

`backend` (predeterminado: `'pem`)
El motor en el que se almacenan las claves. Puede ser `'pem` o `'pkcs11`.

`config` (predeterminada: `"/var/lib/knot/keys/keys"`)
La cadena de configuración del motor. Un ejemplo para PKCS#11 es: `"pkcs11:token=knot;pin-value=1234 /gnu/store/.../lib/pkcs11/libsofthsm2.so"`. La cadena representa una ruta en el sistema de archivos para el motor pem.

`knot-policy-configuration` [Tipo de datos]
Tipo de datos que representa una política de dnssec. El DNS Knot es capaz de firmar automáticamente sus zonas. Puede generar y gestionar sus claves de manera automática o usar las claves que usted genere.

Dnssec se implementa habitualmente usando dos claves: una clave para firma de claves (KSK) que se usa para firmar la segunda, y una clave para firma de zona (ZSK) que se usa para firmar la zona. Para establecer la confianza, la KSK necesita estar presente en la zona padre (habitualmente un dominio de nivel superior). Si su entidad de registro permite dnssec, debe mandarle el hash de su KSK de manera que puedan añadir un registro DS en su zona. No es automático y debe realizarse cada vez que cambie su KSK.

La política también define el tiempo de vida de las claves. Habitualmente, la ZSK puede cambiarse fácilmente y usa funciones criptográficas más débiles (usa parámetros de menor magnitud) para firmar los registros rápidamente, ya que cambian habitualmente. No obstante, la KSK requiere interacción manual con la entidad de registro, por lo que se cambia menos habitualmente y usa parámetros más fuertes debido a que únicamente firma un registro.

Este tipo tiene los siguientes parámetros:

`id` (predeterminado: `""`)
El identificador de la política. No debe estar vacío.

`keystore` (predeterminado: `"default"`)
: Referencia a un almacén de claves, es decir una cadena que contiene el identificador de un almacén de claves definido en un campo de `knot-keystore-configuration`. El identificador predeterminado `"default"` implica el uso del almacén de claves predeterminado (una base de datos kasp que se configura para este servicio).

`manual?` (predeterminado: `#f`)
: Si la gestión de claves es manual o automática.

`single-type-signing?` (predeterminado: `#f`)
: Cuando sea `#t`, usa el esquema de firma de tipo único (Single-Type Signing Scheme).

`algorithm` (predeterminado: `"ecdsap256sha256"`)
: Algoritmo para las claves de firma y las firmas emitidas.

`ksk-size` (predeterminado: `256`)
: The length of the KSK. Note that this value is correct for the default algorithm, but would be unsecure for other algorithms.

`zsk-size` (predeterminado: `256`)
: The length of the ZSK. Note that this value is correct for the default algorithm, but would be unsecure for other algorithms.

`dnskey-ttl` (predeterminado: `'default`)
: El valor del tiempo de vida (TTL) de los registros DNSKEY añadidos al "apex" de la zona. El valor especial `'default` significa el mismo valor que el TTL del SOA de la zona.

`zsk-lifetime` (predeterminado: `(* 30 24 3600)`)
: El periodo entre la publicación de la ZSK y el inicio del siguiente ciclo de renovación.

`propagation-delay` (predeterminado: `(* 24 3600)`)
: Retraso adicional añadido por cada paso del ciclo de renovación de clave. Este valor debe ser suficientemente alto para cubrir la propagación de datos del servidor maestro a todos los esclavos.

`rrsig-lifetime` (predeterminado: `(* 14 24 3600)`)
: Periodo de validez para las nuevas firmas emitidas.

`rrsig-refresh` (predeterminado: `(* 7 24 3600)`)
: Periodo de antelación con el que se realiza el refresco de la firma antes de una expiración de la misma.

`nsec3?` (predeterminado: `#f`)
: Si es `#t`, se usa NSEC3 en vez de NSEC.

`nsec3-iterations` (predeterminado: `5`)
: Número de ejecuciones adicionales de la operación de hash.

`nsec3-salt-length` (predeterminado: `8`)
: La longitud del campo "salt" en octetos, que se añade al nombre de la propietaria original antes de ejecutar la operación de hash.

`nsec3-salt-lifetime` (predeterminado: `(* 30 24 3600)`)
: El periodo de validez de los campos "salt" que se generen.

`knot-zone-configuration` [Tipo de datos]
: Tipo de datos que representa una zona ofrecida por Knot. Este tipo tiene los siguientes parámetros:

  `domain` (predeterminado: `""`)
  : El dominio ofrecido con esta configuración. No debe estar vacío.

  `file` (predeterminado: `""`)
  : El archivo donde se almacena esta zona. Este parámetro se ignora para zonas maestras. Vacío significa la ruta predeterminada que depende del nombre del dominio.

  `zone` (predeterminado: `(zone-file)`)
  : El contenido del archivo de zona. Este parámetro se ignora para zonas esclavas. Debe contener un registro de archivo de zona.

  `master` (predeterminado: `'()`)
  : Lista de maestros remotos. Cuando está vacía, esta zona es maestra. Cuando tiene contenido, esta zona es esclava. Es una lista de identificadores remotos.

  `ddns-master` (predeterminado: `#f`)
  : Maestro principal. Cuando está vacío, apunta de manera predeterminada al primer maestro en la lista de maestros.

  `notify` (predeterminado: `'()`)
  : Una lista de identificadores remotos de esclavos.

  `acl` (predeterminado: `'()`)
  : Lista de identificadores acl.

  `semantic-checks?` (predeterminado: `#f`)
  : Cuando es verdadero, añade más comprobaciones semánticas a la zona.

  `zonefile-sync` (predeterminado: `0`)
  : El retraso entre una modificación en memoria y en disco. 0 significa sincronización inmediata.

  `zonefile-load` (predeterminado: `#f`)
  : La forma en la que los contenidos del archivo de zona se aplican durante la carga de la zona. Los valores posibles son:

    - `#f` para obtener el valor predeterminado de Knot,
    - `'none` para no usar el archivo de zona en absoluto,
    - `'difference` para calcular la diferencia entre los contenidos disponibles actualmente y los contenidos de la zona y los aplica a los contenidos actuales de la zona actual,
    - `'difference-no-serial` es igual que `'difference`, pero ignora el código serie SOA en el archivo de zona, mientras que el servidor se hace cargo de él de manera automática.

- `'whole` para cargar los contenidos de la zona del archivo de zona.

`journal-content` (predeterminado: `'()`)
: La forma en la que se usa el diario para almacenar la zona y sus cambios. Los posibles valores son `'none` para no usarlo en absoluto, `'changes` para almacenar los cambios y `'all` para almacenar los contenidos. `#f` proporciona un valor a esta opción, por lo que se usa el valor predeterminado de Knot.

`max-journal-usage` (predeterminado: `#f`)
: Tamaño máximo del diario en disco. `#f` no proporciona un valor a esta opción, por lo que se usa el valor predeterminado de Knot.

`max-journal-depth` (predeterminado: `#f`)
: Tamaño máximo de la historia. `#f` proporciona un valor a esta opción, por lo que se usa el valor predeterminado de Knot.

`max-zone-size` (predeterminado: `#f`)
: Tamaño máximo del archivo de zona. Este límite se usa para transferencias entrantes y actualizaciones. `#f` no proporciona un valor a esta opción, por lo que se usa el valor predeterminado de Knot.

`dnssec-policy` (predeterminado: `#f`)
: Una referencia a un registro de `knot-policy-configuration`, o el nombre especial `"default"`. Si el valor es `#f`, no se realiza firma dnssec en esta zona.

`serial-policy` (predeterminado: `'increment`)
: Una política entre `'increment` y `'unixtime`.

`knot-configuration` [Tipo de datos]
Tipo de datos que representa la configuración Knot. Este tipo tiene los siguientes parámetros:

`knot` (predeterminado: `knot`)
: El paquete Knot.

`run-directory` (predeterminado: `"/var/run/knot"`)
: El directorio de ejecución. Este directorio se usará para los archivos de PID y de sockets.

`includes` (predeterminada: `'()`)
: Una lista de cadenas u objetos “tipo-archivo” que denota otros archivos que deben incluirse al inicio del archivo de configuración.

  Puede usarse para gestionar secretos en un canal separado. Por ejemplo, las claves secretas pueden almacenarse en un archivo fuera de banda no gestionado por Guix, y por tanto no visible en `/gnu/store`—por ejemplo, puede almacenar su configuración de clave secreta en `/etc/knot/secrets.conf` e incluir este archivo en la lista `includes`.

  Se puede generar una clave secreta tsig (para nsupdate y transferencias de zona) con la orden keymgr del paquete knot. Tenga en cuenta que

el paquete no se instala automáticamente con el servicio. El ejemplo siguiente muestra como generar una clave tsig nueva:

```
keymgr -t misecreto > /etc/knot/secrets.conf
chmod 600 /etc/knot/secrets.conf
```

Tenga también en cuenta que la clave generada se llamará *misecreto*, de modo que ese nombre es el que debe usarse en el campo *key* del registro `knot-acl-configuration` y en otros lugares que hagan referencia a esa clave.

También puede usarse para añadir configuración no implementada por esta interfaz.

`listen-v4` (predeterminada: `"0.0.0.0"`)
La dirección IP en la que escuchar.

`listen-v6` (predeterminada: `"::"`)
La dirección IP en la que escuchar.

`listen-port` (predeterminado: `53`)
El puerto en el que escuchar.

`keys` (predeterminada: `'()`)
La lista de configuraciones knot-key-configuration usadas por esta configuración.

`acls` (predeterminado: `'()`)
La lista de configuraciones knot-acl-configuration usadas por esta configuración.

`remotes` (predeterminada: `'()`)
La lista de configuraciones knot-remote-configuration usadas por esta configuración.

`zones` (predeterminada: `'()`)
La lista de configuraciones knot-zone-configuration usadas por esta configuración.

## Servicio de resolución de Knot

`knot-resolver-service-type` [Variable]

This is the type of the knot resolver service, whose value should be a `knot-resolver-configuration` object as in this example:

```
(service knot-resolver-service-type
         (knot-resolver-configuration
           (kresd-config-file (plain-file "kresd.conf" "
net.listen('192.168.0.1', 5353)
user('knot-resolver', 'knot-resolver')
modules = { 'hints > iterate', 'stats', 'predict' }
cache.size = 100 * MB
"))))
```

For more information, refer its manual (`https://knot-resolver.readthedocs.io/en/stable/config-overview.html`).

`knot-resolver-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de knot-resolver.

`package` (predeterminado: *knot-resolver*)
El objeto paquete de la resolución de DNS de knot.

`kresd-config-file` (predeterminado: %kresd.conf)
Objeto "tipo-archivo" con el archivo de configuración de kresd usado, de manera predeterminada escucha en `127.0.0.1` y `::1`.

`garbage-collection-interval` (predeterminado: 1000)
Número de milisegundos tras los que `kres-cache-gc` realiza una limpieza periódica de la caché.

## Servicio Dnsmasq

`dnsmasq-service-type` [Variable]
This is the type of the dnsmasq service, whose value should be a `dnsmasq-configuration` object as in this example:

```
(service dnsmasq-service-type
         (dnsmasq-configuration
           (no-resolv? #t)
           (servers '("192.168.1.1"))))
```

`dnsmasq-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de dnsmasq.

`package` (predeterminado: *dnsmasq*)
El objeto paquete del servidor dnsmasq.

`no-hosts?` (predeterminado: `#f`)
Cuando es verdadero, no lee los nombres de máquina en /etc/hosts.

`port` (predeterminado: `53`)
El puerto sobre el que se escucha. Proporcionar el valor cero deshabilita las respuestas DNS completamente, dejando las funciones DHCP y/o TFTP únicamente.

`local-service?` (predeterminado: `#t`)
Acepta peticiones DNS únicamente de máquinas cuya dirección esté en una subred local, es decir, subred para la que existe una interfaz en el servidor.

`listen-addresses` (predeterminadas: `'()`)
Escucha en las direcciones IP proporcionadas.

`resolv-file` (predeterminado: `"/etc/resolv.conf"`)
Archivo en el que se obtienen las direcciones IP de los servidores de nombres desde los que se obtienen datos.

`no-resolv?` (predeterminado: `#f`)
Cuando tiene valor verdadero, no se lee *resolv-file*.

`forward-private-reverse-lookup?` (default: `#t`)
When false, all reverse lookups for private IP ranges are answered with "no such domain" rather than being forwarded upstream.

`query-servers-in-order?` (default: `#f`)
When true, dnsmasq queries the servers in the same order as they appear in *servers*.

`servers` (predeterminados: `'()`)
Especifica directamente la dirección IP de los servidores proveedores.

`servers-file` (default: `#f`)
Specify file containing upstream servers. This file is re-read when dnsmasq receives SIGHUP. Could be either a string or a file-like object.

`addresses` (predeterminado: `'()`)
Cada entrada especifica una dirección IP devuelta por cualquiera de las máquinas en los dominios proporcionados. Las búsquedas dentro de los dominios nunca se redirigen y siempre se devuelve la dirección IP especificada.

Es útil para redirigir máquinas localmente, como en este ejemplo:

```
(service dnsmasq-service-type
         (dnsmasq-configuration
           (addresses
             '(; Redirecciona a un servidor web local.
               "/example.org/127.0.0.1"
               ; Redirecciona un dominio a una IP específica.
               "/subdomain.example.org/192.168.1.42"))))
```

Tenga en cuenta que las reglas en el archivo `/etc/hosts` tienen precedencia sobre esto.

`cache-size` (predeterminado: `150`)
Establece el tamaño de la caché de dnsmasq. Proporcionar el valor cero desactiva el almacenamiento en caché.

`negative-cache?` (predeterminado: `#t`)
Cuando es falso, desactiva la caché negativa.

`cpe-id` (default: `#f`)
If set, add a CPE (Customer-Premises Equipment) identifier to DNS queries which are forwarded upstream.

`tftp-enable?` (predeterminado: `#f`)
Determina si se activa el servidor TFTP incluido.

`tftp-no-fail?` (predeterminado: `#f`)
Si es verdadero, dnsmasq no falla si el servidor TFTP no se ha podido arrancar.

`tftp-single-port?` (predeterminado: `#f`)
Determina si se un único puerto para TFTP.

`tftp-secure?` (predeterminado: `#f`)
: Si es verdadero, únicamente los archivos propiedad de la cuenta que ejecuta el proceso dnsmasq están accesibles.

  Si se ejecuta dnsmasq como "root", se aplican diferentes reglas: `tftp-secure?` no tiene efecto, únicamente los archivos que tengan permiso de lectura global el mundo serán accesibles.

`tftp-max` (predeterminado: `#f`)
: Cuando se proporciona un valor indica el número máximo de conexiones simultáneas permitidas.

`tftp-mtu` (predeterminado: `#f`)
: Si se proporciona un valor, establece el tamaño de la unidad de transmisión de mensajes (MTU) para los paquetes TFTP a dicho valor.

`tftp-no-blocksize?` (predeterminado: `#f`)
: Si es verdadero, impide al servidor TFTP la negociación del tamaño del bloque con un cliente.

`tftp-lowercase?` (predeterminado: `#f`)
: Determina si se convierten a minúsculas todos los nombres de archivo en peticiones TFTP.

`tftp-port-range` (predeterminado: `#f`)
: Si se proporciona un valor, este establece el rango (`"<inicio>,<fin>"`) de puertos dinámicos (uno por cliente) usados.

`tftp-root` (predeterminado: `/var/empty,lo`)
: Look for files to transfer using TFTP relative to the given directory. When this is set, TFTP paths which include '`..`' are rejected, to stop clients getting outside the specified root. Absolute paths (starting with '`/`') are allowed, but they must be within the TFTP-root. If the optional interface argument is given, the directory is only used for TFTP requests via that interface.

`tftp-unique-root` (predeterminado: `#f`)
: Si se proporciona un valor, añade la dirección IP o hardware del cliente TFTP como un componente de la ruta tras el final de `tftp-root`. Es válido únicamente si se proporciona una raíz para TFTP y el directorio existe. El valor predeterminado añade la dirección IP (en el formato estándar de cuatro cifras separadas por puntos).

  For instance, if `--tftp-root` is '`/tftp`' and client '`1.2.3.4`' requests file `myfile` then the effective path will be `/tftp/1.2.3.4/myfile` if `/tftp/1.2.3.4` exists or `/tftp/myfile` otherwise. When '`=mac`' is specified it will append the MAC address instead, using lowercase zero padded digits separated by dashes, e.g.: '`01-02-03-04-aa-bb`'. Note that resolving MAC addresses is only possible if the client is in the local network or obtained a DHCP lease from dnsmasq.

### 11.10.23 VNC Services

The `(gnu services vnc)` module provides services related to *Virtual Network Computing* (VNC), which makes it possible to locally use graphical Xorg applications running on a remote machine. Combined with a graphical manager that supports the *X Display Manager Control Protocol*, such as GDM (see [gdm], page 340) or LightDM (see [lightdm], page 345), it is possible to remote an entire desktop for a multi-user environment.

#### Xvnc

Xvnc is a VNC server that spawns its own X window server; which means it can run on headless servers. The Xvnc implementations provided by the `tigervnc-server` and `turbovnc` aim to be fast and efficient.

`xvnc-service-type` [Variable]

The `xvnc-service-type` service can be configured via the `xvnc-configuration` record, documented below. A second virtual display could be made available on a remote machine via the following configuration:

```
(service xvnc-service-type
         (xvnc-configuration (display-number 10)))
```

As a demonstration, the `xclock` command could then be started on the remote machine on display number 10, and it could be displayed locally via the `vncviewer` command:

```
# Start xclock on the remote machine.
ssh -L5910:localhost:5910 your-host -- guix shell xclock \
    -- env DISPLAY=:10 xclock
# Access it via VNC.
guix shell tigervnc-client -- vncviewer localhost:5910
```

The following configuration combines XDMCP and Inetd to allow multiple users to concurrently use the remote system and login graphically via the GDM display manager:

```
(operating-system
  [...]
  (services (cons*
             [...]
             (service xvnc-service-type (xvnc-configuration
                                         (display-number 5)
                                         (localhost? #f)
                                         (xdmcp? #t)
                                         (inetd? #t)))
             (modify-services %desktop-services
               (gdm-service-type config => (gdm-configuration
                                            (inherit config)
                                            (auto-suspend? #f)
                                            (xdmcp? #t)))))))
```

A remote user could then connect to it by using the `vncviewer` command or a compatible VNC client and start a desktop session of their choosing:

```
vncviewer remote-host:5905
```

**Aviso:** Unless your machine is in a controlled environment, for security reasons, the `localhost?` configuration of the `xvnc-configuration` record should be left to its default `#t` value and exposed via a secure means such as an SSH port forward. The XDMCP port, UDP 177 should also be blocked from the outside by a firewall, as it is not a secure protocol and can expose login credentials in clear.

`xvnc-configuration` [Data Type]

Available `xvnc-configuration` fields are:

`xvnc` (default: `tigervnc-server`) (type: file-like)
The package that provides the Xvnc binary.

`display-number` (default: `0`) (type: number)
The display number used by Xvnc. You should set this to a number not already used a Xorg server.

`geometry` (default: `"1024x768"`) (type: string)
The size of the desktop to be created.

`depth` (default: `24`) (type: color-depth)
The pixel depth in bits of the desktop to be created. Accepted values are 16, 24 or 32.

`port` (type: maybe-port)
The port on which to listen for connections from viewers. When left unspecified, it defaults to 5900 plus the display number.

`ipv4?` (default: `#t`) (type: boolean)
Use IPv4 for incoming and outgoing connections.

`ipv6?` (default: `#t`) (type: boolean)
Use IPv6 for incoming and outgoing connections.

`password-file` (type: maybe-string)
The password file to use, if any. Refer to vncpasswd(1) to learn how to generate such a file.

`xdmcp?` (default: `#f`) (type: boolean)
Query the XDMCP server for a session. This enables users to log in a desktop session from the login manager screen. For a multiple users scenario, you'll want to enable the `inetd?` option as well, so that each connection to the VNC server is handled separately rather than shared.

`inetd?` (default: `#f`) (type: boolean)
Use an Inetd-style service, which runs the Xvnc server on demand.

`frame-rate` (default: `60`) (type: number)
The maximum number of updates per second sent to each client.

`security-types` (default: `'("None")`) (type: security-types)
The allowed security schemes to use for incoming connections. The default is `"None"`, which is safe given that Xvnc is configured to authenticate the user via the display manager, and only for local connections.

Accepted values are any of the following: ("None" "VncAuth" "Plain" "TLSNone" "TLSVnc" "TLSPlain" "X509None" "X509Vnc")

`localhost?` (default: `#t`) (type: boolean)
Only allow connections from the same machine. It is set to #true by default for security, which means SSH or another secure means should be used to expose the remote port.

`log-level` (default: `30`) (type: log-level)
The log level, a number between 0 and 100, 100 meaning most verbose output. The log messages are output to syslog.

`extra-options` (default: `'()`) (type: strings)
This can be used to provide extra Xvnc options not exposed via this <xvnc-configuration> record.

### 11.10.24 Servicios VPN

The `(gnu services vpn)` module provides services related to *virtual private networks* (VPNs).

#### Bitmask

`bitmask-service-type` [Variable]
A service type for the Bitmask (`https://bitmask.net`) VPN client. It makes the client available in the system and loads its polkit policy. Please note that the client expects an active polkit-agent, which is either run by your desktop-environment or should be run manually.

#### OpenVPN

It provides a *client* service for your machine to connect to a VPN, and a *server* service for your machine to host a VPN. Both `openvpn-client-service-type` and `openvpn-server-service-type` can be run simultaneously.

`openvpn-client-service-type` [Variable]
Type of the service that runs `openvpn`, a VPN daemon, as a client.

The value for this service is a `<openvpn-client-configuration>` object.

`openvpn-server-service-type` [Variable]
Type of the service that runs `openvpn`, a VPN daemon, as a server.

The value for this service is a `<openvpn-server-configuration>` object.

`openvpn-client-configuration` [Data Type]
Los campos disponibles de `openvpn-client-configuration` son:

`openvpn` (default: `openvpn`) (type: file-like)
El paquete OpenVPN.

`pid-file` (default: `"/var/run/openvpn/openvpn.pid"`) (type: string)
El archivo de pid de OpenVPN.

`proto` (default: `udp`) (type: proto)
: El protocolo (UDP o TCP) usado para la apertura del canal entre clientes y servidores.

`dev` (default: `tun`) (type: dev)
: El tipo de dispositivo usado para representar la conexión VPN.

`ca` (default: `"/etc/openvpn/ca.crt"`) (type: maybe-string)
: La autoridad de certificación contra la que se comprueban las conexiones.

`cert` (default: `"/etc/openvpn/client.crt"`) (type: maybe-string)
: El certificado de la máquina en la que se ejecuta el daemon. Debe estar firmado por la autoridad proporcionada en `ca`.

`key` (default: `"/etc/openvpn/client.key"`) (type: maybe-string)
: La clave de la máquina en la que se ejecuta el daemon. Debe ser la clave cuyo certificado es `cert`.

`comp-lzo?` (default: `#t`) (type: boolean)
: Determina si se usa el algoritmo de compresión lzo.

`persist-key?` (default: `#t`) (type: boolean)
: No vuelve a leer los archivos de claves tras la señal SIGUSR1 o –ping-restart.

`persist-tun?` (default: `#t`) (type: boolean)
: No cierra y reabre el dispositivo TUN/TAP o ejecuta los guiones de parada e inicio tras el reinicio provocado por SIGUSR1 o –ping-restart.

`fast-io?` (default: `#f`) (type: boolean)
: (Experimental) Optimiza las escrituras de E/S de TUN/TAP/UDP evitando llamar poll/epoll/select antes de la operación de escritura (`write`).

`verbosity` (default: `3`) (type: number)
: Nivel de detalle en los mensajes.

`tls-auth` (default: `#f`) (type: tls-auth-client)
: Añade una capa adicional de verificación HMAC sobre el canal de control TLS para protección contra ataques de denegación de servicio (DoS).

`auth-user-pass` (type: maybe-string)
: Activa la identificación con el servidor mediante el uso de usuaria/contraseña. La opción es un archivo que contiene en dos líneas el nombre de usuaria y la contraseña. No use un objeto tipo-archivo, ya que se añadiría al almacén y sería legible para cualquier usuaria.

`verify-key-usage?` (default: `#t`) (type: key-usage)
: Si se comprueba que el certificado del servidor tenga la extensión de uso de servidor.

`bind?` (default: `#f`) (type: bind)
: Asociación a un número específico de puerto local.

`resolv-retry?` (default: `#t`) (type: resolv-retry)
: Reintentos de resolución de la dirección del servidor.

`remote` (default: `'()`) (type: openvpn-remote-list)
Una lista de servidores remotos a los que conectarse.

`openvpn-remote-configuration` [Data Type]
Los campos disponibles de `openvpn-remote-configuration` son:

`name` (default: `"my-server"`) (type: string)
Nombre del servidor.

`port` (default: `1194`) (type: number)
Puerto en el que escucha el servidor.

`openvpn-server-configuration` [Data Type]
Los campos disponibles de `openvpn-server-configuration` son:

`openvpn` (default: `openvpn`) (type: file-like)
El paquete OpenVPN.

`pid-file` (default: `"/var/run/openvpn/openvpn.pid"`) (type: string)
El archivo de pid de OpenVPN.

`proto` (default: `udp`) (type: proto)
El protocolo (UDP o TCP) usado para la apertura del canal entre clientes y servidores.

`dev` (default: `tun`) (type: dev)
El tipo de dispositivo usado para representar la conexión VPN.

`ca` (default: `"/etc/openvpn/ca.crt"`) (type: maybe-string)
La autoridad de certificación contra la que se comprueban las conexiones.

`cert` (default: `"/etc/openvpn/client.crt"`) (type: maybe-string)
El certificado de la máquina en la que se ejecuta el daemon. Debe estar firmado por la autoridad proporcionada en `ca`.

`key` (default: `"/etc/openvpn/client.key"`) (type: maybe-string)
La clave de la máquina en la que se ejecuta el daemon. Debe ser la clave cuyo certificado es `cert`.

`comp-lzo?` (default: `#t`) (type: boolean)
Determina si se usa el algoritmo de compresión lzo.

`persist-key?` (default: `#t`) (type: boolean)
No vuelve a leer los archivos de claves tras la señal SIGUSR1 o –ping-restart.

`persist-tun?` (default: `#t`) (type: boolean)
No cierra y reabre el dispositivo TUN/TAP o ejecuta los guiones de parada e inicio tras el reinicio provocado por SIGUSR1 o –ping-restart.

`fast-io?` (default: `#f`) (type: boolean)
(Experimental) Optimiza las escrituras de E/S de TUN/TAP/UDP evitando llamar poll/epoll/select antes de la operación de escritura (`write`).

`verbosity` (default: `3`) (type: number)
Nivel de detalle en los mensajes.

`tls-auth` (default: `#f`) (type: tls-auth-server)
: Añade una capa adicional de verificación HMAC sobre el canal de control TLS para protección contra ataques de denegación de servicio (DoS).

`port` (default: `1194`) (type: number)
: Especifica el número de puerto en el que escucha el servidor.

`server` (default: `"10.8.0.0 255.255.255.0"`) (type: ip-mask)
: Una IP y una máscara que especifiquen la subred dentro de la red virtual.

`server-ipv6` (default: `#f`) (type: cidr6)
: La especificación de una subred IPv6 dentro de la red virtual en notación CIDR.

`dh` (default: `"/etc/openvpn/dh2048.pem"`) (type: string)
: El archivo de parámetros Diffie-Hellman.

`ifconfig-pool-persist` (default: `"/etc/openvpn/ipp.txt"`) (type: string)
: El archivo que registra IP de clientes.

`redirect-gateway?` (default: `#f`) (type: gateway)
: Cuando sea verdadero, el servidor actuará como una pasarela para sus clientes.

`client-to-client?` (default: `#f`) (type: boolean)
: Cuando es verdadero, se permite la comunicación entre clientes dentro de la VPN.

`keepalive` (default: `(10 120)`) (type: keepalive)
: Hace que se envíen mensajes tipo-ping en ambas direcciones a través del enlace de modo que cada extremo conozca si el otro extremo no está disponible. `keepalive` necesita un par. El primer elemento es el periodo de envío de ping, y el segundo elemento es el plazo máximo antes de considerar que el otro extremo no está disponible.

`max-clients` (default: `100`) (type: number)
: Número máximo de clientes.

`status` (default: `"/var/run/openvpn/status"`) (type: string)
: El archivo de estado. Este archivo muestra un pequeño informe sobre la conexión actual. Su contenido se borra y se reescribe cada minuto.

`client-config-dir` (default: `'()`) (type: openvpn-ccd-list)
: Lista de configuración para algunos clientes.

### strongSwan

Currently, the strongSwan service only provides legacy-style configuration with `ipsec.conf` and `ipsec.secrets` files.

`strongswan-service-type` [Variable]
: A service type for configuring strongSwan for IPsec VPN (Virtual Private Networking). Its value must be a `strongswan-configuration` record as in this example:

```
(service strongswan-service-type
```

```
(strongswan-configuration
 (ipsec-conf "/etc/ipsec.conf")
 (ipsec-secrets "/etc/ipsec.secrets")))
```

`strongswan-configuration` [Data Type]

Data type representing the configuration of the StrongSwan service.

`strongswan`
: The strongSwan package to use for this service.

`ipsec-conf` (default: `#f`)
: The file name of your `ipsec.conf`. If not `#f`, then this and `ipsec-secrets` must both be strings.

`ipsec-secrets` (default `#f`)
: The file name of your `ipsec.secrets`. If not `#f`, then this and `ipsec-conf` must both be strings.

## Wireguard

`wireguard-service-type` [Variable]

A service type for a Wireguard tunnel interface. Its value must be a `wireguard-configuration` record as in this example:

```
(service wireguard-service-type
         (wireguard-configuration
          (peers
           (list
            (wireguard-peer
             (name "my-peer")
             (endpoint "my.wireguard.com:51820")
             (public-key "hzpKg9X1yqu1axN6iJp0mWf6BZGo8m1wteKwtTmDGF4=")
             (allowed-ips '("10.0.0.2/32")))))))
```

`wireguard-configuration` [Data Type]

Data type representing the configuration of the Wireguard service.

`wireguard`
: The wireguard package to use for this service.

`interface` (default: `"wg0"`)
: The interface name for the VPN.

`addresses` (default: `'("10.0.0.1/32")`)
: The IP addresses to be assigned to the above interface.

`port` (default: `51820`)
: The port on which to listen for incoming connections.

`dns` (default: `'()`)
: The DNS server(s) to announce to VPN clients via DHCP.

`monitor-ips?` (default: `#f`)
: Whether to monitor the resolved Internet addresses (IPs) of the endpoints of the configured peers, resetting the peer endpoints using an IP address

that no longer correspond to their freshly resolved host name. Set this to `#t` if one or more endpoints use host names provided by a dynamic DNS service to keep the sessions alive.

`monitor-ips-interval` (default: `'(next-minute (range 0 60 5))`)
The time interval at which the IP monitoring job should run, provided as an mcron time specification (see Section "Guile Syntax" in `mcron`).

`private-key` (default: `"/etc/wireguard/private.key"`)
The private key file for the interface. It is automatically generated if the file does not exist.

`peers` (default: `'()`)
The authorized peers on this interface. This is a list of *wireguard-peer* records.

`pre-up` (default: `'()`)
The script commands to be run before setting up the interface.

`post-up` (default: `'()`)
The script commands to be run after setting up the interface.

`pre-down` (default: `'()`)
The script commands to be run before tearing down the interface.

`post-down` (default: `'()`)
The script commands to be run after tearing down the interface.

`table` (default: `"auto"`)
The routing table to which routes are added, as a string. There are two special values: `"off"` that disables the creation of routes altogether, and `"auto"` (the default) that adds routes to the default table and enables special handling of default routes.

`wireguard-peer` [Data Type]
Data type representing a Wireguard peer attached to a given interface.

`name` The peer name.

`endpoint` (default: `#f`)
The optional endpoint for the peer, such as `"demo.wireguard.com:51820"`.

`public-key`
The peer public-key represented as a base64 string.

`preshared-key` (default: `#f`)
An optional pre-shared key file for this peer. The given file will not be autogenerated.

`allowed-ips`
A list of IP addresses from which incoming traffic for this peer is allowed and to which incoming traffic for this peer is directed.

`keep-alive` (default: `#f`)
An optional time interval in seconds. A packet will be sent to the server endpoint once per time interval. This helps receiving incoming connections from this peer when you are behind a NAT or a firewall.

### 11.10.25 Sistema de archivos en red

El módulo `(gnu services nfs)` proporciona los siguientes servicios, que se usan habitualmente en relación con el montado o la exportación de árboles de directorios como *sistemas de archivos en red* (NFS).

Mientras que es posible usar los componentes individuales de forma conjunta para proporcionar un servicio del sistema de archivos en red NFS, recomendamos la configuración de un servidor NFS mediante `nfs-service-type`.

#### Servicio NFS

El servicio NFS se hace cargo de configurar todos los servicios de componentes de NFS, la configuración del núcleo de sistemas de archivos e instala los archivos de configuración en las rutas que NFS espera.

`nfs-service-type` [Variable]
Este es el tipo de datos para un servidor NFS completo.

`nfs-configuration` [Tipo de datos]
Este tipo de datos representa la configuración del servicio NFS y todos sus subsistemas.

Tiene los siguientes parámetros:

`nfs-utils` (predeterminado: `nfs-utils`)
El paquete nfs-utils usado.

`nfs-versions` (predeterminadas: `'("4.2" "4.1" "4.0")`)
Si se proporciona como valor una lista de cadenas, el daemon `rpc.nfsd` se limitará a la implementación de las versiones del protocolo NFS proporcionadas.

`exports` (predeterminada: `'()`)
Una lista de directorios que el servidor NFS debe exportar. Cada entrada es una lista que consiste en dos elementos: un nombre de directorio y una cadena que contiene todas las opciones. Este es un ejemplo en el que el directorio `/exportado` se proporciona a todos los clientes NFS en modo de solo-lectura:

```
(nfs-configuration
 (exports
  '(("/exportado"
     "*(ro,insecure,no_subtree_check,crossmnt,fsid=0)"))))
```

`rpcmountd-port` (predeterminado: `#f`)
El puerto de red que el daemon `rpc.mountd` debe usar.

`rpcstatd-port` (predeterminado: `#f`)
El puerto de red que el daemon `rpc.statd` debe usar.

`rpcbind` (predeterminado: `rpcbind`)
El paquete rpcbind usado.

`idmap-domain` (predeterminado: `"localdomain"`)
El nombre de dominio local de NFSv4.

`nfsd-port` (predeterminado: `2049`)
: El puerto de red que el daemon `nfsd` debe usar.

`nfsd-threads` (predeterminado: `8`)
: Número de hilos usados en el daemon `nfsd`.

`nfsd-tcp?` (predeterminado: `#t`)
: Determina si el daemon `nfsd` debe escuchar en un puerto TCP.

`nfsd-udp?` (predeterminado: `#f`)
: Determina si el daemon `nfsd` debe escuchar en un puerto UDP.

`pipefs-directory` (predeterminado: `"/var/lib/nfs/rpc_pipefs"`)
: El directorio donde el sistema de archivos pipefs debe montarse.

`debug` (predeterminada: `'()"`)
: Una lista de subsistemas para los cuales debe activarse la salida de depuración. Es una lista de símbolos. Cualquiera de estos símbolos son válidos: `nfsd`, `nfs`, `rpc`, `idmap`, `statd` o `mountd`.

Si no necesita un servicio NFS completo o prefiere construirlo por su cuenta puede usar los componentes individuales que se documentan a continuación.

## Servicio RPC Bind

El servicio RPC Bind proporciona una forma de asociar números de programa con direcciones universales. Muchos servicios relacionados con NFS usan esta característica. De ahí que se inicie automáticamente cuando un servicio dependiente se inicia.

`rpcbind-service-type` [Variable]
: Un tipo de servicio para el daemon de asignación de puertos RPC.

`rpcbind-configuration` [Tipo de datos]
: Tipo de datos que representa la configuración del servicio RPC Bind. Este tipo tiene los siguientes parámetros:

  `rpcbind` (predeterminado: `rpcbind`)
  : El paquete rpcbind usado.

  `warm-start?` (predeterminado: `#t`)
  : Si este parámetro es `#t`, el daemon leerá el archivo de estado durante el arranche, por tanto recargando la información del estado almacenada por la instancia previa.

## Pseudo-sistema de archivos pipefs

El sistema de archivos pipefs se usa para transferir datos relacionados con NFS entre el núcleo y los programas de espacio de usuaria.

`pipefs-service-type` [Variable]
: Un tipo de servicio para el pseudo-sistema de archivos pipefs.

`pipefs-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del servicio del pseudo-sistema de archivos pipefs. Este tipo tiene los siguientes parámetros:

`mount-point` (predeterminado: `"/var/lib/nfs/rpc_pipefs"`)
El directorio al que se debe asociar el sistema de archivos.

## Servicio del daemon GSS

El daemon *sistema de seguridad global* (GSS) proporciona fuertes garantías de seguridad para protocolos basados en RPC. Antes de intercambiar peticiones RPC el cliente debe establecer un contexto de seguridad. Habitualmente esto se lleva a cabo con el uso de la orden `kinit`o automáticamente durante el ingreso al sistema mediante el uso de servicios PAM (see Section 11.10.18 [Servicios Kerberos], page 457).

`gss-service-type` [Variable]
Un tipo de servicio para el daemon del sistema de seguridad global (GSS).

`gss-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del servicio del daemon GSS. Este tipo tiene los siguientes parámetros:

`nfs-utils` (predeterminado: `nfs-utils`)
Paquete en el que se encuentra la orden `rpc.gssd`.

`pipefs-directory` (predeterminado: `"/var/lib/nfs/rpc_pipefs"`)
El directorio donde el sistema de archivos pipefs debe montarse.

## Servicio del daemon IDMAP

El servicio del daemon idmap proporciona una asociación entre identificadores de usuaria y nombres de usuaria. Habitualmente es necesario para acceder sistemas de archivos montados con NFSv4.

`idmap-service-type` [Variable]
Un tipo de servicio para el daemon de asociación de identidades (IDMAP).

`idmap-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del servicio del daemon IDMAP. Este tipo tiene los siguientes parámetros:

`nfs-utils` (predeterminado: `nfs-utils`)
Paquete en el que se encuentra la orden `rpc.idmapd`.

`pipefs-directory` (predeterminado: `"/var/lib/nfs/rpc_pipefs"`)
El directorio donde el sistema de archivos pipefs debe montarse.

`domain` (predeterminado: `#f`)
El nombre de dominio local de NFSv4. Debe ser una cadena o `#f`. Si es `#f` el daemon usará el nombre de dominio totalmente cualificado de la máquina.

`verbosity` (predeterminado: `0`)
El nivel de información de los mensajes del daemon.

### 11.10.26 Samba Services

The `(gnu services samba)` module provides service definitions for Samba as well as additional helper services. Currently it provides the following services.

#### Samba

Samba (`https://www.samba.org`) provides network shares for folders and printers using the SMB/CIFS protocol commonly used on Windows. It can also act as an Active Directory Domain Controller (AD DC) for other hosts in an heterougenious network with different types of Computer systems.

`samba-service-type` [Variable]

The service type to enable the samba services `samba`, `nmbd`, `smbd` and `winbindd`. By default this service type does not run any of the Samba daemons; they must be enabled individually.

Below is a basic example that configures a simple, anonymous (unauthenticated) Samba file share exposing the `/public` directory.

**Tip:** The `/public` directory and its contents must be world readable/writable, so you'll want to run '`chmod -R 777 /public`' on it.

**Caution:** Such a Samba configuration should only be used in controlled environments, and you should not share any private files using it, as anyone connecting to your network would be able to access them.

```
(service samba-service-type (samba-configuration
                             (enable-smbd? #t)
                             (config-file (plain-file "smb.conf" "\
[global]
map to guest = Bad User
logging = syslog@1

[public]
browsable = yes
path = /public
read only = no
guest ok = yes
guest only = yes\n"))))
```

`samba-service-configuration` [Data Type]

Configuration record for the Samba suite.

`package` (default: `samba`)
: The samba package to use.

`config-file` (predeterminado: `#f`)
: The config file to use. To learn about its syntax, run '`man smb.conf`'.

`enable-samba?` (default: `#f`)
: Enable the `samba` daemon.

`enable-smbd?` (default: `#f`)
: Enable the `smbd` daemon.

`enable-nmbd?` (default: `#f`)
: Enable the `nmbd` daemon.

`enable-winbindd?` (default: `#f`)
: Enable the `winbindd` daemon.

## Web Service Discovery Daemon

The WSDD (Web Service Discovery daemon) implements the Web Services Dynamic Discovery (`http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html`) protocol that enables host discovery over Multicast DNS, similar to what Avahi does. It is a drop-in replacement for SMB hosts that have had SMBv1 disabled for security reasons.

`wsdd-service-type` [Variable]
: Service type for the WSD host daemon. The value for this service type is a `wsdd-configuration` record. The details for the `wsdd-configuration` record type are given below.

`wsdd-configuration` [Data Type]
: This data type represents the configuration for the wsdd service.

  `package` (default: `wsdd`)
  : The wsdd package to use.

  `ipv4only?` (default: `#f`)
  : Only listen to IPv4 addresses.

  `ipv6only` (default: `#f`)
  : Only listen to IPv6 addresses. Please note: Activating both options is not possible, since there would be no IP versions to listen to.

  `chroot` (predeterminado: `#f`)
  : Chroot into a separate directory to prevent access to other directories. This is to increase security in case there is a vulnerability in `wsdd`.

  `hop-limit` (default: `1`)
  : Limit to the level of hops for multicast packets. The default is *1* which should prevent packets from leaving the local network.

  `interface` (default: `'()`)
  : Limit to the given list of interfaces to listen to. By default wsdd will listen to all interfaces. Except the loopback interface is never used.

  `uuid-device` (default: `#f`)
  : The WSD protocol requires a device to have a UUID. Set this to manually assign the service a UUID.

  `domain` (predeterminado: `#f`)
  : Notify this host is a member of an Active Directory.

  `host-name` (predeterminado: `#f`)
  : Manually set the hostname rather than letting `wsdd` inherit this host's hostname. Only the host name part of a possible FQDN will be used in the default case.

`preserve-case?` (default: `#f`)
: By default `wsdd` will convert the hostname in workgroup to all uppercase. The opposite is true for hostnames in domains. Setting this parameter will preserve case.

`workgroup` (default: "*WORKGROUP*")
: Change the name of the workgroup. By default `wsdd` reports this host being member of a workgroup.

### 11.10.27 Integración continua

Cuirass (`https://guix.gnu.org/cuirass/`) is a continuous integration tool for Guix. It can be used both for development and for providing substitutes to others (see Section 5.3 [Sustituciones], page 46).

El módulo `(gnu services cuirass)` proporciona el siguiente servicio.

`cuirass-service-type` [Procedimiento]
: El tipo del servicio Cuirass. Su valor debe ser un objeto `cuirass-configuration`, como se describe a continuación.

To add build jobs, you have to set the `specifications` field of the configuration. For instance, the following example will build all the packages provided by the `my-channel` channel.

```
(define %cuirass-specs
  #~(list (specification
           (name "my-channel")
           (build '(channels my-channel))
           (channels
            (cons (channel
                   (name 'my-channel)
                   (url "https://my-channel.git"))
                  %default-channels)))))

(service cuirass-service-type
         (cuirass-configuration
          (specifications %especificacion-de-cuirass)))
```

To build the `linux-libre` package defined by the default Guix channel, one can use the following configuration.

```
(define %cuirass-specs
  #~(list (specification
           (name "my-linux")
           (build '(packages "linux-libre")))))

(service cuirass-service-type
         (cuirass-configuration
          (specifications %especificacion-de-cuirass)))
```

The other configuration possibilities, as well as the specification record itself are described in the Cuirass manual (see Section “Specifications” in *Cuirass*).

Mientras que la información de los trabajos de construcción se encuentra directamente en las especificaciones, la configuración global del proceso `cuirass` está accesible en otros campos de `cuirass-configuration`.

`cuirass-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de Cuirass.

`cuirass` (predeterminado: `cuirass`)
El paquete Cuirass usado.

`log-file` (predeterminado: `"/var/log/cuirass.log"`)
Localización del archivo de registro.

`web-log-file` (predeterminado: `"/var/log/cuirass-web.log"`)
Localización del archivo de registro usado por la interfaz web.

`cache-directory` (predeterminado: `"/var/cache/cuirass"`)
Localización de la caché del repositorio.

`user` (predeterminado: `"cuirass"`)
Propietaria del proceso `cuirass`.

`group` (predeterminado: `"cuirass"`)
Grupo propietario del proceso `cuirass`.

`interval` (predeterminado: `60`)
Número de segundos entre las consulta de repositorios seguida de los trabajos de Cuirass.

`ttl` (default: `2592000`)
Duration to keep build results' GC roots alive, in seconds.

`threads` (default: `#f`)
Number of kernel threads to use for Cuirass. The default value should be appropriate for most cases.

`parameters` (default: `#f`)
Read parameters from the given *parameters* file. The supported parameters are described here (see Section "Parameters" in *Cuirass*).

`remote-server` (default: `#f`)
A `cuirass-remote-server-configuration` record to use the build remote mechanism or `#f` to use the default build mechanism.

`database` (default: `"dbname=cuirass host=/var/run/postgresql"`)
Use *database* as the database containing the jobs and the past build results. Since Cuirass uses PostgreSQL as a database engine, *database* must be a string such as `"dbname=cuirass host=localhost"`.

`port` (predeterminado: `8081`)
Número de puerto usado por el servidor HTTP.

`host` (predeterminado: `"localhost"`)
Escucha en la interfaz de red de la dirección *host*. El comportamiento predeterminado es aceptar conexiones desde la red local.

`specifications` (predeterminada: `#~'()`)
: A gexp (see Section 8.12 [Expresiones-G], page 167) that evaluates to a list of specifications records. The specification record is described in the Cuirass manual (see Section "Specifications" in *Cuirass*).

`one-shot?` (predeterminado: `#f`)
: Evalúa las especificaciones y construye las derivaciones solo una vez.

`fallback?` (predeterminado: `#f`)
: Cuando la sustitución de un binario preconstruido falle, se intentará la construcción local de los paquetes.

`extra-options` (predeterminadas: `'()`)
: Extra options to pass when running the `cuirass register` process.

`web-extra-options` (default: `'()`)
: Extra options to pass when running the `cuirass web` process.

## Cuirass remote building

Cuirass supports two mechanisms to build derivations.

- Using the local Guix daemon. This is the default build mechanism. Once the build jobs are evaluated, they are sent to the local Guix daemon. Cuirass then listens to the Guix daemon output to detect the various build events.
- Using the remote build mechanism. The build jobs are not submitted to the local Guix daemon. Instead, a remote server dispatches build requests to the connect remote workers, according to the build priorities.

To enable this build mode a `cuirass-remote-server-configuration` record must be passed as `remote-server` argument of the `cuirass-configuration` record. The `cuirass-remote-server-configuration` record is described below.

This build mode scales way better than the default build mode. This is the build mode that is used on the GNU Guix build farm at `https://ci.guix.gnu.org`. It should be preferred when using Cuirass to build large amount of packages.

`cuirass-remote-server-configuration` [Data Type]
: Data type representing the configuration of the Cuirass remote-server.

  `backend-port` (default: `5555`)
  : The TCP port for communicating with `remote-worker` processes using ZMQ. It defaults to `5555`.

  `log-port` (default: `5556`)
  : The TCP port of the log server. It defaults to `5556`.

  `publish-port` (default: `5557`)
  : The TCP port of the publish server. It defaults to `5557`.

  `log-file` (default: `"/var/log/cuirass-remote-server.log"`)
  : Localización del archivo de registro.

  `cache` (default: `"/var/cache/cuirass/remote"`)
  : Use *cache* directory to cache build log files.

`log-expiry` (default: 6 months)
: The duration in seconds after which build logs collected by `cuirass remote-worker` may be deleted.

`trigger-url` (default: `#f`)
: Once a substitute is successfully fetched, trigger substitute baking at *trigger-url*.

`publish?` (predeterminado: `#t`)
: If set to false, do not start a publish server and ignore the `publish-port` argument. This can be useful if there is already a standalone publish server standing next to the remote server.

`public-key`
`private-key`
: Usa los *archivo*s específicos como el par de claves pública y privada usadas para firmar los elementos del almacén publicados.

At least one remote worker must also be started on any machine of the local network to actually perform the builds and report their status.

`cuirass-remote-worker-configuration` [Data Type]
Data type representing the configuration of the Cuirass remote-worker.

`cuirass` (predeterminado: `cuirass`)
: El paquete Cuirass usado.

`workers` (default: `1`)
: Start *workers* parallel workers.

`server` (predeterminada: `#f`)
: Do not use Avahi discovery and connect to the given `server` IP address instead.

`systems` (default: `(list (%current-system))`)
: Only request builds for the given *systems*.

`log-file` (default: `"/var/log/cuirass-remote-worker.log"`)
: Localización del archivo de registro.

`publish-port` (default: `5558`)
: The TCP port of the publish server. It defaults to `5558`.

`substitute-urls` (predeterminado: `%default-substitute-urls`)
: La lista de URLs donde se buscarán sustituciones por defecto.

`public-key`
`private-key`
: Usa los *archivo*s específicos como el par de claves pública y privada usadas para firmar los elementos del almacén publicados.

### Laminar

Laminar (`https://laminar.ohwg.net/`) is a lightweight and modular Continuous Integration service. It doesn't have a configuration web UI instead uses version-controllable configuration files and scripts.

Laminar encourages the use of existing tools such as bash and cron instead of reinventing them.

`laminar-service-type` [Variable]

The type of the Laminar service. Its value must be a `laminar-configuration` object, as described below.

All configuration values have defaults, a minimal configuration to get Laminar running is shown below. By default, the web interface is available on port 8080.

```
(service laminar-service-type)
```

`laminar-configuration` [Data Type]

Data type representing the configuration of Laminar.

`laminar` (default: `laminar`)
The Laminar package to use.

`home-directory` (default: `"/var/lib/laminar"`)
The directory for job configurations and run directories.

`supplementary-groups` (default: `()`)
Supplementary groups for the Laminar user account.

`bind-http` (default: `"*:8080"`)
The interface/port or unix socket on which laminard should listen for incoming connections to the web frontend.

`bind-rpc` (default: `"unix-abstract:laminar"`)
The interface/port or unix socket on which laminard should listen for incoming commands such as build triggers.

`title` (default: `"Laminar"`)
The page title to show in the web frontend.

`keep-rundirs` (default: `0`)
Set to an integer defining how many rundirs to keep per job. The lowest-numbered ones will be deleted. The default is 0, meaning all run dirs will be immediately deleted.

`archive-url` (default: `#f`)
The web frontend served by laminard will use this URL to form links to artefacts archived jobs.

`base-url` (default: `#f`)
Base URL to use for links to laminar itself.

## 11.10.28 Servicios de gestión de energía

## Power Profiles Daemon

The `(gnu services pm)` module provides a Guix service definition for the Linux Power Profiles Daemon, which makes power profiles handling available over D-Bus.

The available profiles consist of the default '`balanced`' mode, a '`power-saver`' mode and on supported systems a '`performance`' mode.

> **Importante:** The `power-profiles-daemon` conflicts with other power management tools like `tlp`. Using both together is not recommended.

`power-profiles-daemon-service-type` [Variable]

This is the service type for the Power Profiles Daemon (`https://gitlab.freedesktop.org/upower/power-profiles-daemon/`). The value for this service is a `power-profiles-daemon-configuration`.

To enable the Power Profiles Daemon with default configuration add this line to your services:

```
(service power-profiles-daemon-service-type)
```

`power-profiles-daemon-configuration` [Data Type]

Data type representing the configuration of `power-profiles-daemon-service-type`.

`power-profiles-daemon` (default: `power-profiles-daemon`) (type: file-like)
Package object of power-profiles-daemon.

## Daemon TLP

El módulo `(gnu services pm)` proporciona una definición de servicio Guix para la herramienta de gestión de energía de Linux TLP.

TLP activa varios modos de ahorro de energía en el núcleo y en espacio de usuaria. Al contrario que `upower-service`, no es una herramienta de monitorización pasiva, puesto que aplicará una nueva configuración personalizada cada vez que se detecte una nueva fuente de energía/alimentación. Puede encontrar más información en la página de TLP (`https://linrunner.de/en/tlp/tlp.html`).

`tlp-service-type` [Variable]

EL tipo de servicio para la herramienta TLP. La configuración predeterminada está optimizada para la vida de la batería en la mayoría de los sistemas, pero puede modificarla al completo con las opciones que desee proporcionando un objeto `tlp-configuration` válido:

```
(service tlp-service-type
         (tlp-configuration
          (cpu-scaling-governor-on-ac (list "performance"))
          (sched-powersave-on-bat? #t)))
```

Each parameter definition is preceded by its type; for example, '`boolean foo`' indicates that the `foo` parameter should be specified as a boolean. Types starting with `maybe-` denote parameters that won't show up in TLP config file when their value is left unset, or is explicitly set to the `%unset-value` value.

Los campos disponibles de `tlp-configuration` son:

`package tlp` [parámetro de `tlp-configuration`]
El paquete TLP.

`boolean tlp-enable?` [parámetro de `tlp-configuration`]
Proporcione un valor verdadero si desea activar TLP.

El valor predeterminado es '`#t`'

`string tlp-default-mode` [parámetro de `tlp-configuration`]
Modo predeterminado cuando no se puede detectar una fuente de alimentación. Las alternativas son AC (corriente alterna) y BAT (batería).

El valor predeterminado es '`"AC"`' (corriente alterna).

`entero-no-negativo` [parámetro de `tlp-configuration`]
`disk-idle-secs-on-ac`
Número de segundos que el núcleo Linux debe esperar desde que el disco se queda en espera, antes de sincronizar en corriente alterna (AC).

El valor predeterminado es '`0`'.

`entero-no-negativo` [parámetro de `tlp-configuration`]
`disk-idle-secs-on-bat`
Igual que `disk-idle-ac` pero en modo BAT (batería).

El valor predeterminado es '`2`'.

`entero-no-negativo` [parámetro de `tlp-configuration`]
`max-lost-work-secs-on-ac`
Periodicidad de la evacuación de las páginas sucias, expresada en segundos.

El valor predeterminado es '`15`'.

`entero-no-negativo` [parámetro de `tlp-configuration`]
`max-lost-work-secs-on-bat`
Igual que `max-lost-work-secs-on-ac` pero en modo BAT (batería).

El valor predeterminado es '`60`'.

`maybe-lista-cadena-separada-espacios` [parámetro de `tlp-configuration`]
`cpu-scaling-governor-on-ac`
Gobernador de escalado de frecuencia del procesador en modo de corriente alterna (AC). Con el controlador intel_pstate, las alternativas son "powersave" (ahorro de energía) y "performance" (rendimiento). Con el controlador acpi-cpufreq, las alternativas son "ondemand" (bajo demanda), "powersave", "performance" y "conservative" (conservativo).

El valor predeterminado es '`disabled`'.

`maybe-lista-cadena-separada-espacios` [parámetro de `tlp-configuration`]
`cpu-scaling-governor-on-bat`
Igual que `max-lost-work-secs-on-ac` pero en modo BAT (batería).

El valor predeterminado es '`disabled`'.

`maybe-entero-no-negativo` [parámetro de `tlp-configuration`]
`cpu-scaling-min-freq-on-ac`

Establece la frecuencia mínima disponible para el controlador de escalado en AC.

El valor predeterminado es '`disabled`'.

`maybe-entero-no-negativo` [parámetro de `tlp-configuration`]
`cpu-scaling-max-freq-on-ac`

Establece la frecuencia máxima disponible para el controlador de escalado en AC.

El valor predeterminado es '`disabled`'.

`maybe-entero-no-negativo` [parámetro de `tlp-configuration`]
`cpu-scaling-min-freq-on-bat`

Establece la frecuencia mínima disponible para el controlador de escalado en BAT.

El valor predeterminado es '`disabled`'.

`maybe-entero-no-negativo` [parámetro de `tlp-configuration`]
`cpu-scaling-max-freq-on-bat`

Establece la frecuencia máxima disponible para el controlador de escalado en BAT.

El valor predeterminado es '`disabled`'.

`maybe-entero-no-negativo` [parámetro de `tlp-configuration`]
`cpu-min-perf-on-ac`

Limita el estado-P mínimo para controlar la disipación de potencia del procesador en modo AC. Los valores se indican como un porcentaje de rendimiento disponible.

El valor predeterminado es '`disabled`'.

`maybe-entero-no-negativo` [parámetro de `tlp-configuration`]
`cpu-max-perf-on-ac`

Limita el estado-P máximo para controlar la disipación de potencia del procesador en modo AC. Los valores se indican como un porcentaje de rendimiento disponible.

El valor predeterminado es '`disabled`'.

`maybe-entero-no-negativo` [parámetro de `tlp-configuration`]
`cpu-min-perf-on-bat`

Igual que `cpu-min-perf-on-ac` pero en modo BAT (batería).

El valor predeterminado es '`disabled`'.

`maybe-entero-no-negativo` [parámetro de `tlp-configuration`]
`cpu-max-perf-on-bat`

Igual que `cpu-max-perf-on-ac` pero en modo BAT (batería).

El valor predeterminado es '`disabled`'.

`maybe-boolean cpu-boost-on-ac?` [parámetro de `tlp-configuration`]

Activa la característica "turbo boost" del procesador en modo AC (corriente alterna).

El valor predeterminado es '`disabled`'.

`maybe-boolean cpu-boost-on-bat?` [parámetro de `tlp-configuration`]

Igual que `cpu-boost-on-ac` pero en modo BAT (batería).

El valor predeterminado es '`disabled`'.

`boolean sched-powersave-on-ac?` [parámetro de `tlp-configuration`]
Permite al núcleo Linux minimizar el número de núcleos/hyper-thread del procesador usados bajo condiciones de baja carga.

El valor predeterminado es '`#f`'

`boolean sched-powersave-on-bat?` [parámetro de `tlp-configuration`]
Igual que `sched-powersave-on-ac?` pero en modo BAT (batería).

El valor predeterminado es '`#t`'

`boolean nmi-watchdog?` [parámetro de `tlp-configuration`]
Activa el proceso guardián (watchdog) NMI del núcleo Linux.

El valor predeterminado es '`#f`'

`maybe-string phc-controls` [parámetro de `tlp-configuration`]
Para núcleos Linux con el parche PHC aplicado, cambia los voltajes del procesador. Un valor de ejemplo sería '`"F:V F:V F:V F:V"`'.

El valor predeterminado es '`disabled`'.

`string energy-perf-policy-on-ac` [parámetro de `tlp-configuration`]
Set CPU performance versus energy saving policy on AC. Alternatives are performance, normal, powersave.

El valor predeterminado es '`"performance"`'.

`string energy-perf-policy-on-bat` [parámetro de `tlp-configuration`]
Igual que `energy-perf-policy-ac` pero en modo BAT (batería).

El valor predeterminado es '`"powersave"`'.

`lista-cadena-separada-espacios` [parámetro de `tlp-configuration`]
`disks-devices`
Dispositivos de disco duro.

`lista-cadena-separada-espacios` [parámetro de `tlp-configuration`]
`disk-apm-level-on-ac`
Nivel de APM (gestión avanzada de energía) del disco duro.

`lista-cadena-separada-espacios` [parámetro de `tlp-configuration`]
`disk-apm-level-on-bat`
Igual que `disk-apm-bat` pero en modo BAT (batería).

`maybe-lista-cadena-separada-espacios` [parámetro de `tlp-configuration`]
`disk-spindown-timeout-on-ac`
Plazo para la parada rotacional del disco duro. Se debe especificar un valor por cada disco duro declarado.

El valor predeterminado es '`disabled`'.

`maybe-lista-cadena-separada-espacios` [parámetro de `tlp-configuration`]
`disk-spindown-timeout-on-bat`
Igual que `disk-spindown-timeout-on-ac` pero en modo BAT (batería).

El valor predeterminado es '`disabled`'.

`maybe-lista-cadena-separada-espacios` [parámetro de `tlp-configuration`]
`disk-iosched`

Selecciona el planificador de E/S para dispositivos de disco. Se debe especificar un valor por cada disco duro declarado. Ejemplos de alternativas son “cfq”, “deadline” y “noop”.

El valor predeterminado es ‘`disabled`’.

`string sata-linkpwr-on-ac` [parámetro de `tlp-configuration`]

Nivel de gestión agresiva de energía del enlace (ALPM) de SATA. Las alternativas son “min_power” (energía mínima), “medium_power” (energía media) y “max_performance” (máximo rendimiento).

El valor predeterminado es ‘`"max_performance"`’.

`string sata-linkpwr-on-bat` [parámetro de `tlp-configuration`]

Igual que `sata-linkpwr-ac` pero en modo BAT (batería).

El valor predeterminado es ‘`"min_power"`’.

`maybe-string sata-linkpwr-blacklist` [parámetro de `tlp-configuration`]

Excluye los dispositivos SATA especificados de la gestión de energía del enlace.

El valor predeterminado es ‘`disabled`’.

`maybe-on-off-boolean` [parámetro de `tlp-configuration`]
`ahci-runtime-pm-on-ac?`

Activa la gestión de energía de tiempo de ejecución para controladores AHCI y discos en modo AC.

El valor predeterminado es ‘`disabled`’.

`maybe-on-off-boolean` [parámetro de `tlp-configuration`]
`ahci-runtime-pm-on-bat?`

Igual que `ahci-runtime-pm-on-ac` pero en modo BAT (batería).

El valor predeterminado es ‘`disabled`’.

`entero-no-negativo` [parámetro de `tlp-configuration`]
`ahci-runtime-pm-timeout`

Segundos de inactividad antes de suspender el disco.

El valor predeterminado es ‘`15`’.

`string pcie-aspm-on-ac` [parámetro de `tlp-configuration`]

Nivel de gestión de energía de estado activo de PCI Express. Las alternativas son “default” (predeterminado), “performance” (rendimiento) y “powersave” (ahorro de energía).

El valor predeterminado es ‘`"performance"`’.

`string pcie-aspm-on-bat` [parámetro de `tlp-configuration`]

Igual que `pcie-aspm-ac` pero en modo BAT (batería).

El valor predeterminado es ‘`"powersave"`’.

`maybe-non-negative-integer` [`tlp-configuration` parameter]
`start-charge-thresh-bat0`

Percentage when battery 0 should begin charging. Only supported on some laptops.

El valor predeterminado es '`disabled`'.

`maybe-non-negative-integer` [`tlp-configuration` parameter]
`stop-charge-thresh-bat0`

Percentage when battery 0 should stop charging. Only supported on some laptops.

El valor predeterminado es '`disabled`'.

`maybe-non-negative-integer` [`tlp-configuration` parameter]
`start-charge-thresh-bat1`

Percentage when battery 1 should begin charging. Only supported on some laptops.

El valor predeterminado es '`disabled`'.

`maybe-non-negative-integer` [`tlp-configuration` parameter]
`stop-charge-thresh-bat1`

Percentage when battery 1 should stop charging. Only supported on some laptops.

El valor predeterminado es '`disabled`'.

`string radeon-power-profile-on-ac` [parámetro de `tlp-configuration`]

Nivel de velocidad de reloj de gráficos Radeon. Las alternativas son "low" (bajo), "mid" (medio), "high" (alto), "auto" (automático) y "default" (predeterminado).

El valor predeterminado es '`"high"`'.

`string radeon-power-profile-on-bat` [parámetro de `tlp-configuration`]

Igual que `radeon-power-ac` pero en modo BAT (batería).

El valor predeterminado es '`"low"`'.

`string radeon-dpm-state-on-ac` [parámetro de `tlp-configuration`]

Método de gestión de energía dinámica (DPM) de Radeon. Las alternativas son "battery" (batería) y "performance" (rendimiento).

El valor predeterminado es '`"performance"`'.

`string radeon-dpm-state-on-bat` [parámetro de `tlp-configuration`]

Igual que `radeon-dpm-state-ac` pero en modo BAT (batería).

El valor predeterminado es '`"battery"`'.

`string radeon-dpm-perf-level-on-ac` [parámetro de `tlp-configuration`]

Nivel de rendimiento del DPM de Radeon. Las alternativas son "auto" (automático), "low" (bajo) y "high" (alto).

El valor predeterminado es '`"auto"`'.

`string radeon-dpm-perf-level-on-bat` [parámetro de `tlp-configuration`]

Igual que `radeon-dpm-perf-ac` pero en modo BAT (batería).

El valor predeterminado es '`"auto"`'.

`on-off-boolean wifi-pwr-on-ac?` [parámetro de `tlp-configuration`]
Modo de ahorro de energía de Wifi.

El valor predeterminado es '`#f`'

`on-off-boolean wifi-pwr-on-bat?` [parámetro de `tlp-configuration`]
Igual que `wifi-power-ac?` pero en modo BAT (batería).

El valor predeterminado es '`#t`'

`y-n-boolean wol-disable?` [parámetro de `tlp-configuration`]
Desactiva el encendido desde la red local (wake on LAN).

El valor predeterminado es '`#t`'

`entero-no-negativo` [parámetro de `tlp-configuration`]
`sound-power-save-on-ac`
Duración en segundos del plazo antes de activar el ahorro de energía de audio en dispositivos Intel HDA y AC97. El valor 0 desactiva el ahorro de energía.

El valor predeterminado es '`0`'.

`entero-no-negativo` [parámetro de `tlp-configuration`]
`sound-power-save-on-bat`
Igual que `sound-powersave-ac` pero en modo BAT (batería).

El valor predeterminado es '`1`'.

`y-n-boolean` [parámetro de `tlp-configuration`]
`sound-power-save-controller?`
Desactiva el controlador en modo de ahorro de energía en dispositivos Intel HDA.

El valor predeterminado es '`#t`'

`boolean bay-poweroff-on-bat?` [parámetro de `tlp-configuration`]
Activa las unidades ópticas en UltraBay/MediaBay en modo BAT. La unidad puede volver a alimentarse liberando (y reinsertando) la palanca de eyección o presionando el botón de eyección de disco en modelos más modernos.

El valor predeterminado es '`#f`'

`string bay-device` [parámetro de `tlp-configuration`]
Nombre de la unidad de dispositivos ópticos a apagar.

El valor predeterminado es '`"sr0"`'.

`string runtime-pm-on-ac` [parámetro de `tlp-configuration`]
Gestión de energía en tiempo de ejecución para dispositivos de bus PCI(e). Las alternativas son “on” y “auto”.

El valor predeterminado es '`"on"`'.

`string runtime-pm-on-bat` [parámetro de `tlp-configuration`]
Igual que `runtime-pm-ac` pero en modo BAT (batería).

El valor predeterminado es '`"auto"`'.

`boolean runtime-pm-all?` [parámetro de `tlp-configuration`]
Gestión de energía en tiempo de ejecución (Runtime Power Management) para todos los dispositivos del bus PCI(e), excepto los excluidos.

El valor predeterminado es '`#t`'

`maybe-lista-cadena-separada-espacios runtime-pm-blacklist` [parámetro de `tlp-configuration`]
Excluye las direcciones de dispositivo PCI(e) especificadas de la gestión de energía en tiempo de ejecución (Runtime Power Management).

El valor predeterminado es '`disabled`'.

`lista-cadena-separada-espacios runtime-pm-driver-blacklist` [parámetro de `tlp-configuration`]
Excluye los dispositivos PCI(e) asignados a los controladores especificados de la gestión de energía en tiempo de ejecución (Runtime Power Management).

`boolean usb-autosuspend?` [parámetro de `tlp-configuration`]
Permite la suspensión automática de USB.

El valor predeterminado es '`#t`'

`maybe-string usb-blacklist` [parámetro de `tlp-configuration`]
Excluye los dispositivos especificados de la suspensión automática de USB.

El valor predeterminado es '`disabled`'.

`boolean usb-blacklist-wwan?` [parámetro de `tlp-configuration`]
Excluye los dispositivos WWAN de la suspensión automática de USB.

El valor predeterminado es '`#t`'

`maybe-string usb-whitelist` [parámetro de `tlp-configuration`]
Incluye los dispositivos especificados en la suspensión automática de USB, incluso cuando están excluidos por el controlador o a través de `usb-blacklist-wwan?`.

El valor predeterminado es '`disabled`'.

`maybe-boolean usb-autosuspend-disable-on-shutdown?` [parámetro de `tlp-configuration`]
Activa la suspensión automática de USB antes del apagado.

El valor predeterminado es '`disabled`'.

`boolean restore-device-state-on-startup?` [parámetro de `tlp-configuration`]
Restablece el estado de los dispositivos de radio (bluetooth, wifi, wwan) previo al apagado durante el arranque del sistema.

El valor predeterminado es '`#f`'

### Daemon Thermald

El módulo `(gnu services pm)` proporciona una interfaz con thermald, un servicio de escalado de frecuencia de la CPU que ayuda a prevenir el sobrecalentamiento.

`thermald-service-type` [Variable]

Este es el tipo de servicio para thermald (`https://01.org/linux-thermal-daemon/`), el daemon Thermal de Linux, que es responsable del control del estado térmico de los procesadores y la prevención del sobrecalentamiento.

`thermald-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `thermald-service-type`.

`adaptive?` (default: `#f`)
: Use DPTF (Dynamic Power and Thermal Framework) adaptive tables when present.

`ignore-cpuid-check?` (predeterminado: `#f`)
: Ignora la comprobación de cpuid durante la comprobación de procesadores permitidos.

`thermald` (predeterminado: *thermald*)
: El objeto paquete de thermald.

## 11.10.29 Servicios de audio

El módulo `(gnu services audio)` proporciona un servicio para iniciar MPD (el daemon de reproducción de música).

### Daemon de reproducción de música (MPD)

El daemon de reproducción de música (MPD) es un servicio que puede reproducir música mientras se controla desde la máquina local o sobre una red por una multitud de clientes.

The following example shows the simplest configuration to locally expose, via PulseAudio, a music collection kept at `/srv/music`, with `mpd` running as the default '`mpd`' user. This user will spawn its own PulseAudio daemon, which may compete for the sound card access with that of your own user. In this configuration, you may have to stop the playback of your user audio applications to hear MPD's output and vice-versa.

```
(service mpd-service-type
         (mpd-configuration
          (music-directory "/srv/music")))
```

**Importante:** The music directory must be readable to the MPD user, by default, '`mpd`'. Permission problems will be reported via '`Permission denied`' errors in the MPD logs, which appear in `/var/log/messages` by default.

Most MPD clients will trigger a database update upon connecting, but you can also use the `update` action do to so:

```
herd update mpd
```

All the MPD configuration fields are documented below, and a more complex example follows.

`mpd-service-type` [Variable]
El tipo de servicio para `mpd`.

`mpd-configuration` [Tipo de datos]
Available `mpd-configuration` fields are:

`package` (default: `mpd`) (type: file-like)
The MPD package.

`user` (type: user-account)
Usuaria que ejecuta mpd.

`group` (type: user-group)
The group to run mpd as.

The default `%mpd-group` is a system group with name "mpd".

`shepherd-requirement` (default: `'()`) (type: list-of-symbols)
A list of symbols naming Shepherd services that this service will depend on.

`environment-variables` (default:
`'("PULSE_CLIENTCONFIG=/etc/pulse/client.conf"`
`"PULSE_CONFIG=/etc/pulse/daemon.conf")`) (type: list-of-strings)
A list of strings specifying environment variables.

`log-file` (type: maybe-string)
The location of the log file. Unless specified, logs are sent to the local syslog daemon. Alternatively, a log file name can be specified, for example `/var/log/mpd.log`.

`log-level` (type: maybe-string)
Supress any messages below this threshold. The available values, in decreasing order of verbosity, are: `verbose`, `info`, `notice`, `warning` and `error`.

`music-directory` (type: maybe-string)
El directorio para buscar archivos de música.

`music-dir` (type: maybe-string)
El directorio para buscar archivos de música.

`playlist-directory` (type: maybe-string)
El directorio para almacenar listas de reproducción.

`playlist-dir` (type: maybe-string)
El directorio para almacenar listas de reproducción.

`db-file` (type: maybe-string)
The location of the music database. When left unspecified, `~/.cache/db` is used.

`state-file` (type: maybe-string)
La localización del archivo que almacena el estado actual de MPD.

`sticker-file` (type: maybe-string)
: La localización de la base de datos de pegatinas.

`default-port` (default: `6600`) (type: maybe-port)
: The default port to run mpd on.

`endpoints` (type: maybe-list-of-strings)
: The addresses that mpd will bind to. A port different from *default-port* may be specified, e.g. `localhost:6602` and IPv6 addresses must be enclosed in square brackets when a different port is used. To use a Unix domain socket, an absolute path or a path starting with `~` can be specified here.

`address` (type: maybe-string)
: Dirección a la que mpd se asociará. Para usar un socket de dominio de Unix puede especificarse una ruta absoluta.

`database` (type: maybe-mpd-plugin)
: MPD database plugin configuration.

`partitions` (default: `'()`) (type: list-of-mpd-partition)
: List of MPD "partitions".

`neighbors` (default: `'()`) (type: list-of-mpd-plugin)
: List of MPD neighbor plugin configurations.

`inputs` (default: `'()`) (type: list-of-mpd-plugin)
: List of MPD input plugin configurations.

`archive-plugins` (default: `'()`) (type: list-of-mpd-plugin)
: List of MPD archive plugin configurations.

`auto-update?` (type: maybe-boolean)
: Whether to automatically update the music database when files are changed in the *music-directory*.

`input-cache-size` (type: maybe-string)
: MPD input cache size.

`decoders` (default: `'()`) (type: list-of-mpd-plugin)
: List of MPD decoder plugin configurations.

`resampler` (type: maybe-mpd-plugin)
: MPD resampler plugin configuration.

`filters` (default: `'()`) (type: list-of-mpd-plugin)
: List of MPD filter plugin configurations.

`outputs` (type: list-of-mpd-plugin-or-output)
: Las salidas de audio que MPD puede usar. De manera predeterminada es una salida de audio única usando pulseaudio.

`playlist-plugins` (default: `'()`) (type: list-of-mpd-plugin)
: List of MPD playlist plugin configurations.

`extra-options` (default: `'()`) (type: alist)
An association list of option symbols/strings to string values to be appended to the configuration.

`mpd-plugin` [Data Type]
Data type representing a `mpd` plugin.

`plugin` (type: maybe-string)
Plugin name.

`name` (type: maybe-string)
Name.

`enabled?` (type: maybe-boolean)
Whether the plugin is enabled/disabled.

`extra-options` (default: `'()`) (type: alist)
An association list of option symbols/strings to string values to be appended to the plugin configuration. See MPD plugin reference (`https://mpd.readthedocs.io/en/latest/plugins.html`) for available options.

`mpd-partition` [Data Type]
Data type representing a `mpd` partition.

`name` (type: string)
Partition name.

`extra-options` (default: `'()`) (type: alist)
An association list of option symbols/strings to string values to be appended to the partition configuration. See Configuring Partitions (`https://mpd.readthedocs.io/en/latest/user.html#configuring-partitions`) for available options.

`mpd-output` [Tipo de datos]
Available `mpd-output` fields are:

`name` (default: `"MPD"`) (type: string)
Nombre de la salida de audio.

`type` (default: `"pulse"`) (type: string)
Tipo de la salida de audio.

`enabled?` (default: `#t`) (type: boolean)
Especifica si esta salida de audio se activa cuando se inicia MPD. De manera predeterminada se activan todas las salidas de audio. Esta es la configuración predeterminada cuando no existe un archivo de estado; con un archivo de estado se restaura el estado anterior.

`format` (type: maybe-string)
Force a specific audio format on output. See Global Audio Format (`https://mpd.readthedocs.io/en/latest/user.html#audio-output-format`) for a more detailed description.

`tags?` (default: `#t`) (type: boolean)
: Si se proporciona el valor `#f` MPD no envia etiquetas a esta salida. Es útil únicamente para módulos de salida que pueden recibir etiquetas, por ejemplo el módulo de salida `httpd`.

`always-on?` (default: `#f`) (type: boolean)
: If set to `#t`, then MPD attempts to keep this audio output always open. This may be useful for streaming servers, when you don't want to disconnect all listeners even when playback is accidentally stopped.

`mixer-type` (type: maybe-string)
: This field accepts a string that specifies which mixer should be used for this audio output: the `hardware` mixer, the `software` mixer, the `null` mixer (allows setting the volume, but with no effect; this can be used as a trick to implement an external mixer External Mixer) or no mixer (`none`). When left unspecified, a `hardware` mixer is used for devices that support it.

`replay-gain-handler` (type: maybe-string)
: This field accepts a string that specifies how Replay Gain (`https://mpd.readthedocs.io/en/latest/user.html#replay-gain`) is to be applied. `software` uses an internal software volume control, `mixer` uses the configured (hardware) mixer control and `none` disables replay gain on this audio output.

`extra-options` (default: `'()`) (type: alist)
: An association list of option symbols/strings to string values to be appended to the audio output configuration.

The following example shows a configuration of `mpd` that configures some of its plugins and provides a HTTP audio streaming output.

```
(service mpd-service-type
         (mpd-configuration
           (outputs
             (list (mpd-output
                     (name "streaming")
                     (type "httpd")
                     (mixer-type 'null)
                     (extra-options
                      `((encoder . "vorbis")
                        (port    . "8080"))))))
           (decoders
             (list (mpd-plugin
                     (plugin "mikmod")
                     (enabled? #f))
                   (mpd-plugin
                     (plugin "openmpt")
                     (enabled? #t)
                     (extra-options `((repeat-count . -1)
```

```
                                        (interpolation-filter . 1))))))
              (resampler (mpd-plugin
                          (plugin "libsamplerate")
                          (extra-options `((type . 0)))))))
```

## myMPD

myMPD (`https://jcorporation.github.io/myMPD/`) is a web server frontend for MPD that provides a mobile friendly web client for MPD.

The following example shows a myMPD instance listening on port 80, with album cover caching disabled.

```
(service mympd-service-type
         (mympd-configuration
          (port 80)
          (covercache-ttl 0)))
```

`mympd-service-type` [Variable]

The service type for `mympd`.

`mympd-configuration` [Data Type]

Available `mympd-configuration` fields are:

`package` (default: `mympd`) (type: file-like)
The package object of the myMPD server.

`shepherd-requirement` (default: `'()`) (type: list-of-symbols)
This is a list of symbols naming Shepherd services that this service will depend on.

`user` (default: `%mympd-user`) (type: user-account)
Owner of the `mympd` process.

The default `%mympd-user` is a system user with the name "mympd", who is a part of the group *group* (see below).

`group` (default: `%mympd-group`) (type: user-group)
Owner group of the `mympd` process.

The default `%mympd-group` is a system group with name "mympd".

`work-directory` (default: `"/var/lib/mympd"`) (type: string)
Where myMPD will store its data.

`cache-directory` (default: `"/var/cache/mympd"`) (type: string)
Where myMPD will store its cache.

`acl` (type: maybe-mympd-ip-acl)
ACL to access the myMPD webserver.

`covercache-ttl` (default: `31`) (type: maybe-integer)
How long to keep cached covers, `0` disables cover caching.

`http?` (default: `#t`) (type: boolean)
HTTP support.

`host` (default: `"[::]"`) (type: string)
Host name to listen on.

`port` (default: `80`) (type: maybe-port)
HTTP port to listen on.

`log-level` (default: `5`) (type: integer)
How much detail to include in logs, possible values: `0` to `7`.

`log-to` (type: maybe-string)
Where to send logs. Unless specified, the service logs to the local syslog service under the '`daemon`' facility. Alternatively, a log file name can be specified, for example `/var/log/mympd.log`.

`lualibs` (default: `"all"`) (type: maybe-string)
See `https://jcorporation.github.io/myMPD/scripting/#lua-standard-libraries`.

`uri` (type: maybe-string)
Override URI to myMPD. See `https://github.com/jcorporation/myMPD/issues/950`.

`script-acl` (default: `(mympd-ip-acl (allow '("127.0.0.1")))`) (type: maybe-mympd-ip-acl)
ACL to access the myMPD script backend.

`ssl?` (default: `#f`) (type: boolean)
SSL/TLS support.

`ssl-port` (default: `443`) (type: maybe-port)
Port to listen for HTTPS.

`ssl-cert` (type: maybe-string)
Path to PEM encoded X.509 SSL/TLS certificate (public key).

`ssl-key` (type: maybe-string)
Path to PEM encoded SSL/TLS private key.

`pin-hash` (type: maybe-string)
SHA-256 hashed pin used by myMPD to control settings access by prompting a pin from the user.

`save-caches?` (type: maybe-boolean)
Whether to preserve caches between service restarts.

`mympd-ip-acl` [Data Type]
Available `mympd-ip-acl` fields are:

`allow` (default: `'()`) (type: list-of-strings)
Allowed IP addresses.

`deny` (default: `'()`) (type: list-of-strings)
Disallowed IP addresses.

### 11.10.30 Servicios de virtualización

El módulo `(gnu services virtualization)` proporciona servicios para los daemon libvirt y virtlog, así como otros servicios relacionados con la virtualización.

### Daemon de Libvirt

`libvirtd` is the server side daemon component of the libvirt virtualization management system. This daemon runs on host servers and performs required management tasks for virtualized guests. To connect to the libvirt daemon as an unprivileged user, it must be added to the '`libvirt`' group, as shown in the example below.

`libvirt-service-type` [Variable]

Este es el tipo para el daemon de libvirt (`https://libvirt.org`). Su valor debe ser un objeto `libvirt-configuration`.

```
(users (cons (user-account
              (name "user")
              (group "users")
              (supplementary-groups '("libvirt"
                                      "audio" "video" "wheel")))
             %base-user-accounts))
(service libvirt-service-type
         (libvirt-configuration
          (tls-port "16555")))
```

Los campos disponibles de `libvirt-configuration` son:

`package libvirt` [parámetro de `libvirt-configuration`]

Paquete libvirt.

`boolean listen-tls?` [parámetro de `libvirt-configuration`]

Opción para la escucha de conexiones seguras TLS en el puerto TCP/IP público. Debe haberse proporcionado valor a `listen` para que tenga algún efecto.

Es necesario configurar una autoridad de certificación (CA) y emitir certificados de servidor antes de usar esta característica.

El valor predeterminado es '`#t`'

`boolean listen-tcp?` [parámetro de `libvirt-configuration`]

Escucha de conexiones TCP sin cifrar en el puerto TCP/IP público. Debe haberse proporcionado valor a `listen` para que tenga algún efecto.

El uso del socket TCP necesita de manera predeterminada identificación SASL. Únicamente se permiten mecanismos SASL que implementen cifrado de datos. Estos son DIGEST_MD5 y GSSAPI (Kerberos5).

El valor predeterminado es '`#f`'

`string tls-port` [parámetro de `libvirt-configuration`]

Puerto en el que se aceptan conexiones seguras. Puede ser un número de puerto o un nombre de servicio.

El valor predeterminado es '`"16514"`'.

`string tcp-port` [parámetro de `libvirt-configuration`]
Puerto en el que se aceptan conexiones inseguras. Puede ser un número de puerto o un nombre de servicio.

El valor predeterminado es '`"16509"`'.

`string listen-addr` [parámetro de `libvirt-configuration`]
Dirección IP o nombre de máquina usado para las conexiones de clientes.

El valor predeterminado es '`"0.0.0.0"`'.

`boolean mdns-adv?` [parámetro de `libvirt-configuration`]
Opción que determina el anuncio mDNS del servicio libvirt.

De manera alternativa puede desactivarse para todos los servicios en una máquina parando el daemon Avahi.

El valor predeterminado es '`#f`'

`string mdns-name` [parámetro de `libvirt-configuration`]
Nombre predeterminado del anuncio mDNS. Debe ser único en la red de distribución inmediata.

El valor predeterminado es '`"Virtualization Host <hostname>"`'.<

`string unix-sock-group` [parámetro de `libvirt-configuration`]
Grupo propietario del socket de dominio de UNIX. Puede usarse para permitir a un conjunto de usuarias "de confianza" acceso a las funcionalidades de gestión sin convertirse en root.

Defaults to '`"libvirt"`'.

`string unix-sock-ro-perms` [parámetro de `libvirt-configuration`]
Permisos del socket UNIX de sólo lectura[13]. Se usa únicamente para monitorizar el estado de las máquinas virtuales.

El valor predeterminado es '`"0777"`'.

`string unix-sock-rw-perms` [parámetro de `libvirt-configuration`]
Permisos del socket UNIX de lectura/escritura[14]. El valor predeterminado únicamente permite acceso a root. Si PolicyKit se encuentra activo en el socket, el valor predeterminado cambiará para permitir acceso universal (es decir, 0777).

El valor predeterminado es '`"0770"`'.

`string unix-sock-admin-perms` [parámetro de `libvirt-configuration`]
Permisos del socket UNIX de administración. El valor predeterminado únicamente permite acceso a la propietaria (root), no lo cambie a menos que esté completamente segura de a quién expone el acceso.

El valor predeterminado es '`"0777"`'.

`string unix-sock-dir` [parámetro de `libvirt-configuration`]
Directorio en el que los sockets se encuentran/crean.

El valor predeterminado es '`"/var/run/libvirt"`'.

[13] R/O: Read-Only en inglés.

[14] R/W: Read-Write en inglés.

`string auth-unix-ro` [parámetro de `libvirt-configuration`]
Esquema de identificación para los sockets de solo-lectura de UNIX. Los permisos predeterminados del socket permiten la conexión de cualquier usuaria.

El valor predeterminado es ‘`"polkit"`’.

`string auth-unix-rw` [parámetro de `libvirt-configuration`]
Esquema de identificación para los sockets de lectura/escritura de UNIX. Los permisos predeterminados del socket permiten la conexión únicamente a root. Si se activó en la compilación de libvirt la interoperabilidad con PolicyKit, el valor predeterminado es usar la identificación “policykit”.

El valor predeterminado es ‘`"polkit"`’.

`string auth-tcp` [parámetro de `libvirt-configuration`]
Esquema de identificación para los sockets TCP. Si no activa SASL, todo el tráfico TCP estará en texto plano. No lo haga más allá de un escenario de desarrollo/pruebas.

El valor predeterminado es ‘`"sasl"`’.

`string auth-tls` [parámetro de `libvirt-configuration`]
Esquema de identificación para los sockets TLS. Los sockets TLS ya se encuentran cifrados gracias a la capa TLS, y una identificación limitada se realiza con los certificados.

También es posible hacer uso de cualquier mecanismo de identificación SASL proporcionando “sasl” en esta opción.

El valor predeterminado es ‘`"none"`’.

`lista-opcional access-drivers` [parámetro de `libvirt-configuration`]
Esquema de la API de control de acceso.

De manera predeterminada una usuaria identificada puede acceder a todas las API. Los controladores de acceso pueden incluir restricciones de acceso sobre ello.

El valor predeterminado es ‘`'()`’.

`string key-file` [parámetro de `libvirt-configuration`]
Ruta del archivo con la clave del servidor. Si se proporciona una cadena vacía, no se carga ninguna clave privada.

El valor predeterminado es ‘`""`’.

`string cert-file` [parámetro de `libvirt-configuration`]
Ruta del archivo con la clave del servidor. Si se proporciona una cadena vacía, no se carga ningún certificado.

El valor predeterminado es ‘`""`’.

`string ca-file` [parámetro de `libvirt-configuration`]
Ruta del archivo con la clave del servidor. Si se proporciona una cadena vacía, no se carga ningún certificado de CA.

El valor predeterminado es ‘`""`’.

`string crl-file` [parámetro de `libvirt-configuration`]
Ruta de la lista de revocaciones de certificado. Si se proporciona una cadena vacía, no se carga ninguna lista.

El valor predeterminado es '`""`'.

`boolean tls-no-sanity-cert` [parámetro de `libvirt-configuration`]
Desactiva la verificación de los propios certificados del servidor.

Cuando libvirtd arranca, realiza algunas comprobaciones básicas sobre sus propios certificados.

El valor predeterminado es '`#f`'

`boolean tls-no-verify-cert` [parámetro de `libvirt-configuration`]
Desactiva la verificación de certificados de clientes.

La verificación de certificados de cliente es el mecanismo primario de identificación. Se rechazará cualquier cliente que no presente un certificado firmado por la autoridad de certificación (CA).

El valor predeterminado es '`#f`'

`lista-opcional tls-allowed-dn-list` [parámetro de `libvirt-configuration`]
Lista de nombres distinguidos (DN) x509 permitidos.

El valor predeterminado es '`'()`'.

`lista-opcional sasl-allowed-usernames` [parámetro de `libvirt-configuration`]
Lista de nombres de usuaria SASL permitidos. El formato para el nombre de la usuaria depende del mecanismo de identificación SASL.

El valor predeterminado es '`'()`'.

`string tls-priority` [parámetro de `libvirt-configuration`]
Cambia el valor de la cadena de prioridad de TLS predeterminada en tiempo de compilación. El valor predeterminado habitualmente es '`"NORMAL"`' a menos que se cambiase en tiempo de compilación. Proporcione este valor únicamente si desea que libvirt se desvíe de la configuración global predeterminada.

El valor predeterminado es '`"NORMAL"`'.

`integer max-clients` [parámetro de `libvirt-configuration`]
Número máximo de conexiones concurrentes de clientes permitidas en todos los sockets combinados.

El valor predeterminado es '`5000`'.

`integer max-queued-clients` [parámetro de `libvirt-configuration`]
Longitud máxima de la cola de conexiones a la espera de ser aceptadas por el daemon. Fíjese que algunos protocolos que implementan la retransmisión pueden obedecer esto de manera que un intento posterior de conexión tenga éxito.

El valor predeterminado es '`1000`'.

`integer max-anonymous-clients` [parámetro de `libvirt-configuration`]
Longitud máxima de la cola de clientes aceptados pero no identificados todavía. Proporcione el valor cero para desactivar esta característica.

El valor predeterminado es '`20`'.

`integer min-workers` [parámetro de `libvirt-configuration`]
Número de procesos de trabajo que se lanzarán inicialmente.

El valor predeterminado es '`5`'.

`integer max-workers` [parámetro de `libvirt-configuration`]
Número máximo de hilos de trabajo.

Si el número de clientes excede `min-workers`, se lanzan más hilos, hasta el límite `max-workers`. Habitualmente se desea que `max-workers` sea igual al número máximo de clientes permitido.

El valor predeterminado es '`20`'.

`integer prio-workers` [parámetro de `libvirt-configuration`]
Número de procesos de trabajo prioritarios. Si todos los hilos de trabajo del conjunto previo se encuentran bloqueados, algunas llamadas marcadas como de alta prioridad (notablemente domainDestroy) pueden ejecutarse en este conjunto de hilos.

El valor predeterminado es '`5`'.

`integer max-requests` [parámetro de `libvirt-configuration`]
Límite global total de llamadas RPC concurrentes.

El valor predeterminado es '`20`'.

`integer max-client-requests` [parámetro de `libvirt-configuration`]
Límite de peticiones concurrentes desde una única conexión de cliente. Para evitar que un cliente monopolice el servidor esto debe ser una pequeña fracción de los parámetros globales "max_requests" y "max_workers".

El valor predeterminado es '`5`'.

`integer admin-min-workers` [parámetro de `libvirt-configuration`]
Igual que `min-workers` pero para la interfaz de administración.

El valor predeterminado es '`1`'.

`integer admin-max-workers` [parámetro de `libvirt-configuration`]
Igual que `max-workers` pero para la interfaz de administración.

El valor predeterminado es '`5`'.

`integer admin-max-clients` [parámetro de `libvirt-configuration`]
Igual que `max-clients` pero para la interfaz de administración.

El valor predeterminado es '`5`'.

`integer admin-max-queued-clients` [parámetro de `libvirt-configuration`]
Igual que `max-queued-clients` pero para la interfaz de administración.

El valor predeterminado es '`5`'.

`integer admin-max-client-requests` [parámetro de `libvirt-configuration`]
Igual que `max-client-requests` pero para la interfaz de administración.

El valor predeterminado es '`5`'.

`integer log-level` [parámetro de `libvirt-configuration`]
Nivel de registro. 4 errores, 3 avisos, 2 información, 1 depuración.

El valor predeterminado es '`3`'.

`string log-filters` [parámetro de `libvirt-configuration`]
Filtros del registro.

Un filtro permite la selección de un nivel de registro diferente para una categoría dada de registros. El formato del filtro es uno de los siguientes:

- x:nombre
- x:+nombre

donde `nombre` es una cadena contra la que se compara la categoría proporcionada en la llamada `VIR_LOG_INIT()` al principio de cada archivo de fuentes de libvirt, por ejemplo '`"remote"`', '`"qemu"`' o '`"util.json"`' (el nombre en el filtro puede ser una subcadena del nombre completo de la categoría, para aceptar múltiples categorías con nombres similares), el prefijo opcional '`"+"`' indica a libvirt que registre la pila de llamadas en cada mensaje con el nombre correspondiente, y `x` es el nivel mínimo de los mensajes que deben registrarse:

- 1: DEBUG (depuración)
- 2: INFO (información)
- 3: WARNING (aviso)
- 4: ERROR

Se pueden definir en una única sentencia múltiples filtros, únicamente hace falta separarlos por espacios.

El valor predeterminado es '`"3:remote 4:event"`'.

`string log-outputs` [parámetro de `libvirt-configuration`]
Salidas de log.

Una salida es uno de esos lugares para almacenar información de logging. El formato para una salida puede ser:

`x:stderr` la salida va a stderr

`x:syslog:nombre`
: usa syslog para la salida y usa el nombre proporcionado como identificador

`x:file:ruta_archivo`
: encamina la salida a un archivo, con la ruta proporcionada

`x:journald`
: usa el sistema de logging journald

En todos los casos el prefijo x es el nivel mínimo, que actúa como filtro

- 1: DEBUG (depuración)

- 2: INFO (información)
- 3: WARNING (aviso)
- 4: ERROR

Se pueden definir salidas múltiples, únicamente deben separarse por espacios.

El valor predeterminado es '`"3:stderr"`'.

`integer audit-level` [parámetro de `libvirt-configuration`]

Permite la alteración del uso del sistema de auditoría.

- 0: desactiva la auditoría
- 1: activa la auditoría, únicamente si está activado en la máquina
- 2: activa la auditoría, y sale si está desactivada en la máquina.

El valor predeterminado es '`1`'.

`boolean audit-logging` [parámetro de `libvirt-configuration`]

Envía los mensajes de auditoría a través de la infraestructura de registro de libvirt.

El valor predeterminado es '`#f`'

`string-opcional host-uuid` [parámetro de `libvirt-configuration`]

Host UUID. UUID must not have all digits be the same.

El valor predeterminado es '`""`'.

`string host-uuid-source` [parámetro de `libvirt-configuration`]

Fuente de lectura del UUID de la máquina anfitriona.

- `smbios`: obtiene el UUID de `dmidecode -s system-uuid`
- `machine-id`: obtiene el UUID de `/etc/machine-id`

Si `dmidecode` no proporciona un UUID válido, se generará un UUID temporal.

El valor predeterminado es '`"smbios"`'.

`integer keepalive-interval` [parámetro de `libvirt-configuration`]

Un mensaje "keepalive" se envía al cliente tras `keepalive_interval` segundos de inactividad para comprobar si el cliente todavía responde. Si se proporciona el valor -1, libvirtd nunca enviará peticiones "keepalive"; no obstante los clientes todavía pueden mandarlas y el daemon enviará las respuestas.

El valor predeterminado es '`5`'.

`integer keepalive-count` [parámetro de `libvirt-configuration`]

Número máximo de mensajes "keepalive" que se permite enviar a un cliente sin obtener respuesta antes de considerar que se ha roto la conexión.

En otras palabras, la conexión se cierra automáticamente tras `keepalive_interval * (keepalive_count + 1)` segundos tras la última recepción de un mensaje desde el cliente. Cuando `keepalive_count` tiene valor 0, las conexiones se cerrarán automáticamente tras `keepalive-interval` segundos de inactividad sin mandar ningún mensaje "keepalive".

El valor predeterminado es '`5`'.

`integer admin-keepalive-interval` [parámetro de `libvirt-configuration`]
Igual que la opción anterior pero para la interfaz de administración.

El valor predeterminado es '`5`'.

`integer admin-keepalive-count` [parámetro de `libvirt-configuration`]
Igual que la opción anterior pero para la interfaz de administración.

El valor predeterminado es '`5`'.

`integer ovs-timeout` [parámetro de `libvirt-configuration`]
Plazo máximo para las llamadas a Open vSwitch.

La utilidad `ovs-vsctl` se usa para la configuración y su opción de plazo máximo (timeout) tiene un valor de 5 segundos de manera predeterminada para evitar que esperas potencialmente infinitas bloqueen libvirt.

El valor predeterminado es '`5`'.

## Daemon Virtlog

El servicio virtlogd es un daemon del que se compone el lado servidor de libvirt cuya finalidad es la gestión del registro de las consolas de las máquinas virtuales.

This daemon is not used directly by libvirt client applications, rather it is called on their behalf by `libvirtd`. By maintaining the logs in a standalone daemon, the main `libvirtd` daemon can be restarted without risk of losing logs. The `virtlogd` daemon has the ability to re-exec() itself upon receiving `SIGUSR1`, to allow live upgrades without downtime.

`virtlog-service-type` [Variable]
Este es el tipo del daemon virtlog. Su valor debe ser un objeto `virtlog-configuration`.

```
(service virtlog-service-type
         (virtlog-configuration
          (max-clients 1000)))
```

`package libvirt` [`libvirt` parameter]
Paquete libvirt.

`integer log-level` [parámetro de `virtlog-configuration`]
Nivel de registro. 4 errores, 3 avisos, 2 información, 1 depuración.

El valor predeterminado es '`3`'.

`string log-filters` [parámetro de `virtlog-configuration`]
Filtros del registro.

Un filtro permite la selección de un nivel de registro diferente para una categoría dada de registros. El formato del filtro es uno de los siguientes:

- x:nombre
- x:+nombre

donde `nombre` es una cadena contra la que se compara la categoría proporcionada en la llamada `VIR_LOG_INIT()` al principio de cada archivo de fuentes de libvirt, por ejemplo `"remote"`, `"qemu"` o `"util.json"` (el nombre en el filtro puede ser una

subcadena del nombre completo de la categoría, para aceptar múltiples categorías con nombres similares), el prefijo opcional `"+"` indica a libvirt que registre la pila de llamadas en cada mensaje con el nombre correspondiente, y `x` es el nivel mínimo de los mensajes que deben registrarse:

- 1: DEBUG (depuración)
- 2: INFO (información)
- 3: WARNING (aviso)
- 4: ERROR

Se pueden definir en una única sentencia múltiples filtros, únicamente hace falta separarlos por espacios.

El valor predeterminado es '`"3:remote 4:event"`'.

`string log-outputs` [parámetro de `virtlog-configuration`]

Salidas de log.

Una salida es uno de esos lugares para almacenar información de logging. El formato para una salida puede ser:

`x:stderr` la salida va a stderr

`x:syslog:nombre`
: usa syslog para la salida y usa el nombre proporcionado como identificador

`x:file:ruta_archivo`
: encamina la salida a un archivo, con la ruta proporcionada

`x:journald`
: usa el sistema de logging journald

En todos los casos el prefijo x es el nivel mínimo, que actúa como filtro

- 1: DEBUG (depuración)
- 2: INFO (información)
- 3: WARNING (aviso)
- 4: ERROR

Se pueden definir salidas múltiples, únicamente deben separarse por espacios.

El valor predeterminado es '`"3:stderr"`'.

`integer max-clients` [parámetro de `virtlog-configuration`]

Número máximo de conexiones concurrentes de clientes permitidas en todos los sockets combinados.

El valor predeterminado es '`1024`'.

`integer max-size` [parámetro de `virtlog-configuration`]

Tamaño máximo del archivo antes de pasar al siguiente.

El valor predeterminado es '`2MB`'.

`integer max-backups` [parámetro de `virtlog-configuration`]

Número máximo de archivos de backup que se deben mantener.

El valor predeterminado es '`3`'.

## Emulación transparente con QEMU

`qemu-binfmt-service-type` proporciona la capacidad de emular transparentemente programas binarios construidos para arquitecturas diferentes—por ejemplo, le permite ejecutar de manera transparente un programa de ARMv7 en una máquina x86_64. Esto se consigue mediante la combinación del emulador QEMU (`https://www.qemu.org`) y la característica `binfmt_misc` del núcleo Linux. Esta característica únicamente le permite emular GNU/Linux en una arquitectura diferente, pero más adelante puede ver como implementar también GNU/Hurd.

`qemu-binfmt-service-type` [Variable]

Este es el tipo del servicio de emulación transparente QEMU/binfmt. Su valor debe ser un objeto `qemu-binfmt-configuration`, que especifica el paquete QEMU usado así como las arquitecturas que se desean emular:

```
(service qemu-binfmt-service-type
         (qemu-binfmt-configuration
           (platforms (lookup-qemu-platforms "arm" "aarch64"))))
```

En este ejemplo se activa la emulación transparente para las plataformas ARM y aarch64. La ejecución de `herd stop qemu-binfmt` la desactiva, y la ejecución de `herd start qemu-binfmt` la vuelve a activar (see Section "Invoking herd" in *The GNU Shepherd Manual*).

`qemu-binfmt-configuration` [Tipo de datos]

Esta es la configuración para el servicio `qemu-binfmt`.

`platforms` (predeterminadas: `'()`)

Lista de plataformas de QEMU emuladas. Cada elemento debe ser un *objeto de plataforma* como los devueltos por `lookup-qemu-platforms` (véase a continuación).

Por ejemplo, supongamos que está en una máquina x86_64 y tiene este servicio:

```
(service qemu-binfmt-service-type
         (qemu-binfmt-configuration
           (platforms (lookup-qemu-platforms "arm"))))
```

Puede ejecutar:

```
guix build -s armhf-linux inkscape
```

and it will build Inkscape for ARMv7 *as if it were a native build*, transparently using QEMU to emulate the ARMv7 CPU. Pretty handy if you'd like to test a package build for an architecture you don't have access to!

`qemu` (predeterminado: `qemu`)

El paquete QEMU usado.

`lookup-qemu-platforms` *plataformas...* [Procedimiento]

Devuelve la lista de objetos de plataforma de QEMU que corresponden a *plataformas...*. *plataformas* debe ser una lista de cadenas que correspondan con nombres de plataforma, como `"arm"`, `"sparc"`, `"mips64el"`, etcétera.

`qemu-platform?` *obj* [Procedimiento]
Devuelve verdadero si *obj* es un objeto de plataforma.

`qemu-platform-name` *plataforma* [Procedimiento]
Devuelve el nombre de *plataforma*—una cadena como `"arm"`.

## QEMU Guest Agent

The QEMU guest agent provides control over the emulated system to the host. The `qemu-guest-agent` service runs the agent on Guix guests. To control the agent from the host, open a socket by invoking QEMU with the following arguments:

```
qemu-system-x86_64 \
-chardev socket,path=/tmp/qga.sock,server=on,wait=off,id=qga0 \
-device virtio-serial \
-device virtserialport,chardev=qga0,name=org.qemu.guest_agent.0 \
...
```

This creates a socket at `/tmp/qga.sock` on the host. Once the guest agent is running, you can issue commands with `socat`:

```
$ guix shell socat -- socat unix-connect:/tmp/qga.sock stdio
{"execute": "guest-get-host-name"}
{"return": {"host-name": "guix"}}
```

See QEMU guest agent documentation (`https://wiki.qemu.org/Features/GuestAgent`) for more options and commands.

`qemu-guest-agent-service-type` [Variable]
Service type for the QEMU guest agent service.

`qemu-guest-agent-configuration` [Data Type]
Configuration for the `qemu-guest-agent` service.

`qemu` (predeterminado: `qemu-minimal`)
El paquete QEMU usado.

`device` (default: `""`)
File name of the device or socket the agent uses to communicate with the host. If empty, QEMU uses a default file name.

## Virtual Build Machines

*Virtual build machines* or "build VMs" let you offload builds to a fully controlled environment. "How can it be more controlled than regular builds? And why would it be useful?", you ask. Good questions.

Builds spawned by `guix-daemon` indeed run in a controlled environment; specifically the daemon spawns build processes in separate namespaces and in a chroot, such as that build processes only see their declared dependencies and a well-defined subset of the file system tree (see Section 2.2.1 [Configuración del entorno de construcción], page 6, for details). A few aspects of the environments are not controlled though: the operating system kernel, the CPU model, and the date. Most of the time, these aspects have no impact on the build process: the level of isolation `guix-daemon` provides is "good enough".

However, there are occasionally cases where those aspects *do* influence the build process. A typical example is *time traps*: build processes that stop working after a certain date[15]. Another one is software that optimizes for the CPU microarchitecture it is built on or, worse, bugs that manifest only on specific CPUs.

To address that, `virtual-build-machine-service-type` lets you add a virtual build machine on your system, as in this example:

```
(use-modules (gnu services virtualization))

(operating-system
  ;; ...
  (services (append (list (service virtual-build-machine-service-type))
                    %base-services)))
```

By default, you have to explicitly start the build machine when you need it, at which point builds may be offloaded to it (see Section 2.2.2 [Configuración de delegación del daemon], page 7):

```
herd start build-vm
```

With the default setting shown above, the build VM runs with its clock set to a date several years in the past, and on a CPU model that corresponds to that date—a model possibly older than that of your machine. This lets you rebuild today software from the past that would otherwise fail to build due to a time trap or other issues in its build process. You can view the VM's config like this:

```
herd configuration build-vm
```

You can configure the build VM, as in this example:

```
(service virtual-build-machine-service-type
         (virtual-build-machine
          (cpu "Westmere")
          (cpu-count 8)
          (memory-size (* 1 1024))
          (auto-start? #t)))
```

The available options are shown below.

`virtual-build-machine-service-type` [Variable]

This is the service type to run *virtual build machines*. Virtual build machines are configured so that builds are offloaded to them when they are running.

`virtual-build-machine` [Data Type]

This is the data type specifying the configuration of a build machine. It contains the fields below:

`name` (default: `'build-vm`)

The name of this build VM. It is used to construct the name of its Shepherd service.

[15] The most widespread example of time traps is test suites that involve checking the expiration date of a certificate. Such tests exists in TLS implementations such as OpenSSL and GnuTLS, but also in high-level software such as Python.

`image` The image of the virtual machine (see Chapter 16 [System Images], page 710). This notably specifies the virtual disk size and the operating system running into it (see Section 11.3 [Referencia de operating-system], page 253). The default value is a minimal operating system image.

`qemu` (predeterminado: `qemu-minimal`)
The QEMU package to run the image.

`cpu` The CPU model being emulated as a string denoting a model known to QEMU.

The default value is a model that matches `date` (see below). To see what CPU models are available, run, for example:

```
qemu-system-x86_64 -cpu help
```

`cpu-count` (default: `4`)
The number of CPUs emulated by the virtual machine.

`memory-size` (default: `2048`)
Size in mebibytes (MiB) of the virtual machine's main memory (RAM).

`date` (default: a few years ago)
Date inside the virtual machine when it starts; this must be a SRFI-19 date object (see Section "SRFI-19 Date" in *GNU Guile Reference Manual*).

`port-forwardings` (default: 11022 and 11004)
TCP ports of the virtual machine forwarded to the host. By default, the SSH and secrets ports are forwarded into the host.

`systems` (default: `(list (%current-system))`)
List of system types supported by the build VM—e.g., `"x86_64-linux"`.

`auto-start?` (default: `#f`)
Whether to start the virtual machine when the system boots.

In the next section, you'll find a variant on this theme: GNU/Hurd virtual machines!

## Hurd en una máquina virtual

El servicio `hurd-vm` implementa la ejecución de GNU/Hurd en una máquina virtual (VM), llamado *childhurd*. Este servicio está destinado para su uson GNU/Linux y la configuración de sistema operativo de GNU/Hurd se compila de manera cruzada. La máquina virtual es un servicio de Shepherd al que se puede hacer referencia a través de los nombres `hurd-vm` y `childhurd` y se puede controlar mediante órdenes como las siguientes:

```
herd start hurd-vm
herd stop childhurd
```

Cuando el servicio se encuentra en ejecución, puede ver su consola a través de una conexión con un cliente VNC, por ejemplo con:

```
guix shell tigervnc-client -- vncviewer localhost:5900
```

The default configuration (see `hurd-vm-configuration` below) spawns a secure shell (SSH) server in your GNU/Hurd system, which QEMU (the virtual machine emulator)

redirects to port 10022 on the host. By default, the service enables *offloading* such that the host `guix-daemon` automatically offloads GNU/Hurd builds to the childhurd (see Section 2.2.2 [Configuración de delegación del daemon], page 7). This is what happens when running a command like the following one, where `i586-gnu` is the system type of 32-bit GNU/Hurd:

```
guix build emacs-minimal -s i586-gnu
```

childhurd es volátil y carece de estado: comienza con un sistema de archivos raíz creado de cero cada vez que lo reinicia. No obstante de manera predeterminada todos los archivos en la ruta `/etc/childhurd` de la máquina anfitriona se copian al sistema de archivos raíz de childhurd cuando arranca. Esto permite que configure "secretos" dentro de la máquina virtual: claves de SSH de la máquina, claves de sustituciones autorizadas, etcétera—véase la explicación de `secret-root` a continuación.

You will probably find it useful to create an account for you in the GNU/Hurd virtual machine and to authorize logins with your SSH key. To do that, you can define the GNU/Hurd system in the usual way (see Section 11.2 [Uso de la configuración del sistema], page 244), and then pass that operating system as the `os` field of `hurd-vm-configuration`, as in this example:

```
(define childhurd-os
  ;; Definition of my GNU/Hurd system, derived from the default one.
  (operating-system
    (inherit %hurd-vm-operating-system)

    ;; Add a user account.
    (users (cons (user-account
                  (name "charlie")
                  (comment "This is me!")
                  (group "users")
                  (supplementary-groups '("wheel"))) ;for 'sudo'
                 %base-user-accounts))

    (services
     ;; Modify the SSH configuration to allow login as "root"
     ;; and as "charlie" using public key authentication.
     (modify-services (operating-system-user-services
                       %hurd-vm-operating-system)
       (openssh-service-type
        config => (openssh-configuration
                   (inherit config)
                   (authorized-keys
                    `(("root"
                       ,(local-file
                         "/home/charlie/.ssh/id_rsa.pub"))
                      ("charlie"
                       ,(local-file
                         "/home/charlie/.ssh/id_rsa.pub")))))))))
```

```
(operating-system
  ;; ...
  (services
    ;; Add the 'hurd-vm' service, configured to use the
    ;; operating system configuration above.
    (append (list (service hurd-vm-service-type
                           (hurd-vm-configuration
                             (os %childhurd-os))))
            %base-services)))
```

That's it! The remainder of this section provides the reference of the service configuration.

`hurd-vm-service-type` [Variable]

El tipo del servicio que ejecuta Hurd en una máquina virtual. Su valor debe ser un objeto `hurd-vm-configuration`, que especifica el sistema operativo (see Section 11.3 [Referencia de operating-system], page 253) y el tamaño del disco para la máquina virtual de Hurd, el paquete de QEMU usado así como las opciones de ejecución.

Por ejemplo:

```
(service hurd-vm-service-type
         (hurd-vm-configuration
          (disk-size (* 5000 (expt 2 20))) ;5G
          (memory-size 1024)))             ;1024MiB
```

crearía una imagen de disco suficientemente grande para construir GNU Hello, con algo de memoria adicional.

`hurd-vm-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `hurd-vm-service-type`.

`os` (predeterminado: *%hurd-vm-operating-system*)
: El sistema operativo instanciado. El valor predeterminado es un sistema mínimo con un daemon del intérprete de comandos seguro OpenSSH configurado de forma permisiva asociado al puerto 2222 (see Section 11.10.5 [Servicios de red], page 314).

`qemu` (predeterminado: `qemu-minimal`)
: El paquete QEMU usado.

`image` (predeterminado: *hurd-vm-disk-image*)
: The image object representing the disk image of this virtual machine (see Chapter 16 [System Images], page 710).

`disk-size` (predeterminado: `'guess`)
: El tamaño de la imagen de disco.

`memory-size` (predeterminado: `512`)
: El tamaño de la memoria de la máquina virtual en mebibytes.

`options` (predeterminadas: `'("--snapshot")`)
: Opciones adicionales para ejecutar QEMU.

`id` (predeterminado: `#f`)
: Si se proporciona un valor, debe ser un entero positivo no nulo usado para parametrizar las instancias Childhurd. Se añade al nombre del servicio, por ejemplo `childhurd1`.

`net-options` (predeterminado: *hurd-vm-net-options*)
: El procedimiento usado para producir la lista de opciones de red para QEMU.

  De manera predeterminada produce

```
'("--device" "rtl8139,netdev=net0"
  "--netdev" (string-append
              "user,id=net0,"
              "hostfwd=tcp:127.0.0.1:secrets-port-:1004,"
              "hostfwd=tcp:127.0.0.1:ssh-port-:2222,"
              "hostfwd=tcp:127.0.0.1:vnc-port-:5900"))
```

  con los siguientes puertos redirigidos:

```
secrets-port: (+ 11004 (* 1000 ID))
ssh-port: (+ 10022 (* 1000 ID))
vnc-port: (+ 15900 (* 1000 ID))
```

`offloading?` (default: `#t`)
: Whether to automatically set up offloading of builds to the childhurd.

  When enabled, this lets you run GNU/Hurd builds on the host and have them transparently offloaded to the VM, for instance when running a command like this:

```
guix build coreutils -s i586-gnu
```

  This option automatically sets up offloading like so:

  1. Authorizing the childhurd's key on the host so that the host accepts build results coming from the childhurd, which can be done like so (see Section 5.11 [Invocación de guix archive], page 66, for more on that).
  2. Creating a user account called `offloading` dedicated to offloading in the childhurd.
  3. Creating an SSH key pair on the host and making it an authorized key of the `offloading` account in the childhurd.
  4. Añadir childhurd a `/etc/guix/machines.scm` (see Section 2.2.2 [Configuración de delegación del daemon], page 7).

`secret-root` (predeterminado: `/etc/childhurd`)
: El directorio raíz que contiene los secretos fuera-de-banda que serán instalados en childhurd cuando se ejecute. Los childhurd son volátiles lo que significa que cada arranque los secretos como las claves de SSH de la máquina y la clave de firma de Guix se regeneran.

  Si el directorio `/etc/childhurd` no existe, el servicio `secret-service` que se ejecuta en childhurd recibirá una lista vacía de secretos.

De manera predeterminada el servicio crea `/etc/childhurd` con los siguientes secretos no volátiles, a no ser que ya existan:

```
/etc/childhurd/etc/guix/acl
/etc/childhurd/etc/guix/signing-key.pub
/etc/childhurd/etc/guix/signing-key.sec
/etc/childhurd/etc/ssh/authorized_keys.d/offloading
/etc/childhurd/etc/ssh/ssh_host_ed25519_key
/etc/childhurd/etc/ssh/ssh_host_ecdsa_key
/etc/childhurd/etc/ssh/ssh_host_ed25519_key.pub
/etc/childhurd/etc/ssh/ssh_host_ecdsa_key.pub
```

Tenga en cuenta que la imagen de la máquina virtual es volátil, es decir, los contenidos se pierden cuando se para. Si desea una imagen que mantenga el estado puede modificar la configuración de `image` y `options` sin la opción `--snapshot` usando algo parecido a esto:

```
(service hurd-vm-service-type
         (hurd-vm-configuration
          (image   (const "/no/almacen/y/perm/escritura/hurd.img"))
          (options '())))
```

## Ganeti

**Nota:** Este servicio se considera experimental. Las opciones de configuración pueden cambiar de manera incompatible con versiones previas, y no todas las características han sido probadas en profundidad. Se recomienda a quienes usen este servicio que compartan su experiencia en `guix-devel@gnu.org`.

Ganeti is a virtual machine management system. It is designed to keep virtual machines running on a cluster of servers even in the event of hardware failures, and to make maintenance and recovery tasks easy. It consists of multiple services which are described later in this section. In addition to the Ganeti service, you will need the OpenSSH service (see Section 11.10.5 [Servicios de red], page 314), and update the `/etc/hosts` file (see Section 11.19.3 [Referencia de servicios], page 652) with the cluster name and address (or use a DNS server).

Todos los nodos que participan en el cluster de Ganeti deben tener la misma configuración de Ganeti y en el archivo `/etc/hosts`. A continuación se encuentra un ejemplo de configuración para un nodo del cluster de Ganeti que implementa varios motores de almacenamiento, e instala los *proveedores de sistema operativo* `debootstrap` y `guix`:

```
(use-package-modules virtualization)
(use-service-modules base ganeti networking ssh)
(operating-system
  ;; ...
  (host-name "node1")

  ;; Install QEMU so we can use KVM-based instances, and LVM, DRBD and Ceph
  ;; in order to use the "plain", "drbd" and "rbd" storage backends.
  (packages (append (map specification->package
                         '("qemu" "lvm2" "drbd-utils" "ceph"
                           ;; Add the debootstrap and guix OS providers.
```

```
                          "ganeti-instance-guix" "ganeti-instance-debootstrap"))
                  %base-packages))
      (services
       (append (list (service static-networking-service-type
                              (list (static-networking
                                     (addresses
                                      (list (network-address
                                             (device "eth0")
                                             (value "192.168.1.201/24"))))
                                     (routes
                                      (list (network-route
                                             (destination "default")
                                             (gateway "192.168.1.254"))))
                                     (name-servers '("192.168.1.252"
                                                     "192.168.1.253")))))

                     ;; Ganeti uses SSH to communicate between nodes.
                     (service openssh-service-type
                              (openssh-configuration
                               (permit-root-login 'prohibit-password)))

                     (simple-service 'ganeti-hosts-entries hosts-service-type
                                     (list
                                       (host "192.168.1.200" "ganeti.example.com")
                                       (host "192.168.1.201" "node1.example.com"
                                             '("node1"))
                                       (host "192.168.1.202" "node2.example.com"
                                             '("node2"))))

                     (service ganeti-service-type
                              (ganeti-configuration
                               ;; Esta lista especifica las rutas del
                               ;; sistema de archivos permitidas para el
                               ;; almacenamiento de imágenes de máquinas
                               ;; viruales.
                               (file-storage-paths '("/srv/ganeti/almacenamiento"))
                               ;; Esta variable configura una única
                               ;; "variación" tanto para Debootstrap como
                               ;; Guix que funciona con KVM.
                               (os %default-ganeti-os))))
               %base-services)))
```

Users are advised to read the Ganeti administrators guide (`https://docs.ganeti.org/docs/ganeti/3.0/html/admin.html`) to learn about the various cluster options and day-to-day operations. There is also a blog post (`https://guix.gnu.org/blog/2020/running-a-ganeti-cluster-on-guix/`) describing how to configure and initialize a small cluster.

`ganeti-service-type` [Variable]

Tipo de servicio que incluye todos los distintos servicios que los nodos Ganeti deben ejecutar.

Su valor es un objeto `ganeti-configuration` que define usado para las operaciones de línea de órdenes, así como la configuración para varios daemon. Las rutas de almacenamiento permitidas y los sistemas operativos disponibles para hospedar también se configuran a través de este tipo de datos.

`ganeti-configuration` [Tipo de datos]

El servicio `ganeti` proporciona las siguientes opciones de configuración:

`ganeti` (predeterminado: `ganeti`)

El paquete `ganeti` usado. Dicho paquete se instala en el perfil del sistema y hace que las órdenes `gnt-cluster`, `gnt-instance`, etcétera, estén disponibles. Tenga en cuenta que el valor especificado aquí no afecta a otros servicios ya que cada uno hace referencia a su paquete `ganeti` específico (véase a continuación).

`noded-configuration` (predeterminado: `(ganeti-noded-configuration)`)
`confd-configuration` (predeterminado: `(ganeti-confd-configuration)`)
`wconfd-configuration` (predeterminado: `(ganeti-wconfd-configuration)`)
`luxid-configuration` (predeterminado: `(ganeti-luxid-configuration)`)
`rapi-configuration` (predeterminado: `(ganeti-rapi-configuration)`)
`kvmd-configuration` (predeterminado: `(ganeti-kvmd-configuration)`)
`mond-configuration` (predeterminado: `(ganeti-mond-configuration)`)
`metad-configuration` (predeterminado: `(ganeti-metad-configuration)`)
`watcher-configuration` (predeterminado: `(ganeti-watcher-configuration)`)
`cleaner-configuration` (predeterminado: `(ganeti-cleaner-configuration)`)

Estas opciones controlan los distintos daemon y trabajos de cron que se distribuyen con Ganeti. Los posibles valores se describen en detalle a continuación. Para modificar el valor de un elemento de la configuración se debe usar el tipo de configuración para dicho servicio:

```
(service ganeti-service-type
         (ganeti-configuration
          (rapi-configuration
           (ganeti-rapi-configuration
            (interface "eth1"))))
          (watcher-configuration
           (ganeti-watcher-configuration
            (rapi-ip "10.0.0.1"))))
```

`file-storage-paths` (predeterminada: `'()`)

Lista de directorios permitidos para el motor de almacenamiento de archivos.

`hooks` (default: `#f`)

When set, this should be a file-like object containing a directory with cluster execution hooks (`https://docs.ganeti.org/docs/ganeti/3.0/html/hooks.html`).

`os` (predeterminado: `%default-ganeti-os`)
Lista de registros `<ganeti-os>`.

En esencia `ganeti-service-type` es una abreviación para declara cada uno de los servicios individualmente:

```
(service ganeti-noded-service-type)
(service ganeti-confd-service-type)
(service ganeti-wconfd-service-type)
(service ganeti-luxid-service-type)
(service ganeti-kvmd-service-type)
(service ganeti-mond-service-type)
(service ganeti-metad-service-type)
(service ganeti-watcher-service-type)
(service ganeti-cleaner-service-type)
```

Además de una extensión del servicio `etc-service-type` que configura el motor de almacenamiento de archivos y las variantes de sistema operativo.

`ganeti-os` [Tipo de datos]
Tipo de datos adecuado para proporcionarse como parámetro `os` de `ganeti-configuration`. Tiene los siguientes parámetros:

`name` Nombre para este proveedor de sistema operativo. Solo se usa para especificar dónde termina la configuración. Proporcionar el valor "debootstrap" indica la creación de `/etc/ganeti/instance-debootstrap`.

`extension` (default: `#f`)
The file extension for variants of this OS type. For example `.conf` or `.scm`. It will be appended to the variant file name if set.

`variants` (predeterminadas: `'()`)
This must be either a list of `ganeti-os-variant` objects for this OS, or a "file-like" object (see Section 8.12 [Expresiones-G], page 167) representing the variants directory.

To use the Guix OS provider with variant definitions residing in a local directory instead of declaring individual variants (see *guix-variants* below), you can do:

```
(ganeti-os
 (name "guix")
 (variants (local-file "ganeti-guix-variants"
                       #:recursive? #true)))
```

Note that you will need to maintain the `variants.list` file (see `ganeti-os-interface(7)` (`https://docs.ganeti.org/docs/ganeti/3.0/man/ganeti-os-interface.html`)) manually in this case.

`ganeti-os-variant` [Tipo de datos]
Tipo de datos que representa una variante de SO Ganeti. Este tipo tiene los siguientes parámetros:

`name` El nombre de esta variación.

`configuration`
: Un archivo de configuración para esta variación.

`%default-debootstrap-hooks` [Variable]
: Esta variable contiene extensiones ("hook") para la configuración de la red y el cargador de arranque GRUB.

`%default-debootstrap-extra-pkgs` [Variable]
: Esta variable contiene una lista de paquetes adecuada para una máquina completamente virtualizada.

`debootstrap-configuration` [Tipo de datos]
: Este tipo de datos crea archivos de configuración adecuados para la orden de generación de sistemas operativos debootstrap.

  `hooks` (predeterminada: `%default-debootstrap-hooks`)
  : Cuando no es `#f` debe ser una expresión-G que especifique el directorio con los guiones que deberán ejecutarse cuando se instale el sistema operativo. También puede ser una lista de pares `(nombre . obj-tipo-archivo)`. Por ejemplo:

    ```
    `((99-hola-mundo . ,(plain-file "#!/bin/sh\necho '¡Hola mundo!'")))
    ```

    Esto crea un directorio con un ejecutable que se llama `99-hola-mundo` y se ejecuta cada vez que se instale esta variación. Si se proporciona `#f`, se usarán los archivos del directorio `/etc/ganeti/instance-debootstrap/hooks`, si existe alguno.

  `proxy` (predeterminado: `#f`)
  : Valor opcional de la pasarela HTTP usada.

  `mirror` (predeterminado: `#f`)
  : El servidor espejo de Debian. Habitualmente es algo como `http://ftp.no.debian.org/debian`. El valor predeterminado cambia dependiendo de la distribución.

  `arch` (predeterminada: `#f`)
  : La arquitectura de dpkg. Proporcione `armhf` para generar con debootstrap una instancia de ARMv7 en una máquina AArch64. El valor predeterminado corresponde a la arquitectura del sistema en uso.

  `suite` (predeterminada: `"stable"`)
  : Si se proporciona debe ser el identificador de una entrega o "suite" de distribución de Debian, como por ejemplo `buster` o `focal`. Si se proporciona `#f`, se usael valor predeterminado del proveedor de sistema operativo.

  `extra-pkgs` (predeterminados: `%default-debootstrap-extra-pkgs`)
  : Lista de paquetes adicionales que dpkg instala junto al sistema mínimo.

  `components` (predeterminado: `#f`)
  : Si se proporciona un valor debe ser una lista de "componentes" de repositorio de Debian. Por ejemplo `'("main" "contrib")`.

`generate-cache?` (predeterminado: `#t`)
: Determina si se almacena en caché de manera automática el archivo de debootstrap que se haya generado.

`clean-cache` (predeterminado: `14`)
: Descarta el contenido de la caché tras este número de días. Use `#f` para no descartar contenido de la caché nunca.

`partition-style` (predeterminado: `'msdos`)
: Tipo de partición creado. Cuando se proporciona un valor debe ser `'msdos`, `'none` o una cadena.

`partition-alignment` (predeterminada: `2048`)
: Alineación de la partición en sectores.

`debootstrap-variant` *nombre configuración* [Procedimiento]
Procedimiento auxiliar que crea un registro `ganeti-os-variant`. Toma dos parámetros: un nombre y un objeto `debootstrap-configuration`.

`debootstrap-os` *variantes...* [Procedimiento]
Procedimiento auxiliar que crea un registro `ganeti-os`. Toma como parámetro una lista de variaciones creada con `debootstrap-variant`.

`guix-variant` *nombre configuración* [Procedimiento]
Procedimiento auxiliar que crea un registro `ganeti-os-variant` para ser usado por el proveedor de sistema operativo Guix. Toma como parámetros un nombre y una expresión-G que devuelve un objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167) que contiene configuración para el sistema Guix.

`guix-os` *variantes...* [Procedimiento]
Procedimiento auxiliar que crea un registro `ganeti-os`. Toma como parámetros una lista de variaciones producida por `guix-variant`.

`%default-debootstrap-variants` [Variable]
Variable de conveniencia para que el proveedor debootstrap funcione "automágicamente" sin que quienes lo usan tengan que declarar variaciones manualmente. Contiene una única variación de debootstrap con la configuración predeterminada:

```
(list (debootstrap-variant
       "default"
       (debootstrap-configuration)))
```

`%default-guix-variants` [Variable]
Variable de conveniencia para que el proveedor de sistema operativo Guix funcione sin ninguna configuración adicional. Crea una máquina virtual que tiene un servidor SSH, consola serie y autoriza las claves de SSH de las máquinas de Ganeti.

```
(list (guix-variant
       "default"
       (file-append ganeti-instance-guix
                    "/share/doc/ganeti-instance-guix/examples/dynamic.scm")))
```

Se pueden implementar proveedores de SO no disponibles en Guix mediante la extensión de los registros `ganeti-os` y `ganeti-os-variant` de manera apropiada. Por ejemplo:

```
(ganeti-os
 (name "personalizado")
 (extension ".conf")
 (variants
  (list (ganeti-os-variant
         (name "cosa")
         (configuration (plain-file "archivo" "Esto va bien"))))))
```

Este ejemplo crearía `/etc/ganeti/instance-personalizado/variants/cosa.conf`, el cual apunta a un archivo en el almacén cuyo contenido es `esto va bien`. También se crearía `/etc/ganeti/instance-personalizado/variants/variants.list` con el contenido `cosa`.

Obviamente es posible que esto no funcione con todos los proveedores de sistema operativo existentes. Si está interfaz le está limitando en su implementación, por favor, contacte con `guix-devel@gnu.org`.

El resto de esta sección documenta los distintos servicios incluidos en `ganeti-service-type`.

`ganeti-noded-service-type` [Variable]

`ganeti-noded` es el daemon responsable de las funciones específicas del nodo dentro del sistema Ganeti. El valor de este servicio debe ser un objeto `ganeti-noded-configuration`.

`ganeti-noded-configuration` [Tipo de datos]

Esta es la configuración para el servicio `ganeti-noded`.

`ganeti` (predeterminado: `ganeti`)
: El paquete `ganeti` usado para este servicio.

`port` (predeterminado: `1811`)
: Puerto TCP en el que escucha el daemon del nodo a la espera de peticiones a través de la red.

`address` (predeterminado: `"0.0.0.0"`)
: La dirección de red a la que el daemon se enlaza. La dirección predeterminada significa que el daemon se enlaza a todas las direcciones disponibles.

`interface` (predeterminado: `#f`)
: En caso de proporcionarse un valor, debe especificar el nombre de la interfaz de red (por ejemplo, `eth0` a la que el daemon se enlaza.

`max-clients` (predeterminado: `20`)
: Establece un límite en el número máximo de conexiones simultaneas de clientes que el daemon manejará. Las conexiones que excedan dicho número se aceptan, pero no se envía respuesta hasta que hayan cerrado suficientes conexiones.

`ssl?` (predeterminado: `#t`)
: Determina si se usa SSL/TLS para cifrar las comunicaciones a través de la red. El cluster proporciona automáticamente el certificado y se puede rotar con `gnt-cluster renew-crypto`.

`ssl-key` (predeterminado: `"/var/lib/ganeti/server.pem"`)
: Puede usarse para proporcionar una clave de cifrado específica para comunicaciones TLS.

`ssl-cert` (predeterminado: `"/var/lib/ganeti/server.pem"`)
: Puede usarse para proporcionar un certificado específico para comunicaciones TLS.

`debug?` (predeterminado: `#f`)
: Cuando es verdadero, el daemon almacena registros adicionales con propósitos de depuración. Tenga en cuenta que esto puede exponer detalles de cifrado en los archivos de registro, por lo que debe usarse con precaución.

`ganeti-confd-service-type` [Variable]

`ganeti-confd` responde a las consultas relacionadas con la configuración del cluster Ganeti. El propósito de este daemon es tener una forma rápida y con alta disponibilidad de consultar los valores de configuración del cluster. Se activa de manera automática en todas las máquinas *candidatas de ser coordinadoras*. El valor de este servicio debe ser un objeto `ganeti-confd-configuration`.

`ganeti-confd-configuration` [Tipo de datos]

Esta es la configuración para el servicio `ganeti-confd`.

`ganeti` (predeterminado: `ganeti`)
: El paquete `ganeti` usado para este servicio.

`port` (predeterminado: `1814`)
: El puerto UDP en el que esperarán peticiones a través de la red.

`address` (predeterminado: `"0.0.0.0"`)
: Dirección de red a la que el daemon se asocia.

`debug?` (predeterminado: `#f`)
: Cuando es verdadero, el daemon almacena registros adicionales con propósitos de depuración.

`ganeti-wconfd-service-type` [Variable]

`ganeti-wconfd` es el daemon que proporciona una autoridad de conocimiento sobre la configuración del cluster y es la única entidad que puede aceptar cambios sobre ella. Todos las trabajos que necesiten modificar la configuración deben hacerlo enviando las peticiones adecuadas a este daemon. Únicamente se ejecuta en el *nodo coordinador* y se desactiva automáticamente en otros nodos.

El valor de este servicio debe ser un objeto `ganeti-wconfd-configuration`.

`ganeti-wconfd-configuration` [Tipo de datos]

Esta es la configuración para el servicio `ganeti-wconfd`.

`ganeti` (predeterminado: `ganeti`)
: El paquete `ganeti` usado para este servicio.

`no-voting?` (predeterminado: `#f`)
: El daemon se negará a arrancar si la mayoría de los nodos del cluster no coinciden en que se está ejecutando en el nodo coordinador. Proporcione `#t` para arrancar incluso cuando no se puede alcanzar dicha mayoría (peligroso, úsese con precaución).

`debug?` (predeterminado: `#f`)
: Cuando es verdadero, el daemon almacena registros adicionales con propósitos de depuración.

`ganeti-luxid-service-type` [Variable]

`ganeti-luxid` es un daemon que responde a las peticiones relacionadas con la configuración del estado actual del cluster Ganeti. De manera adicional, es el daemon que tiene el control sobre la cola de trabajos de Ganeti. Los trabajos se pueden emitir a través de este daemon y éste los planifica y arranca.

Recibe un objeto `ganeti-luxid-configuration`.

`ganeti-luxid-configuration` [Tipo de datos]

This is the configuration for the `ganeti-luxid` service.

`ganeti` (predeterminado: `ganeti`)
: El paquete `ganeti` usado para este servicio.

`no-voting?` (predeterminado: `#f`)
: El daemon se negará a arrancar si no puede verificar que la mayoría de los nodos del cluster creen que éste se está ejecutando en el nodo coordinador. Proporcione `#t` para ignorar dichas comprobaciones y arrancar en cualquier caso (puede ser peligroso).

`debug?` (predeterminado: `#f`)
: Cuando es verdadero, el daemon almacena registros adicionales con propósitos de depuración.

`ganeti-rapi-service-type` [Variable]

`ganeti-rapi` proporciona una API remota para cluster de Ganeti. Se ejecuta en el nodo coordinador y se puede usar para realizar programática acciones en el cluster a través de un protocolo de llamada de procedimientos remotos (RPC) basado en JSON.

Se permiten la mayoría de las operaciones de consulta sin identificación (a no ser que se especifique *require-authentication?*), mientras que las operaciones de escritura necesitan autorización explícita a través del archivo `/var/lib/ganeti/rapi/users`. Véase la documentación de la API remota de Ganeti (`https://docs.ganeti.org/docs/ganeti/3.0/html/rapi.html`) para obtener más información.

El valor de este servicio debe ser un objeto `ganeti-rapi-configuration`.

`ganeti-rapi-configuration` [Tipo de datos]

Configuración para el servicio `ganeti-rapi`.

`ganeti` (predeterminado: `ganeti`)
: El paquete `ganeti` usado para este servicio.

`require-authentication?` (predeterminado: `#f`)
: Determina si se requiere identificación incluso para operaciones únicamente de lectura.

`port` (predeterminado: `5080`)
: El puerto TCP en el que esperarán peticiones de la API.

`address` (predeterminado: `"0.0.0.0"`)
: La dirección de red a la que se asocia el servicio. De manera predeterminada el servicio se asocia a todas las direcciones configuradas.

`interface` (predeterminado: `#f`)
: Si se proporciona un valor, debe especificar el nombre de la interfaz de red, como por ejemplo `eth0`, a la que el daemon se asocia.

`max-clients` (predeterminado: `20`)
: Número máximo de conexiones simultaneas de clientes que se manejan. Las conexiones que excedan dicho número se aceptan, pero no se envía respuesta hasta que hayan cerrado suficientes conexiones.

`ssl?` (predeterminado: `#t`)
: Determina si se usa cifrado SSL/TLS en el puerto de la API remota.

`ssl-key` (predeterminado: `"/var/lib/ganeti/server.pem"`)
: Puede usarse para proporcionar una clave de cifrado específica para comunicaciones TLS.

`ssl-cert` (predeterminado: `"/var/lib/ganeti/server.pem"`)
: Puede usarse para proporcionar un certificado específico para comunicaciones TLS.

`debug?` (predeterminado: `#f`)
: Cuando es verdadero, el daemon almacena registros adicionales con propósitos de depuración. Tenga en cuenta que esto puede exponer detalles de cifrado en los archivos de registro, por lo que debe usarse con precaución.

`ganeti-kvmd-service-type` [Variable]

`ganeti-kvmd` es responsable de determinar si una instancia KVM determinada ha sido apagada por una usuaria o una administradora. Normalmente Ganeti reiniciará una instancia que no se haya parado a través de Ganeti. Si la opción del cluster `user_shutdown` es verdadera, este daemon monitoriza el puerto `QMP` que proporciona QEMU y escucha eventos de apagado en él, y marca la instancia como *USER_down* en vez de *ERROR_down* cuando el daemon la apaga de manera adecuada.

Recibe un objeto `ganeti-kvmd-configuration`.

`ganeti-kvmd-configuration` [Tipo de datos]

`ganeti` (predeterminado: `ganeti`)
El paquete `ganeti` usado para este servicio.

`debug?` (predeterminado: `#f`)
Cuando es verdadero, el daemon almacena registros adicionales con propósitos de depuración.

`ganeti-mond-service-type` [Variable]
`ganeti-mond` es un daemon opcional que proporciona la funcionalidad de monitorización de Ganeti. Es responsable de la ejecución de los recolectores de datos y la publicación de la información obtenida a través de una interfaz HTTP.

Recibe un objeto `ganeti-mond-configuration`.

`ganeti-mond-configuration` [Tipo de datos]

`ganeti` (predeterminado: `ganeti`)
El paquete `ganeti` usado para este servicio.

`port` (predeterminado: `1815`)
Puerto en el que el daemon espera conexiones.

`address` (predeterminado: `"0.0.0.0"`)
La dirección de red a la que se asocia el daemon. De manera predeterminada se asocia a todas las interfaces disponibles.

`debug?` (predeterminado: `#f`)
Cuando es verdadero, el daemon almacena registros adicionales con propósitos de depuración.

`ganeti-metad-service-type` [Variable]
`ganeti-metad` es un daemon opcional que se puede usar para proporcionar información del cluster a instancias o guiones de instalación de sistema operativo.

Recibe un objeto `ganeti-metad-configuration`.

`ganeti-metad-configuration` [Tipo de datos]

`ganeti` (predeterminado: `ganeti`)
El paquete `ganeti` usado para este servicio.

`port` (predeterminado: `80`)
Puerto en el que el daemon espera conexiones.

`address` (predeterminada: `#f`)
Si se proporciona un valor el daemon se asociará únicamente a esta dirección. Si no se proporciona valor el comportamiento depende de la configuración del cluster.

`debug?` (predeterminado: `#f`)
Cuando es verdadero, el daemon almacena registros adicionales con propósitos de depuración.

`ganeti-watcher-service-type` [Variable]

`ganeti-watcher` es un guión diseñado para su ejecución periódica en la que comprueba el estado la salud del cluster. Reinicia automáticamente las instancias que se han parado sin el consentimiento de Ganeti, y repara los enlaces DRBD en caso de que un nodo se haya reiniciado. También archiva los trabajos antiguos del cluster y reinicia los daemon de Ganeti si no se están ejecutando. Si se ha proporcionado valor al parámetro del cluster `ensure_node_health`, este proceso también apagará las instancias y dispositivos DRBD si el nodo que las ejecute se ha declarado fuera de línea por alguna de las máquinas candidatas de coordinación conocidas.

Se puede pausar en todos los nodos con `gnt-cluster watcher pause`.

Este servicio toma como valor un objeto `ganeti-watcher-configuration`.

`ganeti-watcher-configuration` [Tipo de datos]

`ganeti` (predeterminado: `ganeti`)
: El paquete `ganeti` usado para este servicio.

`schedule` (predeterminado: `'(next-second-from (next-minute (range 0 60 5)))`)
: Cada cuanto se ejecuta el guión. El valor predeterminado es cada 5 minutos.

`rapi-ip` (predeterminada: `#f`)
: Esta opción se debe especificar únicamente si el daemon de API remota se ha configurado para usar una interfaz o dirección de red concreta. De manera predeterminada se usa la dirección del cluster.

`job-age` (predeterminados: `(* 6 3600)`)
: Archiva los trabajos del cluster cuya antigüedad sea mayor que el el número de segundos proporcionado. El valor predeterminado son 6 horas. Esto mantiene la lista proporcionada `gnt-job list` dentro de límites gestionables.

`verify-disks?` (predeterminado: `#t`)
: Si es `#f`, el proceso de vigilancia ("watcher") no intentará reparar los enlaces de DRBD rotos de manera automática. Esto significa que quienes administren el sistema deberán usar `gnt-cluster verify-disks` manualmente para realizar dicha tarea.

`debug?` (predeterminado: `#f`)
: Cuando es `#t`, el guión registra información adicional para facilitar la depuración.

`ganeti-cleaner-service-type` [Variable]

`ganeti-cleaner` es un guión diseñado para su ejecución periódica en la que elimina archivos antiguos del cluster. Este tipo de servicio controla dos *trabajos de cron*: uno para el nodo coordinador que elimina de manera permanente trabajos antiguos del cluster, y otro para todos los nodos que elimina certificados X509 y claves que hayan expirado así como información desactualizada de `ganeti-watcher`. Como todos los servicios de Ganeti, se puede incluir también en los nodos que no son coordinadores y se desactivará por si mismo si es necesario.

Recibe un objeto `ganeti-cleaner-configuration`.

`ganeti-cleaner-configuration` [Tipo de datos]

`ganeti` (predeterminado: `ganeti`)
El paquete `ganeti` usado para la orden `gnt-cleaner`.

`master-schedule` (predeterminado: `"45 1 * * *"`)
Cada cuanto se ejecuta el trabajo principal de limpieza. El valor predeterminado representa su ejecución una vez al día, a las 01:45:00.

`node-schedule` (predeterminada: `"45 2 * * *"`)
La frecuencia del trabajo de limpieza del nodo. El valor predeterminado es una vez al día, a las 02:45:00.

### 11.10.31 Servicios de control de versiones

The `(gnu services version-control)` module provides a service to allow remote access to local Git repositories. There are three options: the `git-daemon-service-type`, which provides access to repositories via the `git://` unsecured TCP-based protocol, extending the `nginx` web server to proxy some requests to `git-http-backend`, or providing a web interface with `cgit-service-type`.

`git-daemon-service-type` [Variable]
Type for a service that runs `git daemon`, a simple TCP server to expose repositories over the Git protocol for anonymous access.

The value for this service type is a `<git-daemon-configuration>` record, by default it allows read-only access to exported[16] repositories under `/srv/git`.

`git-daemon-configuration` [Tipo de datos]
Data type representing the configuration for `git-daemon-service-type`.

`package` (predeterminado: `git`)
El objeto paquete del sistema distribuido de control de versiones Git.

`export-all?` (predeterminado: `#f`)
Determina si se permite el acceso a todos los repositorios Git, incluso si no tienen el archivo `git-daemon-export-ok`.

`base-path` (predeterminado: `/srv/git`)
Determina si se traducirán todas las rutas de las peticiones como relativas a la ruta proporcionada. Si se encuentra en ejecución el daemon de git con `(base-path "/srv/git"` en example.com, al realizar la solicitud de '`git://example.com/hello.git`', el daemon de git interpretará la ruta como `/srv/git/hello.git`.

`user-path` (predeterminado: `#f`)
Determina si se permite el uso de la notación `~user` en las peticiones. Si se especifica una cadena vacía, una peticione de '`git://máquina/~alicia/algo`' se tomará como una petición de acceso al repositorio `algo` en el directorio de la usuaria `alicia`. Si se especifica

[16] By creating the magic file `git-daemon-export-ok` in the repository directory.

`(user-path "ruta")`, la misma petición se traducirá en una petición de acceso al repositorio *ruta*`/algo` en el directorio de la usuaria `alicia`.

`listen` (predeterminadas: `'()`)
: Determina si se debe escuchar en direcciones IP o nombres de máquina específicos, de manera predeterminada escucha en cualquiera.

`port` (predeterminado: `#f`)
: Determina si se escucha en un puerto alternativo, cuyo valor predeterminado es 9418.

`whitelist` (predeterminada: `'()`)
: Si no está vacío, únicamente permite el acceso a esta lista de directorios.

`extra-options` (predeterminadas: `'()`)
: Extra options that will be passed to `git daemon`.[17]

El protocolo `git://` carece de verificación. Cuando se obtienen datos de un repositorio a través del protocolo `git://`, no puede tener plena confianza en que los datos que reciba procedan realmente de la máquina que ha indicado, y su conexión puede estar sujeta a interceptaciones. Es mejor usar un transporte verificado y cifrado, como `https`. Aunque Git le permite servir repositorios usando servidores web poco sofisticados basados en archivos, existe un protocolo más rápido implementado en el programa `git-http-backend`. Este programa es el motor de un servicio web de Git adecuado. Está diseñado para ejecutarse tras una pasarela FastCGI. See Section 11.10.20 [Servicios Web], page 469, para más información sobre la ejecución del daemon `fcgiwrap` necesario.

Guix tiene un tipo de datos de configuración distinto para proporcionar repositorios Git sobre HTTP.

`git-http-configuration` [Tipo de datos]
: Data type representing the configuration for a future `git-http-service-type`; can currently be used to configure Nginx through `git-http-nginx-location-configuration`.

`package` (predeterminado: *git*)
: El objeto paquete del sistema distribuido de control de versiones Git.

`git-root` (predeterminada: `/srv/git`)
: Directorio que contiene los repositorios Git que se expondrán al mundo.

`export-all?` (predeterminado: `#f`)
: Determina si se expondrá el acceso a todos los repositorios en *git-root*, incluso si no contienen el archivo `git-daemon-export-ok`.

`uri-path` (predeterminada: '`/git/`')
: Prefijo de la ruta del acceso de Git. Con el prefijo predeterminado '`/git/`', '`http://`*servidor*`/git/`*repositorio*`.git`' se traducirá en `/srv/git/`*repositorio*`.git`. Las peticiones cuyas rutas URI no comiencen con dicho prefijo no se pasan a esta instancia de Git.

[17] Run `man git-daemon` for more information.

`fcgiwrap-socket` (predeterminado: `127.0.0.1:9000`)
: Socket en el que el daemon `fcgiwrap` escucha. See Section 11.10.20 [Servicios Web], page 469.

No existe actualmente `git-http-service-type`; en vez de eso puede crear una configuración `nginx-location-configuration` desde `git-http-configuration` y añadir dicha configuración al servidor web.

`git-http-nginx-location-configuration` [Procedimiento]
[*config*=(*git-http-configuration*)]
: Calcula una configuración `nginx-location-configuration` que corresponde con la configuración http de Git proporcionada. Un ejemplo de definición de servicio nginx que ofrece el directorio predeterminado `/srv/git` sobre HTTPS podría ser:

```
(service nginx-service-type
         (nginx-configuration
          (server-blocks
           (list
            (nginx-server-configuration
             (listen '("443 ssl"))
             (server-name "git.mi-maquina.org")
             (ssl-certificate
              "/etc/certs/git.mi-maquina.org/fullchain.pem")
             (ssl-certificate-key
              "/etc/certs/git.mi-maquina.org/privkey.pem")
             (locations
              (list
               (git-http-nginx-location-configuration
                (git-http-configuration (uri-path "/"))))))))))
```

This example assumes that you are using Let's Encrypt to get your TLS certificate. See Section 11.10.21 [Servicios de certificados], page 490. The default `certbot` service will redirect all HTTP traffic on `git.my-host.org` to HTTPS. You will also need to add an `fcgiwrap` proxy to your system services. See Section 11.10.20 [Servicios Web], page 469.

## Servicio Cgit

Cgit (`https://git.zx2c4.com/cgit/`) es un servidor de fachada para repositiorios Git escrito en C.

El ejemplo siguiente configura el servicio con los valores predeterminados. Por omisión, se puede acceder a Cgit en el puerto 80 (`http://localhost:80`).

```
(service cgit-service-type)
```

El tipo `file-object` designa o bien un objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167), o bien una cadena.

Los campos disponibles de `cgit-configuration` son:

`package package` [parámetro de `cgit-configuration`]
: El paquete CGIT.

`lista-nginx-server-configuration` [parámetro de `cgit-configuration`]
`nginx`
Configuración de NGINX.

`file-object about-filter` [parámetro de `cgit-configuration`]
Especifica una orden que se llamará para dar formato al contenido de las páginas "about" (tanto al nivel superior como cada repositorio).

El valor predeterminado es '`""`'.

`string agefile` [parámetro de `cgit-configuration`]
Especifica una ruta, relativa a cada ruta de repositorio, que puede usarse para especificar una fecha y hora de la revisión más reciente del repositorio.

El valor predeterminado es '`""`'.

`file-object auth-filter` [parámetro de `cgit-configuration`]
Especifica una orden que se invocará para la validación de acceso al repositorio.

El valor predeterminado es '`""`'.

`string branch-sort` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`age`', activa la ordenación por fecha en la lista de referencias de ramas, y cuando tiene valor '`name`' activa la ordenación por nombre de rama.

El valor predeterminado es '`"name"`'.

`string cache-root` [parámetro de `cgit-configuration`]
Ruta usada para el almacenamiento de las entradas de caché de cgit.

El valor predeterminado es '`"/var/cache/cgit"`'.

`integer cache-static-ttl` [parámetro de `cgit-configuration`]
Número que especifica el tiempo de vida, en minutos, de la versión en caché de las páginas del repositiorio accedidas mediante un hash SHA1 fijo.

El valor predeterminado es '`-1`'.

`integer cache-dynamic-ttl` [parámetro de `cgit-configuration`]
Número que especifica el tiempo de vida, en minutos, de la versión en caché de las páginas del repositorio accedidas sin un hash SHA1 fijo.

El valor predeterminado es '`5`'.

`integer cache-repo-ttl` [parámetro de `cgit-configuration`]
Número que especifica el tiempo de vida, en minutos, de la versión en caché de la página de resumen del repositorio.

El valor predeterminado es '`5`'.

`integer cache-root-ttl` [parámetro de `cgit-configuration`]
Número que especifica el tiempo de vida, en minutos, de la versión en caché de la página del índice de repositorios.

El valor predeterminado es '`5`'.

`integer cache-scanrc-ttl` [parámetro de `cgit-configuration`]
Número que especifica el tiempo de vida, en minutos, para el resultado de la búsqueda en una ruta para repositorios Git.

El valor predeterminado es '`15`'.

`integer cache-about-ttl` [parámetro de `cgit-configuration`]
Número que especifica el tiempo de vida, en minutos, de la versión en caché de la página de información del repositorio.

El valor predeterminado es '`15`'.

`integer cache-snapshot-ttl` [parámetro de `cgit-configuration`]
Número que especifica el tiempo de vida, en minutos, de la versión en caché de las instantáneas.

El valor predeterminado es '`5`'.

`integer cache-size` [parámetro de `cgit-configuration`]
El número máximo de entradas en la caché de cgit. Cuando el valor es '`0`', se desactiva el almacenamiento en caché.

El valor predeterminado es '`0`'.

`boolean case-sensitive-sort?` [parámetro de `cgit-configuration`]
Ordena los elementos en la lista del repositorio diferenciando las mayúsculas.

El valor predeterminado es '`#t`'

`lista clone-prefix` [parámetro de `cgit-configuration`]
Lista de prefijos comunes que, cuando se combinen con la URL de un repositorio, generan una URL que permite el clonado del repositorio.

El valor predeterminado es '`'()`'.

`lista clone-url` [parámetro de `cgit-configuration`]
Lista de plantillas `clone-url`.

El valor predeterminado es '`'()`'.

`file-object commit-filter` [parámetro de `cgit-configuration`]
Orden ejecutada para el formato de mensajes de revisión.

El valor predeterminado es '`""`'.

`string commit-sort` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`date`', activa la ordenación estricta por fecha en el registro histórico de revisiones, y cuando tiene valor '`topo`' activa la ordenación estricta topológica.

El valor predeterminado es '`"git log"`'.

`file-object css` [parámetro de `cgit-configuration`]
URL que especifica el documento css incluido en todas las páginas de cgit.

El valor predeterminado es '`"/share/cgit/cgit.css"`'.

`file-object email-filter` [parámetro de `cgit-configuration`]
Especifica una orden que se llamará para dar formato a los nombres y las direcciones de correo electrónico de las revisoras, autoras y etiquetadoras con el que se representarán en varios lugares de la interfaz cgit.

El valor predeterminado es '`""`'.

`boolean embedded?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`#t`', hace que cgit genere un fragmento HTML adecuado para embeberse en otras páginas HTML.

El valor predeterminado es '`#f`'

`boolean enable-commit-graph?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene el valor '`#t`', hace que cgit imprima un grafo histórico de la revisión de arte ASCII a la izquierda de los mensajes de revisión en la página del histórico del repositorio.

El valor predeterminado es '`#f`'

`boolean enable-filter-overrides?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`#t`', permite que todas las configuraciones de filtros se sustituyan en los archivos cgitrc específicos del repositorio.

El valor predeterminado es '`#f`'

`boolean enable-follow-links?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`#t`', permite a las usuarias seguir un archivo en la vista de registro (log).

El valor predeterminado es '`#f`'

`boolean enable-http-clone?` [parámetro de `cgit-configuration`]
Si se proporciona '`#t`', cgit actuará como un simple servidor HTTP para los clones de Git.

El valor predeterminado es '`#t`'

`boolean enable-index-links?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`#t`', hace que cgit genere enlaces adicionales `"summary"` (resumen), `"commit"` (revisión) y `"tree"` (árbol) para cada repositorio en el índice de repositorios.

El valor predeterminado es '`#f`'

`boolean enable-index-owner?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`#t`', hace que cgit muestre la propietaria de cada repositorio en el índice del repositorios.

El valor predeterminado es '`#t`'

`boolean enable-log-filecount?` [parámetro de `cgit-configuration`]
Opción que, cuando se proporciona el valor '`#t`', hace que cgit imprima el número de archivos modificados por cada revisión en la página de registro histórico del repositorio (`"log"`).

El valor predeterminado es '`#f`'

`boolean enable-log-linecount?` [parámetro de `cgit-configuration`]
Opción que, cuando se proporciona el valor '`#t`', hace que cgit imprima el número de líneas añadidas y eliminadas en cada revisión en la página de registro histórico (`"log"`).

El valor predeterminado es '`#f`'

`boolean enable-remote-branches?` [parámetro de `cgit-configuration`]
Opción que, cuando se proporciona el valor '`#t`', hace que cgit muestre ramas remotas en las vistas de resumen (`"summary"`) y de referencias (`"refs"`).

El valor predeterminado es '`#f`'

`boolean enable-subject-links?` [parámetro de `cgit-configuration`]
Opción que, cuando se proporciona el valor `1`, hace que cgit use el asunto de la revisión previa como texto del enlace cuando se generen enlaces a revisiones previas en la vista de la revisión.

El valor predeterminado es '`#f`'

`boolean enable-html-serving?` [parámetro de `cgit-configuration`]
Opción que, cuando se proporciona el valor '`#t`', hace que cgit use el asunto de la revisión previa como texto del enlace cuando se generen enlaces a revisiones previas en la vista de la revisión.

El valor predeterminado es '`#f`'

`boolean enable-tree-linenumbers?` [parámetro de `cgit-configuration`]
Opción que, cuando se proporciona el valor '`#t`', hace que cgit genere enlaces de números de línea para los archivos (blob) de texto plano impresos en la vista de árbol.

El valor predeterminado es '`#t`'

`boolean enable-git-config?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`#f`', permite que cgit use la configuración de Git para fijar el valor de cualquier opción específica del repositorio.

El valor predeterminado es '`#f`'

`file-object favicon` [parámetro de `cgit-configuration`]
URL usada para icono de los enlaces a cgit.

El valor predeterminado es '`"/favicon.ico"`'.

`string footer` [parámetro de `cgit-configuration`]
El contenido del archivo especificado con esta opción se incluirá literalmente en la parte inferior de todas las páginas (es decir, sustituye al mensaje estándar `"generated by..."`).

El valor predeterminado es '`""`'.

`string head-include` [parámetro de `cgit-configuration`]
El contenido del archivo especificado con esta opción se incluirá literalmente en la sección HEAD de HTML en todas las páginas.

El valor predeterminado es '`""`'.

`string header` [parámetro de `cgit-configuration`]
El contenido del archivo especificado con esta opción se incluirá literalmente en la parte superior de todas las páginas.

El valor predeterminado es '`""`'.

`file-object include` [parámetro de `cgit-configuration`]
Nombre de un archivo de configuración que debe incluirse antes de procesar el resto del archivo de configuración actual.

El valor predeterminado es '`""`'.

`string index-header` [parámetro de `cgit-configuration`]
El contenido del archivo especificado en esta opción se incluirá literalmente sobre el índice de repositorios.

El valor predeterminado es '`""`'.

`string index-info` [parámetro de `cgit-configuration`]
El contenido del archivo especificado con esta opción se incluirá de manera literal bajo la cabecera en la página de índice del repositorio.

El valor predeterminado es '`""`'.

`boolean local-time?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`#t`', hace que cgit imprima las fechas de revisión y etiqueta en la zona horaria del servidor.

El valor predeterminado es '`#f`'

`file-object logo` [parámetro de `cgit-configuration`]
URL que especifica la fuente de una imagen usada como logo en todas las páginas de cgit.

El valor predeterminado es '`"/share/cgit/cgit.png"`'.

`string logo-link` [parámetro de `cgit-configuration`]
URL que se carga al pulsar la imagen del logo de cgit.

El valor predeterminado es '`""`'.

`file-object owner-filter` [parámetro de `cgit-configuration`]
Orden que se ejecuta para dar formato a la columna de propietaria (Owner) de la página principal.

El valor predeterminado es '`""`'.

`integer max-atom-items` [parámetro de `cgit-configuration`]
Número de elementos a mostrar en la vista de "atom feeds".

El valor predeterminado es '`10`'.

`integer max-commit-count` [parámetro de `cgit-configuration`]
Número de entradas a mostrar por página en la vista del registro histórico (`"log"`).

El valor predeterminado es '`50`'.

`integer max-message-length` [parámetro de `cgit-configuration`]
Número de caracteres del mensaje de la revisión a mostrar en la vista del registro histórico ("log").

El valor predeterminado es '`80`'.

`integer max-repo-count` [parámetro de `cgit-configuration`]
Especifica el número de entradas a mostrar por página en la página de índice de repositorios.

El valor predeterminado es '`50`'.

`integer max-repodesc-length` [parámetro de `cgit-configuration`]
Especifica el número máximo de caracteres mostrados en la descripción del repositorio en la página del índice de repositorios.

El valor predeterminado es '`80`'.

`integer max-blob-size` [parámetro de `cgit-configuration`]
Especifica el tamaño máximo de un archivo (blob) para mostrarlo en HTML en kilobytes.

El valor predeterminado es '`0`'.

`string max-stats` [parámetro de `cgit-configuration`]
Periodo estadístico máximo. Son valores aceptados '`week`', '`month`', '`quarter`' and '`year`'.

El valor predeterminado es '`""`'.

`mimetype-alist mimetype` [parámetro de `cgit-configuration`]
Tipo MIME para la extensión de archivo especificada.

El valor predeterminado es '`'((gif "image/gif") (html "text/html") (jpg "image/jpeg") (jpeg "image/jpeg") (pdf "application/pdf") (png "image/png") (svg "image/svg+xml"))`'.

`file-object mimetype-file` [parámetro de `cgit-configuration`]
Especifica el archivo usado para la búsqueda automática de tipos MIME.

El valor predeterminado es '`""`'.

`string module-link` [parámetro de `cgit-configuration`]
Texto que se usará como la cadena de formato para un enlace cuando un submódulo se imprime en el listado del directorio.

El valor predeterminado es '`""`'.

`boolean nocache?` [parámetro de `cgit-configuration`]
Si se proporciona el valor '`#t`', se desactiva la caché.

El valor predeterminado es '`#f`'

`boolean noplainemail?` [parámetro de `cgit-configuration`]
Si se proporciona '`#t`', se desactiva la impresión de direcciones de correo completas de las autoras.

El valor predeterminado es '`#f`'

`boolean noheader?` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`#t`', hace que cgit omita la cabecera estándar en todas las páginas.

El valor predeterminado es '`#f`'

`lista-proyectos project-list` [parámetro de `cgit-configuration`]
Una lista de subdirectorios dentro de `repository-directory`, relativa a él, que debe cargarse como repositorios Git. La lista vacía significa que se cargarán todos los subdirectorios.

El valor predeterminado es '`'()`'.

`file-object readme` [parámetro de `cgit-configuration`]
Text which will be used as default `repository-cgit-configuration readme`.

El valor predeterminado es '`""`'.

`boolean remove-suffix?` [parámetro de `cgit-configuration`]
Si se proporciona `#t` y `repository-directory` está activo, si se encuentra algún repositorio con el sufijo `.git`, se elimina dicho sufijo de la URL y del nombre.

El valor predeterminado es '`#f`'

`integer renamelimit` [parámetro de `cgit-configuration`]
Número máximo de archivos considerados durante la detección de renombrados.

El valor predeterminado es '`-1`'.

`string repository-sort` [parámetro de `cgit-configuration`]
La forma de ordenar los repositorios de cada sección.

El valor predeterminado es '`""`'.

`lista-robots robots` [parámetro de `cgit-configuration`]
Texto usado como contenido de la meta-etiqueta `robots`.

El valor predeterminado es '`'("noindex" "nofollow")`'.

`string root-desc` [parámetro de `cgit-configuration`]
Texto impreso bajo la cabecera en la página de índice del repositorio.

El valor predeterminado es '`"a fast webinterface for the git dscm"`'.<

`string root-readme` [parámetro de `cgit-configuration`]
El contenido del archivo especificado con esta opción se incluirá de manera literal tras el enlace de información del repositorio ("about") en la página de índice del repositorio.

El valor predeterminado es '`""`'.

`string root-title` [parámetro de `cgit-configuration`]
Texto impreso como cabecera en la página de índice del repositorio.

El valor predeterminado es '`""`'.

`boolean scan-hidden-path` [parámetro de `cgit-configuration`]
Si se proporciona '`#t`' y repository-directory está activo, repository-directory recorrerá recursivamente los directorios cuyos nombres comiencen por punto. En otro caso, repository-directory no tendrá en cuenta dichos directorios, considerados ocultos ("hidden"). Tenga en cuenta que esto no incluye al directorio `.git` en repositorios con una copia de trabajo.

El valor predeterminado es '`#f`'

`lista snapshots` [parámetro de `cgit-configuration`]
Texto que especifica el conjunto predeterminado de formatos de instantánea para los que cgit genera enlaces.

El valor predeterminado es '`'()`'.

`directorio-repositorio` [parámetro de `cgit-configuration`]
`repository-directory`
Nombre del directorio en el que se buscarán repositorios (representa `scan-path`).

El valor predeterminado es '`"/srv/git"`'.

`string section` [parámetro de `cgit-configuration`]
Nombre actual de la sección de repositorios - todos los repositorios definidos tras esta opción heredarán el nombre actual de sección.

El valor predeterminado es '`""`'.

`string section-sort` [parámetro de `cgit-configuration`]
Opción que, cuando tiene valor '`1`', ordenará las secciones en el listado de repositorios por nombre.

El valor predeterminado es '`""`'.

`integer section-from-path` [parámetro de `cgit-configuration`]
Número que, si se define antes de repository-directory, especifica cuantos elementos de ruta de cada ruta de repositorio se usarán como nombre de sección predeterminado.

El valor predeterminado es '`0`'.

`boolean side-by-side-diffs?` [parámetro de `cgit-configuration`]
Si se proporciona el valor '`#t`' se muestran las diferencias lado a lado en vez de usar el formato universal de manera predeterminada.

El valor predeterminado es '`#f`'

`file-object source-filter` [parámetro de `cgit-configuration`]
Especifica la orden que se ejecutará para dar formato a los archivos (blob) de texto plano en la vista de árbol.

El valor predeterminado es '`""`'.

`integer summary-branches` [parámetro de `cgit-configuration`]
Especifica el número de ramas mostradas en la vista resumen ("summary") del repositorio.

El valor predeterminado es '`10`'.

`integer summary-log` [parámetro de `cgit-configuration`]
Especifica el número de entradas del registro mostradas en la vista resumen ("summary") del repositorio.

El valor predeterminado es '`10`'.

`integer summary-tags` [parámetro de `cgit-configuration`]
Especifica el número que etiquetas que se mostrarán en la vista resumen ("summary") del repositorio.

El valor predeterminado es '`10`'.

`string strict-export` [parámetro de `cgit-configuration`]
Nombre de archivo que, en caso de especificarse, debe estar presente en el repositiorio para que se permita el acceso de cgit a dicho repositorio.

El valor predeterminado es '`""`'.

`string virtual-root` [parámetro de `cgit-configuration`]
URL que, en caso de especificarse, se usará como raíz de todos los enlaces de cgit.

El valor predeterminado es '`"/"`'.

`lista-repository-cgit-configuration repositories` [parámetro de `cgit-configuration`]
A list of `repository-cgit-configuration` records.

El valor predeterminado es '`'()`'.

Los campos disponibles de `repository-cgit-configuration` son:

`repo-list snapshots` [parámetro de `repository-cgit-configuration`]
Una máscara de los formatos de instantánea para este repositorio para los que cgit genera enlaces, restringida por la opción de configuración global `snapshots`.

El valor predeterminado es '`'()`'.

`repo-file-object source-filter` [parámetro de `repository-cgit-configuration`]
Sustituye al valor predeterminado de `source-filter`.

El valor predeterminado es '`""`'.

`repo-string url` [parámetro de `repository-cgit-configuration`]
La URL relativa usada para el acceso al repositorio.

El valor predeterminado es '`""`'.

`repo-file-object about-filter` [parámetro de `repository-cgit-configuration`]
Sustituye al valor predeterminado de `about-filter`.

El valor predeterminado es '`""`'.

`repo-string branch-sort` [parámetro de `repository-cgit-configuration`]
Cuando se proporciona el valor '`age`', activa la ordenación por fecha en la lista de referencias de ramas, y cuando se proporciona '`name`' se activa la ordenación por nombre de rama.

El valor predeterminado es '`""`'.

`repo-list clone-url` [parámetro de `repository-cgit-configuration`]
Una lista de URL que se pueden usar para clonar el repositorio.

El valor predeterminado es ‘`'()`’.

`repo-file-object commit-filter` [parámetro de `repository-cgit-configuration`]
Sustituye al valor predeterminado de `commit-filter`.

El valor predeterminado es ‘`""`’.

`repo-string commit-sort` [parámetro de `repository-cgit-configuration`]
Opción que, cuando tiene valor ‘`date`’, activa la ordenación estricta por fecha en el registro histórico de revisiones, y cuando tiene valor ‘`topo`’ activa la ordenación estricta topológica.

El valor predeterminado es ‘`""`’.

`repo-string defbranch` [parámetro de `repository-cgit-configuration`]
Nombre de la rama predeterminada de este repositorio. Si no existe dicha rama en el repositorio, se usará como predeterminado el primer nombre de rama encontrado (tras su ordenación). De manera predeterminada, la rama a la que apunta HEAD, o “master” si no existe un valor adecuado para HEAD.

El valor predeterminado es ‘`""`’.

`repo-string desc` [parámetro de `repository-cgit-configuration`]
El valor a mostrar como descripción del repositorio.

El valor predeterminado es ‘`""`’.

`repo-string homepage` [parámetro de `repository-cgit-configuration`]
El valor a mostrar como página web principal del repositorio.

El valor predeterminado es ‘`""`’.

`repo-file-object email-filter` [parámetro de `repository-cgit-configuration`]
Sustituye al valor predeterminado de `email-filter`.

El valor predeterminado es ‘`""`’.

`maybe-repo-boolean enable-commit-graph?` [parámetro de `repository-cgit-configuration`]
Esta opción se puede usar para forzar el valor de la opción de configuración global `enable-commit-graph?`.

El valor predeterminado es ‘`disabled`’.

`maybe-repo-boolean enable-log-filecount?` [parámetro de `repository-cgit-configuration`]
Esta opción se puede usar para forzar el valor de la opción de configuración global `enable-log-filecount?`.

El valor predeterminado es ‘`disabled`’.

`maybe-repo-boolean` [parámetro de `repository-cgit-configuration`]
`enable-log-linecount?`

Esta opción se puede usar para forzar el valor de la opción de configuración global `enable-log-linecount?`.

El valor predeterminado es '`disabled`'.

`maybe-repo-boolean` [parámetro de `repository-cgit-configuration`]
`enable-remote-branches?`

Opción que, cuando se proporciona el valor '`#t`', hace que cgit muestre ramas remotas en las vistas de resumen (`"summary"`) y de referencias (`"refs"`).

El valor predeterminado es '`disabled`'.

`maybe-repo-boolean` [parámetro de `repository-cgit-configuration`]
`enable-subject-links?`

Esta opción se puede usar para forzar el valor de la opción de configuración global `enable-subject-links?`.

El valor predeterminado es '`disabled`'.

`maybe-repo-boolean` [parámetro de `repository-cgit-configuration`]
`enable-html-serving?`

Esta opción se puede usar para forzar el valor de la opción de configuración global `enable-html-serving?`.

El valor predeterminado es '`disabled`'.

`repo-boolean hide?` [parámetro de `repository-cgit-configuration`]

Opción que, cuando tiene valor '`#t`', oculta el repositorio en el índice.

El valor predeterminado es '`#f`'

`repo-boolean ignore?` [parámetro de `repository-cgit-configuration`]

Opción que, cuando tiene valor '`#t`', ignora el repositorio.

El valor predeterminado es '`#f`'

`repo-file-object logo` [parámetro de `repository-cgit-configuration`]

URL que especifica la fuente de una imagen que se usará como logo en las páginas de este repositorio.

El valor predeterminado es '`""`'.

`repo-string logo-link` [parámetro de `repository-cgit-configuration`]

URL que se carga al pulsar la imagen del logo de cgit.

El valor predeterminado es '`""`'.

`repo-file-object` [parámetro de `repository-cgit-configuration`]
`owner-filter`

Sustituye al valor predeterminado de `owner-filter`.

El valor predeterminado es '`""`'.

`repo-string module-link` [parámetro de `repository-cgit-configuration`]
Texto que se usará como la cadena de formato de un enlace cuando un submódulo se imprima en el listado de un directorio. Los parámetros para la cadena de formato son la ruta y el SHA1 de la revisión del submódulo.

El valor predeterminado es ‘""’.

`ruta-enlace-módulo module-link` [parámetro de `repository-cgit-configuration`]
Texto que se usará como la cadena de formato de un enlace cuando un submódulo con la ruta de subdirectorio especificada se imprima en el listado de un directorio.

El valor predeterminado es ‘'()’.

`repo-string max-stats` [parámetro de `repository-cgit-configuration`]
Sustituye al máximo periodo estadístico predeterminado.

El valor predeterminado es ‘""’.

`repo-string name` [parámetro de `repository-cgit-configuration`]
El valor a mostrar como nombre del repositorio.

El valor predeterminado es ‘""’.

`repo-string owner` [parámetro de `repository-cgit-configuration`]
Un valor usado para identificar a la propietaria del repositorio.

El valor predeterminado es ‘""’.

`repo-string path` [parámetro de `repository-cgit-configuration`]
La ruta absoluta al directorio del repositorio.

El valor predeterminado es ‘""’.

`repo-string readme` [parámetro de `repository-cgit-configuration`]
Una ruta (relativa al repositorio) que especifica un archivo que será incluido literalmente como página de información (“About”) de este repositorio.

El valor predeterminado es ‘""’.

`repo-string section` [parámetro de `repository-cgit-configuration`]
Nombre actual de la sección de repositorios - todos los repositorios definidos tras esta opción heredarán el nombre actual de sección.

El valor predeterminado es ‘""’.

`repo-list extra-options` [parámetro de `repository-cgit-configuration`]
Opciones adicionales a agregar al final del archivo cgitrc.

El valor predeterminado es ‘'()’.

`lista extra-options` [parámetro de `cgit-configuration`]
Opciones adicionales a agregar al final del archivo cgitrc.

El valor predeterminado es ‘'()’.

No obstante, puede ser que únicamente desee usar un archivo `cgitrc` existente. En ese caso, puede proporcionar `opaque-cgit-configuration` como un registro a `cgit-service-type`. Como su nombre en inglés indica, una configuración opaca no tiene gran capacidad reflexiva.

Los campos disponibles de `opaque-cgit-configuration` son:

`package cgit` [parámetro de `opaque-cgit-configuration`]
El paquete cgit.

`string string` [parámetro de `opaque-cgit-configuration`]
El contenido de `cgitrc`, como una cadena.

Por ejemplo, si su `cgitrc` es simplemente la cadena vacía, puede instanciar un servicio cgit de esta manera:

```
(service cgit-service-type
         (opaque-cgit-configuration
          (cgitrc "")))
```

## Servicio Gitolite

Gitolite (`https://gitolite.com/gitolite/`) es una herramienta para el almacenamiento de repositorios Git en un servidor central.

Gitolite puede manejar múltiples repositorios y usuarias, y permite una configuración flexible de los permisos de las usuarias sobre los repositorios.

El siguiente ejemplo configuraría Gitolite con la usuaria predeterminada `git` y la clave pública SSH proporcionada.

```
(service gitolite-service-type
         (gitolite-configuration
           (admin-pubkey (plain-file
                          "sunombre.pub"
                          "ssh-rsa AAAA... guix@example.com"))))
```

Gitolite se configura a través de un repositorio de administración especial que puede clonar, por ejemplo, si configuró Gitolite en `example.org`, ejecutaría la siguiente orden para clonar el repositorio de administración.

```
git clone git@example.com:gitolite-admin
```

Cuando se activa el servicio Gitolite, la clave `admin-pubkey` proporcionada se insertará en el directorio `keydir` en el repositorio gitolite-admin. Si esto resultase en un cambio en el repositorio, la revisión se almacenaría con el mensaje "gitolite setup by GNU Guix".

`gitolite-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de `gitolite-service-type`.

`package` (predeterminado: *gitolite*)
Gitolite package to use. There are optional Gitolite dependencies that are not included in the default package, such as Redis and git-annex. These features can be made available by using the `make-gitolite` procedure in the `(gnu packages version-control)` module to produce a variant of Gitolite with the desired additional dependencies.

The following code returns a package in which the Redis and git-annex programs can be invoked by Gitolite's scripts:

```
(use-modules (gnu packages databases)
             (gnu packages haskell-apps)
             (gnu packages version-control))
(make-gitolite (list redis git-annex))
```

`user` (predeterminada: *git*)
Usuaria usada por Gitolite. Esta será la usuaria a la que se conectará cuando acceda a Gitolite a través de SSH.

`group` (predeterminado: *git*)
Grupo usado por Gitolite.

`home-directory` (predeterminado: "*/var/lib/gitolite*")
Directorio en el que se almacenará la configuración y repositorios de Gitolite.

`rc-file` (predeterminado: *(gitolite-rc-file)*)
Un objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167), que representa la configuración de Gitolite.

`admin-pubkey` (predeterminada: *#f*)
Un objeto "tipo-archivo" (see Section 8.12 [Expresiones-G], page 167) usado para la configuración de Gitolite. Se insertará en el directorio `keydir` dentro del repositorio gitolite-admin.

Para especificar la clave SSH como una cadena, use la función `plain-file`.

```
(plain-file "sunombre.pub" "ssh-rsa AAAA... guix@example.com")
```

`gitolite-rc-file` [Tipo de datos]
Tipo de datos que representa el archivo RC de Gitolite.

`umask` (predeterminada: `#o0077`)
Controla los permisos que Gitolite establece en los repositorios y sus contenidos.

A value like `#o0027` will give read access to the group used by Gitolite (by default: `git`). This is necessary when using Gitolite with software like cgit or gitweb.

`local-code` (default: `"$rc{GL_ADMIN_BASE}/local"`)
Allows you to add your own non-core programs, or even override the shipped ones with your own.

Please supply the FULL path to this variable. By default, directory called "local" in your gitolite clone is used, providing the benefits of versioning them as well as making changes to them without having to log on to the server.

`unsafe-pattern` (default: `#f`)
An optional Perl regular expression for catching unsafe configurations in the configuration file. See Gitolite's documentation (`https://gitolite.`

com/gitolite/git-config.html#compensating-for-unsafe_patt) for more information.

When the value is not `#f`, it should be a string containing a Perl regular expression, such as '`"[`~#\$\&()|;<>]"`', which is the default value used by gitolite. It rejects any special character in configuration that might be interpreted by a shell, which is useful when sharing the administration burden with other people that do not otherwise have shell access on the server.

`git-config-keys` (predeterminadas: `""`)
: Gitolite allows you to set git config values using the '`config`' keyword. This setting allows control over the config keys to accept.

`roles` (predeterminados: `'(("READERS" . 1) ("WRITERS" . ))`)
: Establece los nombres de rol que se permite usar a las usuarias que ejecuten la orden perms.

`enable` (predeterminados: `'("help" "desc" "info" "perms" "writable" "ssh-authkeys" "git-config" "daemon" "gitweb")`)
: Esta configuración controla las órdenes y características activadas dentro de Gitolite.

## Gitile Service

Gitile (`https://git.lepiller.eu/gitile`) is a Git forge for viewing public git repository contents from a web browser.

Gitile works best in collaboration with Gitolite, and will serve the public repositories from Gitolite by default. The service should listen only on a local port, and a webserver should be configured to serve static resources. The gitile service provides an easy way to extend the Nginx service for that purpose (see [NGINX], page 472).

The following example will configure Gitile to serve repositories from a custom location, with some default messages for the home page and the footers.

```
(service gitile-service-type
         (gitile-configuration
           (repositories "/srv/git")
           (base-git-url "https://myweb.site/git")
           (index-title "My git repositories")
           (intro '((p "This is all my public work!")))
           (footer '((p "This is the end")))
           (nginx-server-block
             (nginx-server-configuration
               (ssl-certificate
                 "/etc/certs/myweb.site/fullchain.pem")
               (ssl-certificate-key
                 "/etc/certs/myweb.site/privkey.pem")
               (listen '("443 ssl http2" "[::]:443 ssl http2"))
               (locations
                 (list
                   ;; Allow for https anonymous fetch on /git/ urls.
```

```
                    (git-http-nginx-location-configuration
                      (git-http-configuration
                        (uri-path "/git/")
                        (git-root "/var/lib/gitolite/repositories"))))))))))
```

In addition to the configuration record, you should configure your git repositories to contain some optional information. First, your public repositories need to contain the `git-daemon-export-ok` magic file that allows Git to export the repository. Gitile uses the presence of this file to detect public repositories it should make accessible. To do so with Gitolite for instance, modify your `conf/gitolite.conf` to include this in the repositories you want to make public:

```
repo foo
    R = daemon
```

In addition, Gitile can read the repository configuration to display more information on the repository. Gitile uses the gitweb namespace for its configuration. As an example, you can use the following in your `conf/gitolite.conf`:

```
repo foo
    R = daemon
    desc = A long description, optionally with <i>HTML</i>, shown on the index page
    config gitweb.name = The Foo Project
    config gitweb.synopsis = A short description, shown on the main page of the projec
```

Do not forget to commit and push these changes once you are satisfied. You may need to change your gitolite configuration to allow the previous configuration options to be set. One way to do that is to add the following service definition:

```
(service gitolite-service-type
          (gitolite-configuration
            (admin-pubkey (local-file "key.pub"))
            (rc-file
              (gitolite-rc-file
                (umask #o0027)
                ;; Allow to set any configuration key
                (git-config-keys ".*")
                ;; Allow any text as a valid configuration value
                (unsafe-patt "^$")))))
```

`gitile-configuration` [Data Type]

Data type representing the configuration for `gitile-service-type`.

`package` (default: *gitile*)
: Gitile package to use.

`host` (predeterminado: `"localhost"`)
: The host on which gitile is listening.

`port` (default: `8080`)
: The port on which gitile is listening.

`database` (default: `"/var/lib/gitile/gitile-db.sql"`)
: The location of the database.

`repositories` (default: `"/var/lib/gitolite/repositories"`)
: The location of the repositories. Note that only public repositories will be shown by Gitile. To make a repository public, add an empty `git-daemon-export-ok` file at the root of that repository.

`base-git-url`
: The base git url that will be used to show clone commands.

`index-title` (default: `"Index"`)
: The page title for the index page that lists all the available repositories.

`intro` (default: `'()`)
: The intro content, as a list of sxml expressions. This is shown above the list of repositories, on the index page.

`footer` (default: `'()`)
: The footer content, as a list of sxml expressions. This is shown on every page served by Gitile.

`nginx-server-block`
: An nginx server block that will be extended and used as a reverse proxy by Gitile to serve its pages, and as a normal web server to serve its assets.

  You can use this block to add more custom URLs to your domain, such as a `/git/` URL for anonymous clones, or serving any other files you would like to serve.

### 11.10.32 Servicios de juegos

#### Joycond service

The joycond service allows the pairing of Nintendo joycon game controllers over Bluetooth. (see Section 11.10.9 [Servicios de escritorio], page 363, for setting up Bluetooth.)

`joycond-configuration` [Data Type]
: Data type representing the configuration of `joycond`.

  `package` (default: `joycond`)
  : The joycond package to use.

`joycond-service-type` [Variable]
: Service type for the joycond service.

#### El servicio de La batalla por Wesnoth

La batalla por Wesnoth (`https://wesnoth.org`) es un juego de estrategia táctica, de fantasía y basado en turnos, con varias campañas de una jugadora, y partidas para múltiples jugadoras (tanto en red como localmente).

`wesnothd-service-type` [Variable]
: Tipo de servicio para el servicio wesnothd. Su valor debe ser un objeto `wesnothd-configuration`. Puede instanciarlo de esta manera para ejecutar wesnothd con la configuración predeterminada:

```
(service wesnothd-service-type)
```

`wesnothd-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de `wesnothd`.

`package` (predeterminado: `wesnoth-server`)
El paquete del servidor wesnoth usado.

`port` (predeterminado: `15000`)
Número de puerto usado por el servidor.

### 11.10.33 Servicio PAM Mount

El módulo `(gnu services pam-mount)` proporciona un servicio que permite a las usuarias montar volúmenes cuando ingresen al sistema. Debe ser capaz de montar cualquier formato de volumen que el sistema permita.

`pam-mount-service-type` [Variable]
Tipo de servicio para la implementación de PAM Mount.

`pam-mount-configuration` [Tipo de datos]
Tipo de datos que representa la configuración de PAM Mount.

Toma los siguientes parámetros:

`rules` Las reglas de configuración que se usarán para generar `/etc/security/pam_mount.conf.xml`.

Las reglas de configuración son elementos SXML (see Section "SXML" in *GNU Guile Reference Manual*), y las reglas predeterminadas no incluyen el montado de ningún dispositivo para ningún usuario en el ingreso al sistema:

```
`((debug (@ (enable "0")))
  (mntoptions (@ (allow ,(string-join
                          '("nosuid" "nodev" "loop"
                            "encryption" "fsck" "nonempty"
                            "allow_root" "allow_other")
                          ","))))
  (mntoptions (@ (require "nosuid,nodev")))
  (logout (@ (wait "0")
             (hup "0")
             (term "no")
             (kill "no")))
  (mkmountpoint (@ (enable "1")
                   (remove "true"))))
```

Algunos elementos `volume` deben añadirse de manera automática para montar volúmenes en el ingreso. El siguiente ejemplo permite a la usuaria `alicia` montar su directorio `HOME` cifrado y permite al usuario `rober` montar la partición donde almacena sus datos:

```
(define reglas-pam-mount
`((debug (@ (enable "0")))
          (volume (@ (user "alicia")
                     (fstype "crypt")
```

```
                (path "/dev/sda2")
                (mountpoint "/home/alicia")))
          (volume (@ (user "rober")
                (fstype "auto")
                (path "/dev/sdb3")
                (mountpoint "/home/rober/data")
                (options "defaults,autodefrag,compress")))
          (mntoptions (@ (allow ,(string-join
                                  '("nosuid" "nodev" "loop"
                                    "encryption" "fsck" "nonempty"
                                    "allow_root" "allow_other")
                                  ","))))
          (mntoptions (@ (require "nosuid,nodev")))
          (logout (@ (wait "0")
                (hup "0")
                (term "no")
                (kill "no")))
          (mkmountpoint (@ (enable "1")
                           (remove "true")))))

    (service pam-mount-service-type
             (pam-mount-configuration
               (rules reglas-pam-mount)))
```

La lista completa de opciones posibles se puede encontrar en la página de `man` de pam_mount.conf (`http://pam-mount.sourceforge.net/pam_mount.conf.5.html`).

## PAM Mount Volume Service

PAM mount volumes are automatically mounted at login by the PAM login service according to a set of per-volume rules. Because they are mounted by PAM the password entered during login may be used directly to mount authenticated volumes, such as `cifs`, using the same credentials.

These volumes will be added in addition to any volumes directly specified in `pam-mount-rules`.

Here is an example of a rule which will mount a remote CIFS share from `//remote-server/share` into a sub-directory of `/shares` named after the user logging in:

```
(simple-service 'pam-mount-remote-share pam-mount-volume-service-type
                (list (pam-mount-volume
                       (secondary-group "users")
                       (file-system-type "cifs")
                       (server "remote-server")
                       (file-name "share")
                       (mount-point "/shares/%(USER)")
                       (options "nosuid,nodev,seal,cifsacl"))))
```

`pam-mount-volume-service-type` [Data Type]

Configuration for a single volume to be mounted. Any fields not specified will be omitted from the run-time PAM configuration. See the man page (`http://pam-mount.sourceforge.net/pam_mount.conf.5.html`) for the default values when unspecified.

`user-name` (type: maybe-string)
: Mount the volume for the given user.

`user-id` (type: maybe-integer-or-range)
: Mount the volume for the user with this ID. This field may also be specified as a pair of `(start . end)` indicating a range of user IDs for whom to mount the volume.

`primary-group` (type: maybe-string)
: Mount the volume for users with this primary group name.

`group-id` (type: maybe-integer-or-range)
: Mount the volume for the users with this primary group ID. This field may also be specified as a cons cell of `(start . end)` indicating a range of group ids for whom to mount the volume.

`secondary-group` (type: maybe-string)
: Mount the volume for users who are members of this group as either a primary or secondary group.

`file-system-type` (type: maybe-string)
: The file system type for the volume being mounted (e.g., `cifs`)

`no-mount-as-root?` (type: maybe-boolean)
: Whether or not to mount the volume with root privileges. This is normally disabled, but may be enabled for mounts of type `fuse`, or other user-level mounts.

`server` (type: maybe-string)
: The name of the remote server to mount the volume from, when necessary.

`file-name` (type: maybe-string)
: The location of the volume, either local or remote, depending on the `file-system-type`.

`mount-point` (type: maybe-string)
: Where to mount the volume in the local file-system. This may be set to `~` to indicate the home directory of the user logging in. If this field is omitted then `/etc/fstab` is consulted for the mount destination.

`options` (type: maybe-string)
: The options to be passed as-is to the underlying mount program.

`ssh?` (type: maybe-boolean)
: Enable this option to pass the login password to SSH for use with mounts involving SSH (e.g., `sshfs`).

`cipher` (type: maybe-string)
: Cryptsetup cipher name for the volume. To be used with the `crypt` `file-system-type`.

`file-system-key-cipher` (type: maybe-string)
: Cipher name used by the target volume.

`file-system-key-hash` (type: maybe-string)
: SSL hash name used by the target volume.

`file-system-key-file-name` (type: maybe-string)
: File name of the file system key for the target volume.

### 11.10.34 Servicios de Guix

## Build Farm Front-End (BFFE)

The Build Farm Front-End (`https://git.cbaines.net/guix/bffe/`) assists with building Guix packages in bulk. It's responsible for submitting builds and displaying the status of the build farm.

`bffe-service-type` [Variable]
: Service type for the Build Farm Front-End. Its value must be a `bffe-configuration` object.

`bffe-configuration` [Data Type]
: Data type representing the configuration of the Build Farm Front-End.

  `package` (default: `bffe`)
  : The Build Farm Front-End package to use.

  `user` (default: `"bffe"`)
  : Usuaria del sistema que ejecuta el servicio.

  `group` (default: `"bffe"`)
  : Grupo del sistema que ejecuta el servicio.

  `arguments`
  : A list of arguments to the Build Farm Front-End. These are passed to the `run-bffe-service` procedure when starting the service.

    For example, the following value directs the Build Farm Front-End to submit builds for derivations available from `data.guix.gnu.org` to the Build Coordinator instance assumed to be running on the same machine.

```
(list
 #:build
 (list
  (build-from-guix-data-service
   (data-service-url "https://data.guix.gnu.org")
   (build-coordinator-url "http://127.0.0.1:8746")
   (branches '("master"))
   (systems '("x86_64-linux" "i686-linux"))
   (systems-and-targets
    (map (lambda (target)
           (cons "x86_64-linux" target))
         '("aarch64-linux-gnu"
```

```
                "i586-pc-gnu")))
          (build-priority (const 0))))
        #:web-server-args
        '(#:event-source "https://example.com"
          #:controller-args
          (#:title "example.com build farm")))
```

`extra-environment-variables` (default: *'()*)
: Extra environment variables to set via the shepherd service.

## Coordinador de construcciones de Guix

El coordinador de construcciones de Guix (`https://git.cbaines.net/guix/build-coordinator/`) ayuda en la distribución de las construcciones de derivaciones entre máquinas que ejecuten un *agente*. El daemon de construcción se usa todavía para la construcción de las derivaciones, pero el coordinador de construcciones de Guix gestiona su lanzamiento y trabaja con los resultados.

The Guix Build Coordinator consists of one *coordinator*, and one or more connected *agent* processes. The coordinator process handles clients submitting builds, and allocating builds to agents. The agent processes talk to a build daemon to actually perform the builds, then send the results back to the coordinator.

Existe un guión para ejecutar el componente de coordinación del servicio de coordinación de construcciones de Guix, pero el servicio de Guix utiliza un guión Guile personalizado en vez de este, para mejorar la integración con las expresiones-G usadas en la configuración.

`guix-build-coordinator-service-type` [Variable]
: Tipo de servicio para la coordinación de construcciones de Guix. Su valor debe ser un objeto `guix-build-coordinator-configuration`.

`guix-build-coordinator-configuration` [Tipo de datos]
: Tipo de datos que representa la configuración del servicio de coordinación de construcciones de Guix.

  `package` (predeterminado: `guix-build-coordinator`)
  : El paquete del servicio de coordinación de construcciones de Guix usado.

  `user` (predeterminado: `"guix-build-coordinator"`)
  : Usuaria del sistema que ejecuta el servicio.

  `group` (predeterminado: `"guix-build-coordinator"`)
  : Grupo del sistema que ejecuta el servicio.

  `database-uri-string` (predeterminada: `"sqlite:///var/lib/guix-build-coordinator/guix_build_coordinator.db"`)
  : URI usada para la conexión a la base de datos.

  `agent-communication-uri` (predeterminada: `"http://0.0.0.0:8745"`)
  : La URI que describe cómo escuchar peticiones de los procesos agentes.

  `client-communication-uri` (predeterminada: `"http://127.0.0.1:8746"`)
  : La URI que describe cómo escuchar peticiones de los clientes. La interfaz para los clientes permite la emisión de construcciones y no implementa

identificación actualmente, por lo que tenga cuidado cuando configure este valor.

`allocation-strategy` (predeterminada: `#~basic-build-allocation-strategy`)
: Una expresión-G para la estrategia de reservas usada. Es un procedimiento que recibe la ruta del almacén como un parámetro y rellena la planificación de reservas en la base de datos.

`hooks` (predeterminada: *'()*)
: Una lista asociativa de procedimientos de extensión[18]. Proporcionan una forma de ejecutar código arbitrario tras ciertos eventos, como inicio del procesamiento del resultado de una construcción.

`parallel-hooks` (default: *'()*)
: Hooks can be configured to run in parallel. This parameter is an association list of hooks to do in parallel, where the key is the symbol for the hook and the value is the number of threads to run.

`guile` (predeterminado: `guile-3.0-latest`)
: El paquete Guile con el que se ejecuta la coordinación de construcciones de Guix.

`extra-environment-variables` (default: *'()*)
: Extra environment variables to set via the shepherd service.

`guix-build-coordinator-agent-service-type` [Variable]
: Tipo de servicio para un agente del coordinador de construcciones de Guix. Su valor debe ser un objeto `guix-build-coordinator-agent-configuration`.

`guix-build-coordinator-agent-configuration` [Tipo de datos]
: Tipo de datos que representa la configuración de un agente del coordinador de construcciones de Guix.

`package` (default: `guix-build-coordinator/agent-only`)
: El paquete del servicio de coordinación de construcciones de Guix usado.

`user` (predeterminado: `"guix-build-coordinator-agent"`)
: Usuaria del sistema que ejecuta el servicio.

`coordinator` (predeterminado: `"http://localhost:8745"`)
: URI usada para la conexión al nodo coordinador.

`authentication`
: Record describing how this agent should authenticate with the coordinator. Possible record types are described below.

`systems` (predeterminado: `#f`)
: Los sistemas para los cuales este agente debe obtener construcciones. Los procesos agente usan el sistema sobre el que se ejecuten como valor predeterminado.

`max-parallel-builds` (predeterminado: `1`)
: El número de construcciones que se ejecutan en paralelo.

[18] NdT: "hook" en inglés tiene el sentido del gancho donde se "enganchan" las extensiones.

`max-parallel-uploads` (default: `1`)
: The number of uploads to perform in parallel.

`max-allocated-builds` (default: `#f`)
: The maximum number of builds this agent can be allocated.

`max-1min-load-average` (default: `#f`)
: Load average value to look at when considering starting new builds, if the 1 minute load average exceeds this value, the agent will wait before starting new builds.

  This will be unspecified if the value is `#f`, and the agent will use the number of cores reported by the system as the max 1 minute load average.

`derivation-substitute-urls` (default: `#f`)
: Identificadores URL a partir de los cuales se intentará obtener sustituciones para derivaciones, si las derivaciones no están ya disponibles.

`non-derivation-substitute-urls` (default: `#f`)
: Identificadores URL a partir de los cuales se intentará obtener sustituciones para las entradas de las construcciones, si los elementos de entrada del almacén no están ya disponibles.

`guix-build-coordinator-agent-password-auth` [Data Type]

Data type representing an agent authenticating with a coordinator via a UUID and password.

`uuid` El identificador UUID del agente. Debe ser generado por el proceso de coordinación, almacenarse en la base de datos de coordinación, y usarse por parte del agente en cuestión.

`password` The password to use when connecting to the coordinator.

`guix-build-coordinator-agent-password-file-auth` [Data Type]

Data type representing an agent authenticating with a coordinator via a UUID and password read from a file.

`uuid` El identificador UUID del agente. Debe ser generado por el proceso de coordinación, almacenarse en la base de datos de coordinación, y usarse por parte del agente en cuestión.

`password-file`
: Un archivo que contiene la contraseña usada para la conexión a la máquina coordinadora.

`guix-build-coordinator-agent-dynamic-auth` [Data Type]

Data type representing an agent authenticating with a coordinator via a dynamic auth token and agent name.

`agent-name`
: Name of an agent, this is used to match up to an existing entry in the database if there is one. When no existing entry is found, a new entry is automatically added.

`token` Dynamic auth token, this is created and stored in the coordinator database, and is used by the agent to authenticate.

`guix-build-coordinator-agent-dynamic-auth-with-file` [Data Type]
Data type representing an agent authenticating with a coordinator via a dynamic auth token read from a file and agent name.

`agent-name`
Name of an agent, this is used to match up to an existing entry in the database if there is one. When no existing entry is found, a new entry is automatically added.

`token-file`
File containing the dynamic auth token, this is created and stored in the coordinator database, and is used by the agent to authenticate.

## Servicio de datos de Guix

El servicio de datos de Guix (`http://data.guix.gnu.org`) procesa, almacena y proporciona datos acerca de GNU Guix. Esto incluye información sobre paquetes, derivaciones y avisos de "lint".

Los datos se almacenan en una base de datos PostgreSQL, y están disponibles a través de una interfaz web.

`guix-data-service-type` [Variable]
Tipo de servicio para el servicio de datos de Guix. Su valor debe ser un objeto `guix-data-service-configuration`. El servicio opcionalmente extiende el servicio getmail, puesto que la lista de correo guix-commits se usa para conocer los cambios del repositorio git de Guix.

`guix-data-service-configuration` [Tipo de datos]
Tipo de datos que representa la configuración del servicio de datos de Guix.

`package` (predeterminado: `guix-data-service`)
El paquete del servicio de datos de Guix usado.

`user` (usuaria: `"guix-data-service"`)
Usuaria del sistema que ejecuta el servicio.

`group` (predeterminado: `"guix-data-service"`)
Grupo del sistema que ejecuta el servicio.

`port` (predeterminado: `8765`)
El puerto al que se asociará el servicio web.

`host` (predeterminada: `"127.0.0.1"`)
El nombre de máquina al que se asociará el servicio web.

`getmail-idle-mailboxes` (predeterminado: `#f`)
Si se proporciona un valor, es la lista de bandejas de correo en las cuales la configuración debe indicar su lectura al servicio getmail.

`commits-getmail-retriever-configuration` (predeterminado: `#f`)
: Si se proporciona un valor, es el objeto `getmail-retriever-configuration` con el que se configura getmail para obtener recibir el correo de la lista guix-commits.

`extra-options` (predeterminadas: *'()*)
: Opciones de línea de órdenes adicionales para `guix-data-service`.

`extra-process-jobs-options` (predeterminadas: *'()*)
: Opciones de línea de órdenes adicionales para `guix-data-service-process-jobs`.

## Guix Home Service

The Guix Home service is a way to let Guix System deploy the home environment of one or more users (see Chapter 13 [Home Configuration], page 671, for more on Guix Home). That way, the system configuration embeds declarations of the home environment of those users and can be used to deploy everything consistently at once, saving users the need to run `guix home reconfigure` independently.

`guix-home-service-type` [Variable]
: Service type for the Guix Home service. Its value must be a list of lists containing user and home environment pairs. The key of each pair is a string representing the user to deploy the configuration under and the value is a home-environment configuration.

```
(use-modules (gnu home))

(define my-home
  (home-environment
    ...))

(operating-system
  (services (append (list (service guix-home-service-type
                                   `(("alice" ,my-home))))
                    %base-services)))
```

This service can be extended by other services to add additional home environments, as in this example:

```
(simple-service 'my-extra-home home-service-type
                `(("bob" ,my-extra-home))))
```

## Nar Herder

The Nar Herder (`https://git.cbaines.net/guix/nar-herder/about/`) is a utility for managing a collection of nars.

`nar-herder-type` [Variable]
: Service type for the Guix Data Service. Its value must be a `nar-herder-configuration` object. The service optionally extends the getmail service, as the guix-commits mailing list is used to find out about changes in the Guix git repository.

`nar-herder-configuration` [Data Type]

Tipo de datos que representa la configuración del servicio de datos de Guix.

`package` (default: `nar-herder`)
: The Nar Herder package to use.

`user` (default: `"nar-herder"`)
: Usuaria del sistema que ejecuta el servicio.

`group` (default: `"nar-herder"`)
: Grupo del sistema que ejecuta el servicio.

`port` (default: `8734`)
: Número de puerto usado por el servidor.

`host` (predeterminada: `"127.0.0.1"`)
: The host to bind the server to.

`mirror` (predeterminado: `#f`)
: Optional URL of the other Nar Herder instance which should be mirrored. This means that this Nar Herder instance will download it's database, and keep it up to date.

`database` (default: `"/var/lib/nar-herder/nar_herder.db"`)
: Location for the database. If this Nar Herder instance is mirroring another, the database will be downloaded if it doesn't exist. If this Nar Herder instance isn't mirroring another, an empty database will be created.

`database-dump` (default: `"/var/lib/nar-herder/nar_herder_dump.db"`)
: Location of the database dump. This is created and regularly updated by taking a copy of the database. This is the version of the database that is available to download.

`storage` (default: `#f`)
: Optional location in which to store nars.

`storage-limit` (default: `"none"`)
: Limit in bytes for the nars stored in the storage location. This can also be set to "none" so that there is no limit.

  When the storage location exceeds this size, nars are removed according to the nar removal criteria.

`storage-nar-removal-criteria` (default: `'()`)
: Criteria used to remove nars from the storage location. These are used in conjunction with the storage limit.

  When the storage location exceeds the storage limit size, nars will be checked against the nar removal criteria and if any of the criteria match, they will be removed. This will continue until the storage location is below the storage limit size.

  Each criteria is specified by a string, then an equals sign, then another string. Currently, only one criteria is supported, checking if a nar is stored on another Nar Herder instance.

`ttl` (predeterminado: `#f`)
: Produce cabeceras HTTP `Cache-Control` que anuncian un tiempo-de-vida (TTL) de *ttl*. *ttl* debe indicar una duración: `5d` significa 5 días, `1m` significa un mes, etc.

  This allows the user's Guix to keep substitute information in cache for *ttl*.

`new-ttl` (default: `#f`)
: If specified, this will override the `ttl` setting when used for the `Cache-Control` headers, but this value will be used when scheduling the removal of nars.

  Use this setting when the TTL is being reduced to avoid removing nars while clients still have cached narinfos.

`negative-ttl` (default: `#f`)
: Similarly produce `Cache-Control` HTTP headers to advertise the time-to-live (TTL) of *negative* lookups—missing store items, for which the HTTP 404 code is returned. By default, no negative TTL is advertised.

`log-level` (default: `'DEBUG`)
: Log level to use, specify a log level like `'INFO` to stop logging individual requests.

`cached-compressions` (default: `'()`)
: Activate generating cached nars with different compression details from the stored nars. This is a list of nar-herder-cached-compression-configuration records.

`min-uses` (default: `3`)
: When cached-compressions are enabled, generate cached nars when at least this number of requests are made for a nar.

`workers` (default: `2`)
: Number of cached nars to generate at a time.

`nar-source` (default: `#f`)
: Location to fetch nars from when computing cached compressions. By default, the storage location will be used.

`extra-environment-variables` (default: `'()`)
: Extra environment variables to set via the shepherd service.

`nar-herder-cached-compression-configuration` [Data Type]

Data type representing the cached compression configuration.

`type` Type of compression to use, e.g. `'zstd`.

`workers` (predeterminado: `#f`)
: Level of the compression to use.

`directory` (default: `#f`)
: Location to store the cached nars. If unspecified, they will be stored in /var/cache/nar-herder/nar/TYPE.

`directory-max-size` (default: `#f`)
: Maximum size in bytes of the directory.

`unused-removal-duration` (default: `#f`)
: If a cached nar isn't used for `unused-removal-duration`, it will be scheduled for removal.

  *unused-removal-duration* must denote a duration: `5d` means 5 days, `1m` means 1 month, and so on.

`ttl` (predeterminado: `#f`)
: If specified this overrides the `ttl` used for narinfos when this cached compression is available.

`new-ttl` (default: `#f`)
: As with the `new-ttl` option for `nar-herder-configuration`, this value will override the `ttl` when used for narinfo requests.

### 11.10.35 Servicios de Linux

## Servicio Early OOM

Early OOM (`https://github.com/rfjakob/earlyoom`), también conocido como Earlyoom, es un daemon minimalista de gestión del llenado de la memoria[19] que se ejecuta en espacio de usuaria y proporciona una alternativa al gestor del propio núcleo con una respuesta más inmediata y más configurable. Es útil para prevenir que el sistema no responda cuando se queda sin memoria.

`earlyoom-service-type` [Variable]
: Tipo de servicio para el servicio `earlyoom`, el daemon Early OOM. Su valor debe ser un objeto `earlyoom-configuration`, descrito a continuación. El servicio se puede instanciar con su configuración predeterminada de esta manera:

```
(service earlyoom-service-type)
```

`earlyoom-configuration` [Tipo de datos]
: Esta es el registro de configuración para el servicio `earlyoom-service-type`.

  `earlyoom` (predeterminado: *earlyoom*)
  : El paquete Earlyoom usado.

  `minimum-available-memory` (predeterminado: `10`)
  : El límite inferior de memoria *disponible*, en porcentaje.

  `minimum-free-swap` (predeterminado: `10`)
  : El límite inferior de memoria de intercambio libre, en porcentaje.

  `prefer-regexp` (predeterminado: `#f`)
  : Una expresión regular (como cadena) que corresponda con los nombres de los procesos que preferiblemente deban pararse.

  `avoid-regexp` (predeterminado: `#f`)
  : Una expresión regular (como cadena) que corresponda con los nombres de los procesos que *no* deban pararse.

[19] NdT: Del inglés Out Of Memory.

`memory-report-interval` (predeterminado: `0`)
: Intervalo en segundos con el cual se imprime el informe de memoria. No está activo de manera predeterminada.

`ignore-positive-oom-score-adj?` (predeterminado: `#f`)
: Valor booleano que indica si se deben ignoran los ajustes positivos realizados en `/proc/*/oom_score_adj`.

`show-debug-messages?` (predeterminado: `#f`)
: Valor booleano que indica si los mensajes de depuración deben imprimirse. Los registros se almacenan en `/var/log/earlyoom.log`.

`send-notification-command` (predeterminada: `#f`)
: Puede usarse para proporcionar una orden personalizada para el envío de notificaciones.

## fstrim Service

The command `fstrim` can be used to discard (or *trim*) unused blocks on a mounted file system.

> **Aviso:** Running `fstrim` frequently, or even using `mount -o discard`, might negatively affect the lifetime of poor-quality SSD devices. For most desktop and server systems a sufficient trimming frequency is once a week. Note that not all devices support a queued trim, so each trim command incurs a performance penalty on whatever else might be trying to use the disk at the time.

`fstrim-service-type` [Variable]
: Type for a service that periodically runs `fstrim`, whose value must be an `<fstrim-configuration>` object. The service can be instantiated in its default configuration with:

```
(service fstrim-service-type)
```

`fstrim-configuration` [Data Type]
: Available `fstrim-configuration` fields are:

`package` (default: `util-linux`) (type: file-like)
: The package providing the `fstrim` command.

`schedule` (default: `"0 0 * * 0"`) (type: mcron-time)
: Schedule for launching `fstrim`. This can be a procedure, a list or a string. For additional information, see Section “Job specification” in *the mcron manual*. By default this is set to run weekly on Sunday at 00:00.

`listed-in` (default: `'("/etc/fstab" "/proc/self/mountinfo")`) (type: maybe-list-of-strings)
: List of files in fstab or kernel mountinfo format. All missing or empty files are silently ignored. The evaluation of the list *stops* after the first non-empty file. File systems with `X-fstrim.notrim` mount option in fstab are skipped.

`verbose?` (default: `#t`) (type: boolean)
: Verbose execution.

`quiet-unsupported?` (default: `#t`) (type: boolean)
: Suppress error messages if trim operation (ioctl) is unsupported.

`extra-arguments` (type: maybe-list-of-strings)
: Extra options to append to `fstrim` (run '`man fstrim`' for more information).

### Servicio de carga de módulos del núcleo

The kernel module loader service allows one to load loadable kernel modules at boot. This is especially useful for modules that don't autoload and need to be manually loaded, as is the case with `ddcci`.

`kernel-module-loader-service-type` [Variable]
: Tipo de servicio para la carga de módulos del núcleo durante el arranque con `modprobe`. Su valor debe ser una lista de cadenas que representan nombres de módulo. Por ejemplo, la carga de los controladores proporcionados por `ddcci-driver-linux` en modo de depuración proporcionando algunos parámetros para el módulo puede realizarse de la siguiente manera:

```
(use-modules (gnu) (gnu services))
(use-package-modules linux)
(use-service-modules linux)

(define ddcci-config
  (plain-file "ddcci.conf"
              "options ddcci dyndbg delay=120"))

(operating-system
  ...
  (services (cons* (service kernel-module-loader-service-type
                            '("ddcci" "ddcci_backlight"))
                   (simple-service 'ddcci-config etc-service-type
                                   (list `("modprobe.d/ddcci.conf"
                                           ,ddcci-config)))
                   %base-services))
  (kernel-loadable-modules (list ddcci-driver-linux)))
```

### Cachefilesd Service

The Cachefilesd service starts a daemon that caches network file system data locally. It is especially useful for NFS and AFS shares, where it reduces latencies for repeated access when reading files.

The daemon can be configured as follows:

```
(service cachefilesd-service-type
         (cachefilesd-configuration
           (cache-directory "/var/cache/fscache")))
```

`cachefilesd-service-type` [Variable]
: The service type for starting `cachefilesd`. The value for this service type is a `cachefilesd-configuration`, whose only required field is *cache-directory*.

`cachefilesd-configuration` [Data Type]

Available `cachefilesd-configuration` fields are:

`cachefilesd` (default: `cachefilesd`) (type: file-like)
The cachefilesd package to use.

`debug-output?` (default: `#f`) (type: boolean)
Print debugging output to stderr.

`use-syslog?` (default: `#t`) (type: boolean)
Log to syslog facility instead of stdout.

`scan?` (default: `#t`) (type: boolean)
Scan for cachable objects.

`cache-directory` (type: maybe-string)
Location of the cache directory.

`cache-name` (default: `"CacheFiles"`) (type: maybe-string)
Name of cache (keep unique).

`security-context` (type: maybe-string)
SELinux security context.

`pause-culling-for-block-percentage` (default: `7`) (type: maybe-non-negative-integer)
Pause culling when available blocks exceed this percentage.

`pause-culling-for-file-percentage` (default: `7`) (type: maybe-non-negative-integer)
Pause culling when available files exceed this percentage.

`resume-culling-for-block-percentage` (default: `5`) (type: maybe-non-negative-integer)
Start culling when available blocks drop below this percentage.

`resume-culling-for-file-percentage` (default: `5`) (type: maybe-non-negative-integer)
Start culling when available files drop below this percentage.

`pause-caching-for-block-percentage` (default: `1`) (type: maybe-non-negative-integer)
Pause further allocations when available blocks drop below this percentage.

`pause-caching-for-file-percentage` (default: `1`) (type: maybe-non-negative-integer)
Pause further allocations when available files drop below this percentage.

`log2-table-size` (default: `12`) (type: maybe-non-negative-integer)
Size of tables holding cullable objects in logarithm of base 2.

`cull?` (default: `#t`) (type: boolean)
Create free space by culling (consumes system load).

`trace-function-entry-in-kernel-module?` (default: `#f`) (type: boolean)
Trace function entry in the kernel module (for debugging).

`trace-function-exit-in-kernel-module?` (default: `#f`) (type: boolean)
Trace function exit in the kernel module (for debugging).

`trace-internal-checkpoints-in-kernel-module?` (default: `#f`) (type: boolean)
Trace internal checkpoints in the kernel module (for debugging).

## Rasdaemon Service

The Rasdaemon service provides a daemon which monitors platform RAS (Reliability, Availability, and Serviceability) reports from Linux kernel trace events, logging them to syslogd.

Reliability, Availability and Serviceability is a concept used on servers meant to measure their robustness.

**Relability** is the probability that a system will produce correct outputs:

- Generally measured as Mean Time Between Failures (MTBF), and
- Enhanced by features that help to avoid, detect and repair hardware faults

**Availability** is the probability that a system is operational at a given time:

- Generally measured as a percentage of downtime per a period of time, and
- Often uses mechanisms to detect and correct hardware faults in runtime.

**Serviceability** is the simplicity and speed with which a system can be repaired or maintained:

- Generally measured on Mean Time Between Repair (MTBR).

Among the monitoring measures, the most usual ones include:

- CPU – detect errors at instruction execution and at L1/L2/L3 caches;
- Memory – add error correction logic (ECC) to detect and correct errors;
- I/O – add CRC checksums for transferred data;
- Storage – RAID, journal file systems, checksums, Self-Monitoring, Analysis and Reporting Technology (SMART).

By monitoring the number of occurrences of error detections, it is possible to identify if the probability of hardware errors is increasing, and, on such case, do a preventive maintenance to replace a degraded component while those errors are correctable.

For detailed information about the types of error events gathered and how to make sense of them, see the kernel administrator's guide at `https://www.kernel.org/doc/html/latest/admin-guide/ras.html`.

`rasdaemon-service-type` [Variable]
Service type for the `rasdaemon` service. It accepts a `rasdaemon-configuration` object. Instantiating like

```
(service rasdaemon-service-type)
```

will load with a default configuration, which monitors all events and logs to syslogd.

`rasdaemon-configuration` [Data Type]
The data type representing the configuration of `rasdaemon`.

`record?` (default: `#f`)
A boolean indicating whether to record the events in an SQLite database. This provides a more structured access to the information contained in the log file. The database location is hard-coded to `/var/lib/rasdaemon/ras-mc_event.db`.

## Servicio de dispositivo Zram

El servicio de dispositivo Zram proporciona un dispositivo comprimido de intercambio en la memoria del sistema. La documentación del núcleo Linux contiene más información sobre dispositivos zram (`https://www.kernel.org/doc/html/latest/admin-guide/blockdev/zram.html`).

`zram-device-service-type` [Variable]
Este servicio crea el dispositivo de bloques zram, le proporciona el formato de la memoria de intercambio y lo activa. El valor del servicio es un registro `zram-device-configuration`.

`zram-device-configuration` [Tipo de datos]
Este tipo de datos representa la configuración del servicio zram-device.

`size` (predeterminado: `"1G"`)
Es la cantidad de espacio que desea proporcionar al dispositivo zram. Acepta una cadena y puede ser el número de bytes o usar un sufijo, como por ejemplo `"512M"` o `1024000`.

`compression-algorithm` (predeterminado: `'lzo`)
Algoritmo de compresión que desea usar. Es difícil enumerar todas las opciones de compresión posibles, pero las más habituales entre las implementadas por el núcleo Linux Libre usado por Guix incluyen `'lzo`, `'lz4` y `'zstd`.

`memory-limit` (predeterminado: `0`)
Cantidad máxima de memoria que el dispositivo zram puede usar. Proporcionar el valor '0' desactiva el límite. Mientras que generalmente se espera que la compresión tenga una relación con los datos sin comprimir de 2:1, es posible que se escriban datos en la memoria de intercambio que no se pueden comprimir más, y esta es una forma de limitar cuanta memoria puede usarse. Acepta una cadena y puede ser un número de bytes o un usar sufijo, por ejemplo `"2G"`.

`priority` (default `#f`)
This is the priority of the swap device created from the zram device. See Section 11.6 [Swap Space], page 265, for a description of swap priorities. You might want to set a specific priority for the zram device, otherwise it could end up not being used much for the reasons described there.

### 11.10.36 Servicios de Hurd

`hurd-console-service-type` [Variable]
Este servicio inicia el cliente de consola `VGA` en Hurd.

El valor de este servicio es un registro `hurd-console-configuration`.

`hurd-console-configuration` [Tipo de datos]
Este tipo de datos representa la configuración del servicio hurd-console-service.

`hurd` (predeterminado: *hurd*)
El paquete de Hurd usado.

`hurd-getty-service-type` [Variable]
Este servicio inicia una interfaz de teletipo ("tty") mediante el uso del programa `getty` de Hurd.

El valor del servicio es un registro `hurd-getty-configuration`.

`hurd-getty-configuration` [Tipo de datos]
Este tipo de datos representa la configuración del servicio hurd-getty-service.

`hurd` (predeterminado: *hurd*)
El paquete de Hurd usado.

`tty` El nombre de la consola en la que se ejecuta este Getty—por ejemplo, `"tty1"`.

`baud-rate` (predeterminado: `38400`)
Un entero que especifica la tasa de transmisión de interfaz de teletipo.

### 11.10.37 Servicios misceláneos

#### Servicio de huella dactilar

El módulo `(gnu services authentication)` proporciona un servicio DBus para leer e identificar huellas dactilares mediante un sensor de huellas.

`fprintd-service-type` [Variable]
El tipo de servicio para `fprintd`, que proporciona la capacidad de lectura de huellas dactilares.

```
(service fprintd-service-type)
```

#### Servicios de control del sistema

El módulo `(gnu services sysctl)` proporciona servicios para configurar parámetros del núcleo durante el arranque.

`sysctl-service-type` [Variable]
El tipo de servicio para `sysctl`, que modifica parámetros del núcleo bajo `/proc/sys`. Para activar el encaminamiento de tráfico IPv4 se puede instanciar de esta manera:

```
(service sysctl-service-type
         (sysctl-configuration
           (settings '(("net.ipv4.ip_forward" . "1")))))
```

Since `sysctl-service-type` is used in the default lists of services, `%base-services` and `%desktop-services`, you can use `modify-services` to change its configuration and add the kernel parameters that you want (see Section 11.19.3 [Referencia de servicios], page 652).

```
(modify-services %base-services
  (sysctl-service-type config =>
                       (sysctl-configuration
                         (settings (append '(("net.ipv4.ip_forward" . "1"))
                                           %default-sysctl-settings)))))
```

`sysctl-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `sysctl`.

`sysctl` (predeterminado: `(file-append procps "/sbin/sysctl")`)
El ejecutable `sysctl` usado.

`settings` (default: `%default-sysctl-settings`)
Una lista asociativa que especifica parámetros del núcleo y sus valores.

`%default-sysctl-settings` [Variable]

An association list specifying the default `sysctl` parameters on Guix System.

## Servicio del daemon de tarjetas inteligentes PC/SC

The `(gnu services security-token)` module provides the following service to run `pcscd`, the PC/SC Smart Card Daemon. `pcscd` is the daemon program for pcsc-lite and the MuscleCard framework. It is a resource manager that coordinates communications with smart card readers, smart cards and cryptographic tokens that are connected to the system.

`pcscd-service-type` [Variable]

Tipo de servicio para el servicio `pcscd`. Su valor debe ser un objeto `pcscd-configuration`. Puede instanciarlo de esta manera para ejecutar pcscd con la configuración predeterminada:

```
(service pcscd-service-type)
```

`pcscd-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de `pcscd`.

`pcsc-lite` (predeterminado: `pcsc-lite`)
El paquete pcsc-lite que proporciona pcscd.

`usb-drivers` (predeterminado: `(list ccid)`)
List of packages that provide USB drivers to pcscd. Drivers are expected to be under `pcsc/drivers` in the store directory of the package.

## LIRC Service

El módulo `(gnu services lirc)` proporciona el siguiente servicio.

`lirc-service-type` [Variable]

Type for a service that runs LIRC (`http://www.lirc.org`), a daemon that decodes infrared signals from remote controls.

The value for this service is a `<lirc-configuration>` object.

`lirc-configuration` [Data Type]
Data type representing the configuration of `lircd`.

`lirc` (default: `lirc`) (type: file-like)
Package object for `lirc`.

`device` (default: `#f`) (type: string)
`driver` (default: `#f`) (type: string)
`config-file` (default: `#f`) (type: string-or-file-like)
TODO. See `lircd` manual for details.

`extra-options` (default: `'()`) (type: list-of-string)
Additional command-line options to pass to `lircd`.

## SPICE Service

El módulo `(gnu services spice)` proporciona el siguiente servicio.

`spice-vdagent-service-type` [Variable]
Type of the service that runs VDAGENT (`https://www.spice-space.org`), a daemon that enables sharing the clipboard with a vm and setting the guest display resolution when the graphical console window resizes.

`spice-vdagent-configuration` [Data Type]
Data type representing the configuration of `spice-vdagent-service-type`.

`spice-vdagent` (default: `spice-vdagent`) (type: file-like)
Package object for VDAGENT.

## Servicio inputattach

El servicio inputattach (`https://linuxwacom.github.io/`) permite el uso de dispositivos de entrada como tabletas Wacom, pantallas táctiles, o joysticks con el servidor gráfico Xorg.

`inputattach-service-type` [Variable]
Tipo de un servicio que ejecuta `inputattach` con un dispositivo y reenvía los eventos que produzca.

`inputattach-configuration` [Tipo de datos]

`device-type` (predeterminado: `"wacom"`)
Tipo del dispositivo al que conectarse. Ejecute `inputattach --help`, del paquete `inputattach`, para ver la lista de tipos de dispositivo implementados.

`device` (predeterminado: `"/dev/ttyS0"`)
El nombre de archivo para la conexión al dispositivo.

`baud-rate` (predeterminado: `#f`)
Tasa de transmisión usada para las conexiones serie. Debe ser un número o `#f`.

`log-file` (predeterminado: `#f`)
Si es verdadero, debe ser el nombre de un archivo en el que registrar los mensajes.

## Servicio de diccionario

El módulo `(gnu services dict)` proporciona el servicio siguiente:

`dicod-service-type` [Variable]

Tipo de servicio que ejecuta el daemon `dicod`, una implementación del servidor DICT (see Section "Dicod" in *GNU Dico Manual*).

Puede añadir `open localhost` en su archivo `~/.dico` para hacer que `localhost` sea el servidor predeterminado de su cliente `dico` (see Section "Initialization File" in *GNU Dico Manual*).

> **Nota:** This service is also available for Guix Home, where it runs directly with your user privileges (see Section 13.3.16 [Miscellaneous Home Services], page 701).

`dicod-configuration` [Tipo de datos]

Tipo de datos que representa la configuración de dicod.

`dico` (predeterminado: *dico*)
: El objeto paquete del servidor de diccionario GNU Dico.

`interfaces` (predeterminada: *'("localhost")*)
: Es la lista de direcciones IP y puertos, y posiblemente nombres de archivo de sockets, en los que se debe escuchar (see Section "Server Settings" in *GNU Dico Manual*).

`handlers` (predeterminados: *'()*)
: Lista de objetos `<dicod-handler>` que identifican los controladores (instancias de módulos).

`databases` (predeterminada: *(list %dicod-database:gcide)*)
: Lista de objetos `<dicod-database>` que identifican los diccionarios proporcionados.

`dicod-handler` [Tipo de datos]

Tipo de datos que representa un controlador de diccionario (instancia de un módulo).

`name` Nombre del controlador (instancia de un módulo).

`module` (predeterminado: *#f*)
: Nombre del módulo del controlador de dicod (instancia). Si es `#f`, el módulo tiene el mismo nombre que el controlador. (see Section "Modules" in *GNU Dico Manual*).

`options` Lista de cadenas o expresiones-G que representan los parámetros al módulo de control

`dicod-database` [Tipo de datos]

Tipo de datos que representa una base de datos de diccionario.

`name` Nombre de la base de datos, será usada en las órdenes DICT.

`handler` Nombre del controlador de dicod (instancia de un módulo) usado por esta base de datos (see Section "Handlers" in *GNU Dico Manual*).

`complex?` (predeterminado: *#f*)
: Determina si se usará la configuración compleja. La configuración compleja necesita un objeto `<dicod-handler>`, que no es necesario en otro caso.

`options`
: Lista de cadenas o expresiones-g que representan los parámetros para la base de datos (see Section "Databases" in *GNU Dico Manual*).

`%dicod-database:gcide` [Variable]
: Un objeto `<dicod-service>` que ofrece el diccionario internacional colaborativo de inglés de GNU usando el paquete `gcide`.

The following is an example `dicod-service-type` configuration.

```
(service dicod-service-type
         (dicod-configuration
          (handlers (list
                     (dicod-handler
                      (name "wordnet")
                      (module "wordnet")
                      (options
                       (list #~(string-append "wnhome=" #$wordnet))))))
          (databases (list
                      (dicod-database
                       (name "wordnet")
                       (complex? #t)
                       (handler "wordnet"))
                      %dicod-database:gcide))))
```

## Servicio Docker

El módulo `(gnu services docker)` proporciona los siguientes servicios.

`docker-service-type` [Variable]
: Este es el tipo del servicio que ejecuta Docker (`https://www.docker.com`), un daemon que puede ejecutar empaquetados de aplicaciones (a los que a veces nos referimos como "contenedores") en entornos aislados.

`docker-configuration` [Tipo de datos]
: Este es el tipo de datos que representa la configuración de Docker y Containerd.

`docker` (default: `docker`)
: El paquete de daemon de Docker usado.

`docker-cli` (default: `docker-cli`)
: El paquete de cliente de Docker usado.

`containerd` (predeterminado: *containerd*)
: El paquete Containerd usado.

`proxy` (predeterminado: *docker-libnetwork-cmd-proxy*)
: La pasarela de espacio de usuario para red de Docker usada.

`enable-proxy?` (predeterminado: `#t`)
: Activa o desactiva el uso de la pasarela de espacio de usuario para red de Docker.

`debug?` (predeterminado: `#f`)
: Activa o desactiva la salida de depuración.

`enable-iptables?` (predeterminado: `#t`)
: Activa o desactiva la adición de reglas para iptables.

`environment-variables` (default: `'()`)
: List of environment variables to set for `dockerd`.

  This must be a list of strings where each string has the form '*key*`=`*value*' as in this example:

```
(list "LANGUAGE=eo:ca:eu"
      "TMPDIR=/tmp/dockerd")
```

`config-file` (type: maybe-file-like)
: JSON configuration file pass to `dockerd`.

`singularity-service-type` [Variable]
: Tipo de servicio que le permite ejecutar Singularity (`https://www.sylabs.io/singularity/`), una herramienta tipo-Docker para crear y ejecutar aplicaciones empaquetadas (también conocidas como "contenedores"). El valor para este servicio es el paquete de Singularity usado.

  El servicio no instala un daemon; en vez de ello, instala programas auxiliares con con el bit setuid de root (see Section 11.11 [Programas con setuid], page 620) de modo que usuarias sin privilegios puedan ejecutar `singularity run` y ordenes similares.

## OCI backed services

Should you wish to manage your Docker containers with the same consistent interface you use for your other Shepherd services, *oci-container-service-type* is the tool to use: given an Open Container Initiative (OCI) container image, it will run it in a Shepherd service. One example where this is useful: it lets you run services that are available as Docker/OCI images but not yet packaged for Guix.

`oci-container-service-type` [Variable]
: This is a thin wrapper around Docker's CLI that executes OCI images backed processes as Shepherd Services.

```
(service oci-container-service-type
         (list
          (oci-container-configuration
           (image
            (oci-image
             (repository "guile")
             (tag "3")
             (value (specifications->manifest '("guile")))
             (pack-options '(#:symlinks (("/bin/guile" -> "bin/guile"))
                             #:max-layers 2))))
```

```
                (entrypoint "/bin/guile")
                (command
                 '("-c" "(display \"hello!\n\")")))
               (oci-container-configuration
                (image "prom/prometheus")
                (network "host")
                (ports
                  '(("9000" . "9000")
                    ("9090" . "9090"))))
               (oci-container-configuration
                (image "grafana/grafana:10.0.1")
                (network "host")
                (ports
                  '(("3000" . "3000")))
                (volumes
                  '("/var/lib/grafana:/var/lib/grafana")))))
```

In this example two different Shepherd services are going be added to the system. Each `oci-container-configuration` record translates to a `docker run` invocation and its fields directly map to options. You can refer to the upstream (`https://docs.docker.com/engine/reference/commandline/run`), documentation for the semantics of each value. If the images are not found they will be pulled (`https://docs.docker.com/engine/reference/commandline/pull/`). The spawned services are going to be attached to the host network and are supposed to behave like other processes.

`oci-container-configuration` [Data Type]

Available `oci-container-configuration` fields are:

`user` (default: `"oci-container"`) (type: string)
: The user under whose authority docker commands will be run.

`group` (default: `"docker"`) (type: string)
: The group under whose authority docker commands will be run.

`command` (default: `'()`) (type: list-of-strings)
: Overwrite the default command (`CMD`) of the image.

`entrypoint` (default: `""`) (type: string)
: Overwrite the default entrypoint (`ENTRYPOINT`) of the image.

`host-environment` (default: `'()`) (type: list)
: Set environment variables in the host environment where `docker run` is invoked. This is especially useful to pass secrets from the host to the container without having them on the `docker run`'s command line: by setting the `MYSQL_PASSWORD` on the host and by passing `--env MYSQL_PASSWORD` through the `extra-arguments` field, it is possible to securely set values in the container environment. This field's value can be a list of pairs or strings, even mixed:

```
(list '(\"LANGUAGE\" . \"eo:ca:eu\")
      \"JAVA_HOME=/opt/java\")
```

Pair members can be strings, gexps or file-like objects. Strings are passed directly to `make-forkexec-constructor`.

`environment` (default: `'()`) (type: list)
Set environment variables. This can be a list of pairs or strings, even mixed:

```
(list '("LANGUAGE" . "eo:ca:eu")
      "JAVA_HOME=/opt/java")
```

Pair members can be strings, gexps or file-like objects. Strings are passed directly to the Docker CLI. You can refer to the upstream (`https://docs.docker.com/engine/reference/commandline/run/#env`) documentation for semantics.

`image` (type: string-or-oci-image)
The image used to build the container. It can be a string or an `oci-image` record. Strings are resolved by the Docker Engine, and follow the usual format `myregistry.local:5000/testing/test-image:tag`.

`provision` (default: `""`) (type: string)
Set the name of the provisioned Shepherd service.

`requirement` (default: `'()`) (type: list-of-symbols)
Set additional Shepherd services dependencies to the provisioned Shepherd service.

`network` (default: `""`) (type: string)
Set a Docker network for the spawned container.

`ports` (default: `'()`) (type: list)
Set the port or port ranges to expose from the spawned container. This can be a list of pairs or strings, even mixed:

```
(list '("8080" . "80")
      "10443:443")
```

Pair members can be strings, gexps or file-like objects. Strings are passed directly to the Docker CLI. You can refer to the upstream (`https://docs.docker.com/engine/reference/commandline/run/#publish`) documentation for semantics.

`volumes` (default: `'()`) (type: list)
Set volume mappings for the spawned container. This can be a list of pairs or strings, even mixed:

```
(list '("/root/data/grafana" . "/var/lib/grafana")
      "/gnu/store:/gnu/store")
```

Pair members can be strings, gexps or file-like objects. Strings are passed directly to the Docker CLI. You can refer to the upstream (`https://docs.docker.com/engine/reference/commandline/run/#volume`) documentation for semantics.

`container-user` (default: `""`) (type: string)
: Set the current user inside the spawned container. You can refer to the upstream (`https://docs.docker.com/engine/reference/run/#user`) documentation for semantics.

`workdir` (default: `""`) (type: string)
: Set the current working for the spawned Shepherd service. You can refer to the upstream (`https://docs.docker.com/engine/reference/run/#workdir`) documentation for semantics.

`extra-arguments` (default: `'()`) (type: list)
: A list of strings, gexps or file-like objects that will be directly passed to the `docker run` invokation.

`oci-image` [Data Type]

Available `oci-image` fields are:

`repository` (type: string)
: A string like `myregistry.local:5000/testing/test-image` that names the OCI image.

`tag` (default: `"latest"`) (type: string)
: A string representing the OCI image tag. Defaults to `latest`.

`value` (type: oci-lowerable-image)
: A `manifest` or `operating-system` record that will be lowered into an OCI compatible tarball. Otherwise this field's value can be a gexp or a file-like object that evaluates to an OCI compatible tarball.

`pack-options` (default: `'()`) (type: list)
: An optional set of keyword arguments that will be passed to the `docker-image` procedure from `guix scripts pack`. They can be used to replicate `guix pack` behavior:

```
(oci-image
  (repository "guile")
  (tag "3")
  (value
    (specifications->manifest '("guile")))
  (pack-options '(#:symlinks (("/bin/guile" -> "bin/guile"))
                  #:max-layers 2)))
```

  If the `value` field is an `operating-system` record, this field's value will be ignored.

`system` (default: `""`) (type: string)
: Attempt to build for a given system, e.g. "i686-linux"

`target` (default: `""`) (type: string)
: Attempt to cross-build for a given triple, e.g. "aarch64-linux-gnu"

`grafts?` (default: `#f`) (type: boolean)
: Whether to allow grafting or not in the pack build.

## Servicio Auditd

El módulo `(gnu services auditd)` proporciona el servicio siguiente:

`auditd-service-type` [Variable]

Este es el tipo del servicio que ejecuta auditd (`https://people.redhat.com/sgrubb/audit/`), un daemon que recolecta información relevante a la seguridad en su sistema.

Ejemplos de cosas que se pueden recolectar:

1. Acceso a archivos
2. Llamadas al sistema
3. Órdenes invocadas
4. Intentos fallidos de ingreso en el sistema
5. Activaciones de filtros en el cortafuegos
6. Accesos de red

Puede usarse `auditctl` del paquete `audit` para añadir o eliminar eventos a recolectar (hasta el siguiente reinicio). Para hacer permanente la recolección de dichos eventos, introduzca los parámetros de la línea de órdenes de auditctl en un archivo llamado `audit.rules` del directorio de configuración (véase a continuación). También se puede usar `aureport` del paquete `audit` para ver un informe de todos los eventos registrados. El daemon audit habitualmente registra los eventos en el archivo `/var/log/audit`.

`auditd-configuration` [Tipo de datos]

Este es el tipo de datos que representa la configuración de auditd.

`audit` (predeterminado: `audit`)
: El paquete audit usado.

`configuration-directory` (predeterminado: `%default-auditd-configuration-directory`)
: Directorio que contiene el archivo de configuración para el paquete audit, cuyo nombre debe ser `auditd.conf`, y de manera opcional reglas de audit que se instanciarán en el arranque.

## servicio R-Shiny

El módulo `(gnu services science)` proporciona el siguiente servicio.

`rshiny-service-type` [Variable]

Tipo de servicio usado para ejecutar una aplicación web creada con `r-shiny`. Este servicio proporciona el valor adecuado a la variable de entorno `R_LIBS_USER` y ejecuta el guión proporcionado para llamar a `runApp`.

`rshiny-configuration` [Tipo de datos]

Este es el tipo de datos que representa la configuración del rshiny.

`package` (predeterminado: `r-shiny`)
: El paquete usado.

`binary` (default `"rshiny"`)
: Nombre del binario o del guión en el directorio `paquete/bin/` ejecutado cuando se arranca este servicio.

  La manera habitual de crear este archivo es la siguiente:

```
...
(let* ((out       (assoc-ref %outputs "out"))
       (targetdir (string-append out "/share/" ,name))
       (app       (string-append out "/bin/" ,name))
       (Rbin      (search-input-file %build-inputs "/bin/Rscript")
  ;; ...
  (mkdir-p (string-append out "/bin"))
  (call-with-output-file app
    (lambda (port)
      (format port
"#!~a
library(shiny)
setwd(\"~a\")
runApp(launch.browser=0, port=4202)~%\n"
      Rbin targetdir))))
```

## Servicio Nix

El módulo `(gnu services nix)` proporciona el siguiente servicio.

`nix-service-type` [Variable]
: Tipo del servicio que ejecuta el daemon de construcción del gestor de paquetes Nix (`https://nixos.org/nix/`). Este es un ejemplo de cómo usarlo:

```
(use-modules (gnu))
(use-service-modules nix)
(use-package-modules package-management)

(operating-system
  ;; ...
  (packages (append (list nix)
                    %base-packages))

  (services (append (list (service nix-service-type))
                    %base-services)))
```

  Tras `guix system reconfigure`, configure Nix para su usuaria:

  - Añada un canal Nix y lance una actualización. Véase Nix channels (`https://wiki.nixos.org/wiki/Nix_channels`) for more information about the available channels. If you would like to use the unstable Nix channel you can do this by running:

```
$ nix-channel --add https://nixos.org/channels/nixpkgs-unstable
$ nix-channel --update
```

  - Create your Nix profile directory:

```
$ sudo mkdir -p /nix/var/nix/profiles/per-user/$USER
```

```
$ sudo chown $USER:root /nix/var/nix/profiles/per-user/$USER
```

- Cree un enlace simbólico a su perfil y active el perfil de Nix:

```
$ ln -s "/nix/var/nix/profiles/per-user/$USER/profile" ~/.nix-profile
$ source /run/current-system/profile/etc/profile.d/nix.sh
```

`nix-configuration` [Tipo de datos]

Este tipo de datos representa la configuración del daemon de Nix.

`nix` (predeterminado: `nix`)
: El paquete Nix usado.

`sandbox` (predeterminado: `#t`)
: Especifica si las construcciones se ejecutan en un entorno aislado ("sandbox") de manera predeterminada.

`build-directory` (default: `"/tmp"`)
: The directory where build directory are stored during builds. This is useful to change if, for example, the default location does not have enough space to hold build trees for big packages.

  This is similar to setting the `TMPDIR` environment variable for `guix-daemon`. Section 2.2.1 [Configuración del entorno de construcción], page 6, for more info.

`build-sandbox-items` (predeterminada: `'()`)
: Lista de cadenas u objetos añadida al final del campo `build-sandbox-items` en el archivo de configuración.

`extra-config` (predeterminada: `'()`)
: Es una lista de cadenas u objetos añadida al final del archivo de configuración. Se usa para proporcionar texto adicional para ser introducido de forma literal en el archivo de configuración.

`extra-options` (predeterminadas: `'()`)
: Opciones de línea de órdenes adicionales para `nix-service-type`.

## Fail2Ban service

`fail2ban` (`http://www.fail2ban.org/`) scans log files (e.g. `/var/log/apache/error_log`) and bans IP addresses that show malicious signs – repeated password failures, attempts to make use of exploits, etc.

`fail2ban-service-type` service type is provided by the `(gnu services security)` module.

This service type runs the `fail2ban` daemon. It can be configured in various ways, which are:

Basic configuration
: The basic parameters of the Fail2Ban service can be configured via its `fail2ban` configuration, which is documented below.

User-specified jail extensions
: The `fail2ban-jail-service` function can be used to add new Fail2Ban jails.

Shepherd extension mechanism
: Service developers can extend the `fail2ban-service-type` service type itself via the usual service extension mechanism.

`fail2ban-service-type` [Variable]
: This is the type of the service that runs `fail2ban` daemon. Below is an example of a basic, explicit configuration:

```
(append
 (list
  (service fail2ban-service-type
           (fail2ban-configuration
            (extra-jails
             (list
              (fail2ban-jail-configuration
               (name "sshd")
               (enabled? #t))))))
  ;; There is no implicit dependency on an actual SSH
  ;; service, so you need to provide one.
  (service openssh-service-type))
 %base-services)
```

`fail2ban-jail-service` *svc-type jail* [Procedure]
: Extend *svc-type*, a `<service-type>` object with *jail*, a `fail2ban-jail-configuration` object.

  Por ejemplo:

```
(append
 (list
  (service
   ;; The 'fail2ban-jail-service' procedure can extend any service type
   ;; with a fail2ban jail.  This removes the requirement to explicitly
   ;; extend services with fail2ban-service-type.
   (fail2ban-jail-service
    openssh-service-type
    (fail2ban-jail-configuration
     (name "sshd")
     (enabled? #t)))
   (openssh-configuration ...))))
```

Below is the reference for the different `jail-service-type` configuration records.

`fail2ban-configuration` [Data Type]
: Available `fail2ban-configuration` fields are:

  `fail2ban` (default: `fail2ban`) (type: package)
  : The `fail2ban` package to use. It is used for both binaries and as base default configuration that is to be extended with `<fail2ban-jail-configuration>` objects.

`run-directory` (default: `"/var/run/fail2ban"`) (type: string)
The state directory for the `fail2ban` daemon.

`jails` (default: `'()`) (type: list-of-fail2ban-jail-configurations)
Instances of `<fail2ban-jail-configuration>` collected from extensions.

`extra-jails` (default: `'()`) (type: list-of-fail2ban-jail-configurations)
Instances of `<fail2ban-jail-configuration>` explicitly provided.

`extra-content` (default: `'()`) (type: text-config)
Extra raw content to add to the end of the `jail.local` file, provided as a list of file-like objects.

`fail2ban-ignore-cache-configuration` [Data Type]
Available `fail2ban-ignore-cache-configuration` fields are:

`key` (type: string)
Cache key.

`max-count` (type: integer)
Cache size.

`max-time` (type: integer)
Cache time.

`fail2ban-jail-action-configuration` [Data Type]
Available `fail2ban-jail-action-configuration` fields are:

`name` (type: string)
Action name.

`arguments` (default: `'()`) (type: list-of-arguments)
Action arguments.

`fail2ban-jail-configuration` [Data Type]
Available `fail2ban-jail-configuration` fields are:

`name` (type: string)
Required name of this jail configuration.

`enabled?` (default: `#t`) (type: boolean)
Whether this jail is enabled.

`backend` (type: maybe-symbol)
Backend to use to detect changes in the `log-path`. The default is 'auto. To consult the defaults of the jail configuration, refer to the `/etc/fail2ban/jail.conf` file of the `fail2ban` package.

`max-retry` (type: maybe-integer)
The number of failures before a host gets banned (e.g. `(max-retry 5)`).

`max-matches` (type: maybe-integer)
The number of matches stored in ticket (resolvable via tag `<matches>`) in action.

`find-time` (type: maybe-string)
The time window during which the maximum retry count must be reached for an IP address to be banned. A host is banned if it has generated `max-retry` during the last `find-time` seconds (e.g. `(find-time "10m")`). It can be provided in seconds or using Fail2Ban's "time abbreviation format", as described in `man 5 jail.conf`.

`ban-time` (type: maybe-string)
The duration, in seconds or time abbreviated format, that a ban should last. (e.g. `(ban-time "10m")`).

`ban-time-increment?` (type: maybe-boolean)
Whether to consider past bans to compute increases to the default ban time of a specific IP address.

`ban-time-factor` (type: maybe-string)
The coefficient to use to compute an exponentially growing ban time.

`ban-time-formula` (type: maybe-string)
This is the formula used to calculate the next value of a ban time.

`ban-time-multipliers` (type: maybe-string)
Used to calculate next value of ban time instead of formula.

`ban-time-max-time` (type: maybe-string)
The maximum number of seconds a ban should last.

`ban-time-rnd-time` (type: maybe-string)
The maximum number of seconds a randomized ban time should last. This can be useful to stop "clever" botnets calculating the exact time an IP address can be unbanned again.

`ban-time-overall-jails?` (type: maybe-boolean)
When true, it specifies the search of an IP address in the database should be made across all jails. Otherwise, only the current jail of the ban IP address is considered.

`ignore-self?` (type: maybe-boolean)
Never ban the local machine's own IP address.

`ignore-ip` (default: `'()`) (type: list-of-strings)
A list of IP addresses, CIDR masks or DNS hosts to ignore. `fail2ban` will not ban a host which matches an address in this list.

`ignore-cache` (type: maybe-fail2ban-ignore-cache-configuration)
Provide cache parameters for the ignore failure check.

`filter` (type: maybe-fail2ban-jail-filter-configuration)
The filter to use by the jail, specified via a `<fail2ban-jail-filter-configuration>` object. By default, jails have names matching their filter name.

`log-time-zone` (type: maybe-string)
The default time zone for log lines that do not have one.

`log-encoding` (type: maybe-symbol)
: The encoding of the log files handled by the jail. Possible values are: `'ascii`, `'utf-8` and `'auto`.

`log-path` (default: `'()`) (type: list-of-strings)
: The file names of the log files to be monitored.

`action` (default: `'()`) (type: list-of-fail2ban-jail-actions)
: A list of `<fail2ban-jail-action-configuration>`.

`extra-content` (default: `'()`) (type: text-config)
: Extra content for the jail configuration, provided as a list of file-like objects.

`fail2ban-jail-filter-configuration` [Data Type]

Available `fail2ban-jail-filter-configuration` fields are:

`name` (type: string)
: Filter to use.

`mode` (type: maybe-string)
: Mode for filter.

## Backup Services

The `(gnu services backup)` module offers services for backing up file system trees. For now, it provides the `restic-backup-service-type`.

With `restic-backup-service-type`, you can periodically back up directories and files with Restic (`https://restic.net/`), which supports end-to-end encryption and deduplication. Consider the following configuration:

```
(use-service-modules backup ...) ;for 'restic-backup-service-type'
(use-package-modules sync ...)   ;for 'rclone'

(operating-system
  ;; ...
  (packages (append (list rclone)    ;for use by restic
                    %base-packages))
  (services
    (list
      (service restic-backup-service-type
               (restic-backup-configuration
                 (jobs
                   (list (restic-backup-job
                           (name "remote-ftp")
                           (repository "rclone:remote-ftp:backup/restic")
                           (password-file "/root/.restic")
                           ;; Every day at 23.
                           (schedule "0 23 * * *")
                           (files '("/root/.restic"
                                    "/root/.config/rclone"
```

```
                                "/etc/ssh/ssh_host_rsa_key"
                                "/etc/ssh/ssh_host_rsa_key.pub"
                                "/etc/guix/signing-key.pub"
                                "/etc/guix/signing-key.sec")))))))))
```

Each `restic-backup-job` translates to an mcron job which sets the `RESTIC_PASSWORD` environment variable by reading the first line of `password-file` and runs `restic backup`, creating backups using rclone of all the files listed in the `files` field.

The `restic-backup-service-type` installs as well `restic-guix` to the system profile, a `restic` utility wrapper that allows for easier interaction with the Guix configured backup jobs. For example the following could be used to instantaneusly trigger a backup for the above shown configuration, without waiting for the scheduled job:

```
restic-guix backup remote-ftp
```

`restic-backup-configuration` [Data Type]

Available `restic-backup-configuration` fields are:

`jobs` (default: `'()`) (type: list-of-restic-backup-jobs)
The list of backup jobs for the current system.

`restic-backup-job` [Data Type]

Available `restic-backup-job` fields are:

`restic` (default: `restic`) (type: package)
The restic package to be used for the current job.

`user` (default: `"root"`) (type: string)
The user used for running the current job.

`repository` (type: string)
The restic repository target of this job.

`name` (type: string)
A string denoting a name for this job.

`password-file` (type: string)
Name of the password file, readable by the configured `user`, that will be used to set the `RESTIC_PASSWORD` environment variable for the current job.

`schedule` (type: gexp-or-string)
A string or a gexp that will be passed as time specification in the mcron job specification (see Section "Syntax" in *GNU mcron*).

`files` (default: `'()`) (type: list-of-lowerables)
The list of files or directories to be backed up. It must be a list of values that can be lowered to strings.

`verbose?` (default: `#f`) (type: boolean)
Whether to enable verbose output for the current backup job.

`extra-flags` (default: `'()`) (type: list-of-lowerables)
A list of values that are lowered to strings. These will be passed as command-line arguments to the current job `restic backup` invokation.

## 11.11 Programas con setuid

Some programs need to run with elevated privileges, even when they are launched by unprivileged users. A notorious example is the `passwd` program, which users can run to change their password, and which needs to access the `/etc/passwd` and `/etc/shadow` files—something normally restricted to root, for obvious security reasons. To address that, `passwd` should be *setuid-root*, meaning that it always runs with root privileges (see Section "How Change Persona" in *The GNU C Library Reference Manual*, for more info about the setuid mechanism).

The store itself *cannot* contain setuid programs: that would be a security issue since any user on the system can write derivations that populate the store (see Section 8.9 [El almacén], page 157). Thus, a different mechanism is used: instead of changing the setuid or setgid bits directly on files that are in the store, we let the system administrator *declare* which programs should be entrusted with these additional privileges.

The `setuid-programs` field of an `operating-system` declaration contains a list of `<setuid-program>` denoting the names of programs to have a setuid or setgid bit set (see Section 11.2 [Uso de la configuración del sistema], page 244). For instance, the `mount.nfs` program, which is part of the nfs-utils package, with a setuid root can be designated like this:

```
(setuid-program
  (program (file-append nfs-utils "/sbin/mount.nfs")))
```

And then, to make `mount.nfs` setuid on your system, add the previous example to your operating system declaration by appending it to `%setuid-programs` like this:

```
(operating-system
  ;; Some fields omitted...
  (setuid-programs
    (append (list (setuid-program
                    (program (file-append nfs-utils "/sbin/mount.nfs"))))
            %setuid-programs)))
```

`setuid-program` [Data Type]

This data type represents a program with a setuid or setgid bit set.

`program` A file-like object having its setuid and/or setgid bit set.

`setuid?` (default: `#t`)
: Whether to set user setuid bit.

`setgid?` (default: `#f`)
: Whether to set group setgid bit.

`user` (default: `0`)
: UID (integer) or user name (string) for the user owner of the program, defaults to root.

`group` (default: `0`)
: GID (integer) group name (string) for the group owner of the program, defaults to root.

Un conjunto predeterminado de programas con el bit setuid se define en la variable `%setuid-programs` del módulo `(gnu system)`.

`%setuid-programs` [Variable]

A list of `<setuid-program>` denoting common programs that are setuid-root.

La lista incluye órdenes como `passwd`, `ping`, `su` y `sudo`.

Para su implementación, los programas con setuid reales se crean en el directorio `/run/setuid-programs` durante la activación del sistema. Los archivos en este directorio hacen referencia a los binarios "reales", que están en el almacén.

## 11.12 Certificados X.509

En las conexiones HTTPS a servidores Web (esto es, HTTP sobre el mecanismo de seguridad de la capa de transporte, TLS) se envía a los programas clientes un *certificado X.509* que el cliente puede usar para *autentificar* al servidor. Para hacerlo, los clientes verifican que el certificado del servidor está firmado por una de las llamadas *autoridades de certificación* (AC, CA en inglés). Pero para verificar la firma de una AC, los clientes deben haber obtenido previamente el certificado de dicha AC.

Los navegadores Web como GNU IceCat incluyen su propio conjunto de certificados de AC, de manera que pueden verificar las firmas independientemente.

No obstante, a la mayor parte de otros programas que pueden comunicarse a través de HTTPS—`wget`, `git`, `w3m`, etc.—se les debe informar de dónde pueden encontrar los certificados de CA.

For users of Guix System, this is done by adding a package that provides certificates to the `packages` field of the `operating-system` declaration (see Section 11.3 [Referencia de operating-system], page 253). Guix includes one such package, `nss-certs`, which is a set of CA certificates provided as part of Mozilla's Network Security Services.

This package is part of `%base-packages`, so there is no need to explicitly add it. The `/etc/ssl/certs` directory, which is where most applications and libraries look for certificates by default, points to the certificates installed globally.

Las usuarias sin privilegios, incluyendo a usuarias de Guix en una distribución distinta, pueden también instalar su propio paquete de certificados en su perfil. Es necesario definir cierto número de variables de entorno de manera que las aplicaciones y bibliotecas sepan dónde encontrarlos. Por ejemplo, la biblioteca OpenSSL inspecciona las variables `SSL_CERT_DIR` y `SSL_CERT_FILE`. Algunas aplicaciones añaden sus variables de entorno propias; por ejemplo, el sistema de control de versiones Git inspecciona el empaquetado de certificados al que apunta la variable de entorno `GIT_SSL_CAINFO`. Por tanto, en el caso típico se debe ejecutar algo parecido a esto:

```
guix install nss-certs
export SSL_CERT_DIR="$HOME/.guix-profile/etc/ssl/certs"
export SSL_CERT_FILE="$HOME/.guix-profile/etc/ssl/certs/ca-certificates.crt"
export GIT_SSL_CAINFO="$SSL_CERT_FILE"
```

Como otro ejemplo, R necesita que la variable de entorno `CURL_CA_BUNDLE` apunte al empaquetado de certificados, de manera que se debe ejecutar algo parecido a esto:

```
guix install nss-certs
```

```
export CURL_CA_BUNDLE="$HOME/.guix-profile/etc/ssl/certs/ca-certificates.crt"
```

Para otras aplicaciones puede tener que buscar la variable de entorno necesaria en la documentación relevante.

## 11.13 Selector de servicios de nombres

El módulo `(gnu system nss)` proporciona una interfaz con el archivo de configuración del *selector de servicios de nombres* o *NSS* (see Section "NSS Configuration File" in *The GNU C Library Reference Manual*). En resumen, NSS es un mecanismo que permite la extensión de libc con nuevos métodos de búsqueda de "nombres", lo que incluye nombres de máquinas, nombres de servicios, cuentas de usuaria y más (see Section "Name Service Switch" in *The GNU C Library Reference Manual*).

La configuración de NSS especifica, para cada base de datos del sistema, que método de búsqueda debe ser usado, y cómo los varios métodos se enlazan entre sí—por ejemplo, bajo qué circunstancias NSS deberá probar con el siguiente método en la lista. La configuración de NSS se proporciona en el campo `name-service-switch` de las declaraciones `operating-system` (see Section 11.3 [Referencia de operating-system], page 253).

Como ejemplo, la siguiente declaración configura NSS para que use el motor `nss-mdns` (`https://0pointer.de/lennart/projects/nss-mdns/`), que permite las búsquedas de nombres de máquinas sobre DNS multicast (mDNS) para nombres de máquinas terminados en `.local`:

```
(name-service-switch
   (hosts (list %files    ;primero, comprueba /etc/hosts

                ;; Si lo anterior no funcionó, prueba
                ;; con 'mdns_minimal'.
                (name-service
                  (name "mdns_minimal")

                  ;; 'mdns_minimal' tiene autoridad sobre
                  ;; '.local'.  Cuando devuelve 'not-found,
                  ;; no es necesario intentarlo con los
                  ;; métodos siguientes.
                  (reaction (lookup-specification
                             (not-found => return))))

                ;; Si no, usa DNS.
                (name-service
                  (name "dns"))

                ;; Finalmente, prueba con 'mdns' "al completo".
                (name-service
                  (name "mdns")))))
```

No se preocupe: la variable `%mdns-host-lookup-nss` (véase a continuación) contiene esta configuración, de manera que no tiene que escribirla si todo lo que desea es que funcione la búsqueda de nombres de máquina en `.local`.

Fíjese que, en este caso, además de establecer el valor de `name-service-switch` en la declaración `operating-system`, es necesario también usar el servicio `avahi-service-type` (see Section 11.10.5 [Servicios de red], page 314) o `%desktop-services`, donde está incluido. Esto permite el acceso a `nss-mdsn` desde el daemon de la caché del servicio de nombres (see Section 11.10.1 [Servicios base], page 276).

Por conveniencia, las siguientes variables proporcionan configuraciones NSS típicas.

`%default-nss` [Variable]
Esta es la configuración predeterminada del selector de servicios de nombres, un objeto `name-service-switch`.

`%mdns-host-lookup-nss` [Variable]
Esta es la configuración del selector de servicios de nombres que permite la búsqueda de nombres de máquinas por DNS multicast (mDNS) para nombres de máquinas terminados en `.local`.

La referencia de la configuración del selector de servicios de nombres se proporciona a continuación. Tiene una asociación directa con el formato del archivo de configuración de la biblioteca C, por lo que se recomienda el manual de la biblioteca C para obtener más información (see Section "NSS Configuration File" in *The GNU C Library Reference Manual*). En comparación con el formato del archivo de configuración del NSS de libc, no solo tiene solo la ventaja de la cálida sensación proporcionada por la adición de paréntesis que tanto nos gustan, sino que también tiene comprobaciones estáticas: conocerá los errores sintácticos y tipográficos con la ejecución de `guix system`.

`name-service-switch` [Tipo de datos]
El tipo de datos que representa la configuración del selector de servicios de nombres (NSS). Cada campo a continuación representa una de las bases de datos del sistema admitidas.

`aliases`
`ethers`
`group`
`gshadow`
`hosts`
`initgroups`
`netgroup`
`networks`
`password`
`public-key`
`rpc`
`services`
`shadow` The system databases handled by the NSS. Each of these fields must be a list of `<name-service>` objects (see below).

`name-service` [Tipo de datos]
Este es el tipo de datos que representa un servicio de nombres real y la acción de búsqueda asociada.

`name` Una cadena que denota el nombre de servicio (see Section "Services in the NSS configuration" in *The GNU C Library Reference Manual*).

Fíjese que los servicios de nombres enumerados aquí deben ser visibles para nscd. Esto se consigue mediante la adición del parámetro `#:name-services` a `nscd-service` con la lista de paquetes que proporcionan los servicios de nombres necesarios (see Section 11.10.1 [Servicios base], page 276).

`reaction` Una acción especificada mediante el uso del macro `lookup-specification` (see Section "Actions in the NSS configuration" in *The GNU C Library Reference Manual*). Por ejemplo:

```
(lookup-specification (unavailable => continue)
                      (success => return))
```

## 11.14 Disco en RAM inicial

Para el propósito del arranque inicial, se le proporciona al núcleo Linux-libre un *disco inicial en RAM*, o *initrd*. Un initrd contiene un sistema de archivos raíz temporal así como un guión de inicialización. Este último es responsable del montaje del sistema de archivos raíz real, así como de la carga de cualquier módulo del núcleo que pueda ser necesario para esta tarea.

El campo `initrd-modules` de una declaración `operating-system` le permite especificar qué módulos del núcleo Linux-libre deben estar disponibles en el initrd. En particular, aquí es donde se debe enumerar los módulos que controlen realmente el disco duro donde su partición raíz se encuentre—aunque el valor predeterminado de `initrd-modules` debería cubrir la mayor parte de casos de uso. Por ejemplo, en caso de necesitar el módulo `megaraid_sas` además de los módulos predeterminados para poder acceder a sistema de archivos raíz, se podría escribir:

```
(operating-system
  ;; ...
  (initrd-modules (cons "megaraid_sas" %base-initrd-modules)))
```

`%base-initrd-modules` [Variable]

Esta es la lista de módulos del núcleo que se incluyen en el initrd predeterminado.

Más allá, si necesita personalizaciones de un nivel más bajo, el campo `initrd` de una declaración `operating-system` le permite especificar qué initrd desea usar. El módulo `(gnu system linux-initrd)` proporciona tres formas de construir un initrd: el procedimiento de alto nivel `base-initrd` y los procedimientos de bajo nivel `raw-initrd` y `expression->initrd`.

El procedimiento `base-initrd` está pensado para cubrir la mayor parte de usos comunes. Por ejemplo, si desea añadir algunos módulos del núcleo que deben cargarse durante el arranque, puede definir el campo `initrd` de la declaración de sistema operativo de esta forma:

```
(initrd (lambda (sistemas-de-archivos . resto)
          ;; Crea un initrd estándar pero configura la red
          ;; con los parámetros que QEMU espera por omisión.
```

```
(apply base-initrd sistemas-de-archivos
       #:qemu-networking? #t
       resto)))
```

El procedimiento `base-initrd` también maneja casos de uso comunes que implican el uso del sistema en un anfitrión QEMU, o como un sistema "live" con un sistema de archivos raíz volátil.

The `base-initrd` procedure is built from `raw-initrd` procedure. Unlike `base-initrd`, `raw-initrd` doesn't do anything high-level, such as trying to guess which kernel modules and packages should be included to the initrd. An example use of `raw-initrd` is when a user has a custom Linux kernel configuration and default kernel modules included by `base-initrd` are not available.

El disco inicial en RAM producido por `base-initrd` o `raw-initrd` inspecciona varias opciones proporcionadas por la línea de órdenes al núcleo Linux (esto es, argumentos pasados a través de la orden `linux` de GRUB, o de la opción `-append` de QEMU), notablemente:

`gnu.load=`*boot*
: Indica al disco de RAM inicial que cargue *arranque*, un archivo que contiene un programa Scheme, una vez haya montado el sistema de archivos raíz.

  Guix usa esta opción para proporcionar el control a un programa de arranque que ejecuta los programas de activación de servicios y lanza GNU Shepherd, el sistema de inicialización.

`root=`*root*
: Monta *raíz* como el sistema de archivos raíz. *raíz* puede ser un nombre de dispositivo como `/dev/sda1`, una etiqueta del sistema de archivos o un UUID del sistema de archivos. Si no se proporciona se usa el nombre del sistema de archivos raíz de la declaración del sistema operativo.

`rootfstype=`*type*
: Set the type of the root file system. It overrides the `type` field of the root file system specified via the `operating-system` declaration, if any.

`rootflags=`*options*
: Set the mount *options* of the root file system. It overrides the `options` field of the root file system specified via the `operating-system` declaration, if any.

`fsck.mode=`*mode*
: Whether to check the *root* file system for errors before mounting it. *mode* is one of `skip` (never check), `force` (always check), or `auto` to respect the root `<file-system>` object's `check?` setting (see Section 11.4 [Sistemas de archivos], page 257) and run a full scan only if the file system was not cleanly shut down.

  `auto` is the default if this option is not present or if *mode* is not one of the above.

`fsck.repair=`*level*
: The level of repairs to perform automatically if errors are found in the *root* file system. *level* is one of `no` (do not write to *root* at all if possible), `yes` (repair as much as possible), or `preen` to repair problems considered safe to repair automatically.

`preen` is the default if this option is not present or if *level* is not one of the above.

`gnu.system=`*system*
: Hace que `/run/booted-system` y `/run/current-system` apunten a *sistema*.

`modprobe.blacklist=`*módulos...*
: Indica al disco inicial en RAM así como a la orden `modprobe` (del paquete kmod) que deben negarse a cargar *módulos*. *módulos* debe ser una lista separada por comas de nombres de módulos—por ejemplo, `usbkbd,9pnet`.

`gnu.repl`
: Inicia una sesión interactiva (REPL) desde el disco inicial en RAM antes de que intente cargar los módulos del núcleo y del montaje del sistema de archivos raíz. Nuestro departamento comercial lo llama *arranca-en-Guile*. Como amante de Scheme, lo adorará. See Section "Using Guile Interactively" in *GNU Guile Reference Manual*, para más información sobre sesiones interactivas Guile.

Una vez conocidas todas las características que proporcionan los discos iniciales en RAM que producen `base-initrd` y `raw-initrd`, a continuación veremos cómo usarlas y personalizarlos más aún.

`raw-initrd` *sistemas-de-archivos* [*#:linux-modules '()*] [Procedimiento]
[*#:pre-mount #t*] [*#:mapped-devices '()*] [*#:keyboard-layout #f*]
[*#:helper-packages '()*] [*#:qemu-networking? #f*]
[#:volatile-root? #f] Return a derivation that builds a raw initrd. *file-systems* is a list of file systems to be mounted by the initrd, possibly in addition to the root file system specified on the kernel command line via `root`. *linux-modules* is a list of kernel modules to be loaded at boot time. *mapped-devices* is a list of device mappings to realize before *file-systems* are mounted (see Section 11.5 [Dispositivos traducidos], page 263). *pre-mount* is a G-expression to evaluate before realizing *mapped-devices*. *helper-packages* is a list of packages to be copied in the initrd. It may include `e2fsck/static` or other packages needed by the initrd to check the root file system.

Cuando su valor es verdadero, *keyboard-layout* es un registro `<keyboard-layout>` que denota la distribución de teclado en consola deseada. Esto se realiza previamente a que los dispositivos configurados en *mapped-devices* se inicien y antes de que los sistemas de archivos en *file-systems* se monten, de manera que, en caso de que la usuaria tuviese que introducir una contraseña o usar la sesión interactiva, esto suceda usando la distribución de teclado deseada.

Cuando *qemu-networking?* es verdadero, configura la red con los parámetros QEMU estándar. Cuando *virtio?* es verdadero, carga módulos adicionales para que la imagen en RAM pueda ser usada como un sistema virtualizado por QEMU con controladores paravirtualizados de E/S.

Cuando *volatile-root?* es verdadero, el sistema de archivos raíz tiene permisos de escritura pero cualquier cambio realizado se perderá.

`base-initrd` *sistemas-de-archivos* [*#:mapped-devices '()*] [Procedimiento]
[*#:keyboard-layout #f*] [*#:qemu-networking? #f*]
[#:volatile-root? #f] [#:linux-modules '()] Devuelve como un objeto tipo-archivo una imagen de inicio en RAM genérica, con los módulos del núcleo tomados de *linux*.

*sistemas-de-archivos* es una lista de sistemas de archivos listos para ser montados por la imagen, posiblemente en adición al sistema de archivos raíz especificado en la línea de órdenes del núcleo via `root`. *mapped-devices* es una lista de asociación de dispositivos a realizar antes de montar los *sistemas-de-archivos*.

Cuando su valor es verdadero, *keyboard-layout* es un registro `<keyboard-layout>` que denota la distribución de teclado en consola deseada. Esto se realiza previamente a que los dispositivos configurados en *mapped-devices* se inicien y antes de que los sistemas de archivos en *file-systems* se monten, de manera que, en caso de que la usuaria tuviese que introducir una contraseña o usar la sesión interactiva, esto suceda usando la distribución de teclado deseada.

*qemu-networking?* y *volatile-root?* funcionan como en `raw-initrd`.

El initrd incorpora automáticamente todos los módulos del núcleo necesarios para *sistemas-de-archivos* y para las opciones proporcionadas. Módulos del núcleo adicionales pueden proporcionarse a través de *linux-modules*. Se añadirán al initrd y se cargarán en tiempo de arranque en el orden que aparezcan.

No es necesario decir que los initrd que producimos y usamos embeben un Guile enlazado estáticamente, y que el programa de inicialización es un programa Guile. Esto proporciona mucha flexibilidad. El procedimiento `expression->initrd` construye un initrd de ese tipo, una vez proporcionado el programa a ejecutar en dicho initrd.

`expression->initrd` *exp* [*#:guile %guile-static-initrd*] [*#:name* "*guile-initrd*"] *Return as a file-like* [Procedimiento]
object a Linux initrd (a gzipped cpio archive) containing *guile* and that evaluates *exp*, a G-expression, upon booting. All the derivations referenced by *exp* are automatically copied to the initrd.

## 11.15 Configuración del gestor de arranque

El sistema operativo permite varios cargadores de arranque. El cargador de arranque se configura mediante el uso de la declaración `bootloader-configuration`. Todos los campos de esta estructura son independientes del cargador de arranque excepto uno, `bootloader`, que indica el cargador de arranque a configurar e instalar.

Algunos de los cargadores de arranque no inspeccionan todos los campos de `bootloader-configuration`. Por ejemplo, el cargador de arranque extlinux no permite temas y por lo tanto ignora el campo `theme`.

`bootloader-configuration` [Tipo de datos]
El tipo de una declaración de configuración del cargador de arranque.

`bootloader`
The bootloader to use, as a `bootloader` object. For now `grub-bootloader`, `grub-efi-bootloader`, `grub-efi-removable-bootloader`, `grub-efi-netboot-bootloader`, `grub-efi-netboot-removable-bootloader`, `extlinux-bootloader` and `u-boot-bootloader` are supported.

Los cargadores de arranque se describen en los módulos `(gnu bootloader ...)`. En particular, `(gnu bootloader u-boot)` contiene definiciones

de cargadores de arranque para un amplio rango de sistemas ARM y AArch64, mediante el uso del cargador de arranque U-Boot (`https://www.denx.de/wiki/U-Boot/`).

`grub-bootloader` permite el arranque en máquinas basadas en Intel en modo "antiguo" BIOS.

`grub-efi-bootloader` permite el arranque en sistemas modernos que usan la *interfaz extendida de firmware unificada* (UEFI). Es el que debería ser usado si la imagen de instalación contiene un directorio `/sys/firmware/efi` cuando la arranca en su sistema.

`grub-efi-removable-bootloader` allows you to boot your system from removable media by writing the GRUB file to the UEFI-specification location of `/EFI/BOOT/BOOTX64.efi` of the boot directory, usually `/boot/efi`. This is also useful for some UEFI firmwares that "forget" their configuration from their non-volatile storage. Like `grub-efi-bootloader`, this can only be used if the `/sys/firmware/efi` directory is available.

> **Nota:** This *will* overwrite the GRUB file from any other operating systems that also place their GRUB file in the UEFI-specification location; making them unbootable.

`grub-efi-netboot-bootloader` allows you to boot your system over network through TFTP. In combination with an NFS root file system this allows you to build a diskless Guix system.

The installation of the `grub-efi-netboot-bootloader` generates the content of the TFTP root directory at `targets` (see Section 11.15 [Configuración del gestor de arranque], page 627) below the sub-directory `efi/Guix`, to be served by a TFTP server. You may want to mount your TFTP server directories onto the `targets` to move the required files to the TFTP server automatically during installation.

Si tiene pensado usar también un sistema de archivos raíz NFS (en realidad si monta el almacén desde un directorio compartido con NFS) el servidor TFTP también debe proporcionar el archivo `/boot/grub/grub.cfg` y otros archivos desde el almacén (como las imágenes de fondo de GRUB, el núcleo (see Section 11.3 [Referencia de operating-system], page 253) y el disco en RAM para el arranque (see Section 11.3 [Referencia de operating-system], page 253)). GRUB accederá a todos estos archivos del almacén a través de TFTP con su ruta del almacén habitual, como por ejemplo `tftp://tftp-server/gnu/store/...-initrd/initrd.cpio.gz`.

Two symlinks are created to make this possible. For each target in the `targets` field, the first symlink is '`target`'`/efi/Guix/boot/grub/grub.cfg` pointing to `../../../boot/grub/grub.cfg`, where '`target`' may be `/boot`. In this case the link is not leaving the served TFTP root directory, but otherwise it does. The second link is '`target`'`/gnu/store` and points to `../gnu/store`. This link is leaving the served TFTP root directory.

The assumption behind all this is that you have an NFS server exporting the root file system for your Guix system, and additionally a TFTP server exporting your `targets` directories—usually a single `/boot`—from that same root file system for your Guix system. In this constellation the symlinks will work.

For other constellations you will have to program your own bootloader installer, which then takes care to make necessary files from the store accessible through TFTP, for example by copying them into the TFTP root directory for your `targets`.

It is important to note that symlinks pointing outside the TFTP root directory may need to be allowed in the configuration of your TFTP server. Further the store link exposes the whole store through TFTP. Both points need to be considered carefully for security aspects. It is advised to disable any TFTP write access!

Please note, that this bootloader will not modify the 'UEFI Boot Manager' of the system.

Además de lo expresado anteriormente—`grub-efi-netboot-bootloader`, los servidores TFTP y NFS—también necesitará un servidor DHCP configurado correctamente para hacer posible el arranque a través de la red. Para esto únicamente podemos recomendarle por el momento que busque información sobre PXE (Preboot eXecution Environment).

If a local EFI System Partition (ESP) or a similar partition with a FAT file system is mounted in `targets`, then symlinks cannot be created. In this case everything will be prepared for booting from local storage, matching the behavior of `grub-efi-bootloader`, with the difference that all GRUB binaries are copied to `targets`, necessary for booting over the network.

`grub-efi-netboot-removable-bootloader` is identical to `grub-efi-netboot-bootloader` with the exception that the sub-directory `efi/boot` will be used instead of `efi/Guix` to comply with the UEFI specification for removable media.

> **Nota:** This *will* overwrite the GRUB file from any other operating systems that also place their GRUB file in the UEFI-specification location; making them unbootable.

`targets` This is a list of strings denoting the targets onto which to install the bootloader.

The interpretation of targets depends on the bootloader in question. For `grub-bootloader`, for example, they should be device names understood by the bootloader `installer` command, such as `/dev/sda` or `(hd0)` (see Section "Invoking grub-install" in *GNU GRUB Manual*). For `grub-efi-bootloader` and `grub-efi-removable-bootloader` they should be mount points of the EFI file system, usually `/boot/efi`. For `grub-efi-netboot-bootloader`, `targets` should be the mount points corresponding to TFTP root directories served by your TFTP server.

`menu-entries` (predeterminadas: `'()`)
: Una lista posiblemente vacía de objetos `menu-entry` (véase a continuación), que indican entradas que deben aparecer en el menú del cargador de arranque, además de la entrada del sistema actual y la entrada que apunta a generaciones previas del sistema.

`default-entry` (predeterminada: `0`)
: El índice de la entrada del menú de arranque por omisión. El índice 0 es para la entrada del sistema actual.

`timeout` (predeterminado: `5`)
: El número de segundos que se esperará entrada por el teclado antes de arrancar. El valor 0 indica que se debe arrancar de forma inmediata, y -1 que se debe esperar indefinidamente.

`keyboard-layout` (predeterminada: `#f`)
: Si es `#f`, el menú del cargador de arranque (si existe) usa la distribución de teclado predeterminada, habitualmente inglés estadounidense ("qwerty").

  En otro caso, debe ser un objeto `keyboard-layout` (see Section 11.8 [Distribución de teclado], page 271).

  > **Nota:** Esta opción se ignora actualmente por todos los cargadores de arranque menos `grub` y `grub-efi`.

`theme` (predeterminado: *#f*)
: El objeto del tema del cargador de arranque que describe el tema usado. Si no se proporciona ningún tema, algunos cargadores de arranque pueden usar un tema por omisión, lo cual es cierto en GRUB.

`terminal-outputs` (predeterminadas: `'(gfxterm)`)
: Los terminales de salida que se usarán para el menú de arranque, como una lista de símbolos. GRUB acepta los valores: `console`, `serial`, `serial_{0-3}`, `gfxterm`, `vga_text`, `mda_text`, `morse` y `pkmodem`. Este campo corresponde con la variable `GRUB_TERMINAL_OUTPUT` (see Section "Simple configuration" in *GNU GRUB manual*).

`terminal-inputs` (predeterminadas: `'()`)
: Los terminales de entrada que se usarán para el menú de arranque, como una lista de símbolos. Para GRUB, el valor predeterminado es el terminal nativo de la plataforma determinado en tiempo de ejecución. GRUB acepta los valores: `console`, `serial`, `serial{0-3}`, `at_keyboard` y `usb_keyboard`. Este campo corresponde a la variable GRUB `GRUB_TERMINAL_INPUT` (see Section "Simple configuration" in *GNU GRUB manual*).

`serial-unit` (predeterminada: `#f`)
: La unidad serie usada por el cargador de arranque, como un entero del 0 al 3. Para GRUB, se selecciona en tiempo de ejecución; actualmente GRUB selecciona 0 lo que corresponde a COM1 (see Section "Serial terminal" in *GNU GRUB manual*).

`serial-speed` (predeterminada: `#f`)
: La velocidad de la interfaz serie, como un entero. Para GRUB, el valor predeterminado se selecciona en tiempo de ejecución, actualmente GRUB

selecciona 9600 bps (see Section "Serial terminal" in *GNU GRUB manual*).

`device-tree-support?` (default: `#t`)
Whether to support Linux device tree (`https://en.wikipedia.org/wiki/Devicetree`) files loading.

This option in enabled by default. In some cases involving the `u-boot` bootloader, where the device tree has already been loaded in RAM, it can be handy to disable the option by setting it to `#f`.

`extra-initrd` (default: `#f`)
File name of an additional initrd to load during the boot. It may or may not point to a file in the store, but the main use case is for out-of-store files containing secrets.

In order to be able to provide decryption keys for the LUKS device, they need to be available in the initial ram disk. However they cannot be stored inside the usual initrd, since it is stored in the store and being a world-readable (as files in the store are) is not a desired property for a initrd containing decryption keys. You can therefore use this field to instruct GRUB to also load a manually created initrd not stored in the store.

For any use case not involving secrets, you should use regular initrd (see Section 11.3 [Referencia de operating-system], page 253) instead.

Suitable image can be created for example like this:

```
echo /key-file.bin | cpio -oH newc >/key-file.cpio
chmod 0000 /key-file.cpio
```

After it is created, you can use it in this manner:

```
;; Operating system with encrypted boot partition
(operating-system
  ...
  (bootloader (bootloader-configuration
               (bootloader grub-efi-bootloader)
               (targets '("/boot/efi"))
               ;; Load the initrd with a key file
               (extra-initrd "/key-file.cpio")))
  (mapped-devices
   (list (mapped-device
          (source (uuid "12345678-1234-1234-1234-123456789abc"))
          (target "my-root")
          (type (luks-device-mapping-with-options
                 ;; And use it to unlock the root device
                 #:key-file "/key-file.bin"))))))
```

Be careful when using this option, since pointing to a file that is not readable by the grub while booting will cause the boot to fail and require a manual edit of the initrd line in the grub menu.

Currently only supported by GRUB.

Si desease listar entradas adicionales para el menú de arranque a través del campo `menu-entries` mostrado previamente, deberá crearlas con la forma `menu-entry`. Por ejemplo, imagine que desea ser capaz de arrancar otra distribución (¡difícil de imaginar!), puede definir una entrada de menú de esta forma:

```
(menu-entry
  (label "La otra distribución")
  (linux "/boot/old/vmlinux-2.6.32")
  (linux-arguments '("root=/dev/sda2"))
  (initrd "/boot/old/initrd"))
```

Los detalles se encuentran a continuación.

`menu-entry` [Tipo de datos]

El tipo de una entrada en el menú del cargador de arranque.

`label` La etiqueta a mostrar en el menú—por ejemplo, `"GNU"`.

`linux` (predeterminado: `#f`)
La imagen del núcleo Linux a arrancar, por ejemplo:

```
(file-append linux-libre "/bzImage")
```

Con GRUB, también es posible especificar un dispositivo explícitamente mediante el uso de la convención de nombres de dispositivo de GRUB (see Section "Naming convention" in *GNU GRUB manual*), por ejemplo:

```
"(hd0,msdos1)/boot/vmlinuz"
```

Si se especifica el dispositivo explícitamente como en el ejemplo anterior, el campo `device` se ignora completamente.

`linux-arguments` (predeterminados: `'()`)
La lista de parámetros extra de línea de órdenes para el núcleo Linux—por ejemplo, `'("console=ttyS0")`.

`initrd` (predeterminado: `#f`)
Una expresión-G o una cadena que contiene el nombre de archivo del disco inicial en RAM usado (see Section 8.12 [Expresiones-G], page 167).

`device` (predeterminado: `#f`)
El dispositivo donde se encuentran el núcleo y el initrd—es decir, para GRUB, *raíz* de esta entrada de menú (see Section "root" in *GNU GRUB manual*).

Puede ser una etiqueta de sistema de archivos (una cadena), un UUID de sistema de archivos (un vector de bytes, see Section 11.4 [Sistemas de archivos], page 257), o `#f`, en cuyo caso el cargador de arranque buscará el dispositivo que contenga el archivo especificado por el campo `linux` (see Section "search" in *GNU GRUB manual*). *No* debe ser un nombre de dispositivo del SO como `/dev/sda1`.

`multiboot-kernel` (predeterminado: `#f`)
El núcleo a arrancar en modo Multiboot (see Section "multiboot" in *GNU GRUB manual*). Cuando se proporciona este campo, se genera una entrada Multiboot en el menú. Por ejemplo:

```
(file-append mach "/boot/gnumach")
```

`multiboot-arguments` (predeterminados: `'()`)
: Lista de parámetros adicionales que se proporcionan al núcleo Multiboot en la línea de órdenes.

  For example, when running in QEMU it can be useful to use a text-based console (use options `--nographic --serial mon:stdio`):

  ```
  '("console=com0")
  ```

  To use the new and still experimental rumpdisk user-level disk driver (`https://darnassus.sceen.net/~hurd-web/rump_kernel/`) instead of GNU Mach's in-kernel IDE driver, set `kernel-arguments` to:

  ```
  '("noide")
  ```

  Of course, these options can be combined:

  ```
  '("console=com0" "noide")
  ```

`multiboot-modules` (predeterminados: `'()`)
: Lista de ordenes para cargar módulos de Multiboot. Por ejemplo:

  ```
  (list (list (file-append hurd "/hurd/ext2fs.static") "ext2fs"
              ...)
        (list (file-append libc "/lib/ld.so.1") "exec"
              ...))
  ```

`chain-loader` (default: `#f`)
: A string that can be accepted by `grub`'s `chainloader` directive. This has no effect if either `linux` or `multiboot-kernel` fields are specified. The following is an example of chainloading a different GNU/Linux system.

  ```
  (bootloader
   (bootloader-configuration
    ;; ...
    (menu-entries
     (list
      (menu-entry
       (label "GNU/Linux")
       (device (uuid "1C31-A17C" 'fat))
       (chain-loader "/EFI/GNULinux/grubx64.efi"))))))
  ```

De momento únicamente GRUB permite el uso de temas. Los temas de GRUB se crean mediante el uso de `grub-theme`, todavía no documentado completamente.

`grub-theme` [Tipo de datos]
: Tipo de datos que representa la configuración de un tema de GRUB.

  `gfxmode` (predeterminado: `'("auto")`)
  : El modo gráfico `gfxmode` de GRUB configurado (una lista de cadenas con resoluciones de pantalla, see Section "gfxmode" in *GNU GRUB manual*).

`grub-theme` [Procedimiento]
: Devuelve el tema predeterminado de GRUB que usa el sistema operativo si no se especifica el campo `theme` en el registro `bootloader-configuration`.

  Viene con una bonita imagen de fondo que muestra los logos de GNU y Guix.

Por ejemplo, para usar una resolución distinta de la predeterminada, puede usar algo como esto:

```
(bootloader
 (bootloader-configuration
 ;; ...
 (theme (grub-theme
         (inherit (grub-theme))
         (gfxmode '("1024x786x32" "auto"))))))
```

## 11.16 Invoking `guix system`

Una vez haya escrito la declaración de sistema operativo como se ha visto en la sección previa, puede *instanciarse* mediante el uso de la orden `guix system`. Su sinopsis es:

```
guix system opciones... acción archivo
```

*archivo* debe ser el nombre de un archivo que contenga una declaración `operating-system`. *acción* especifica cómo se instancia el sistema operativo. Actualmente se permiten los siguientes valores:

`search` Muestra las definiciones de tipos de servicio disponibles que corresponden con las expresiones regulares proporcionadas, ordenadas por relevancia:

```
$ guix system search console
name: console-fonts
location: gnu/services/base.scm:806:2
extends: shepherd-root
description: Install the given fonts on the specified ttys (fonts are per
+ virtual console on GNU/Linux).  The value of this service is a list of
+ tty/font pairs.  The font can be the name of a font provided by the `kbd'
+ package or any valid argument to `setfont', as in this example:
+
+      '(("tty1" . "LatGrkCyr-8x16")
+        ("tty2" . (file-append
+                   font-tamzen
+                   "/share/kbd/consolefonts/TamzenForPowerline10x20.psf"
+        ("tty3" . (file-append
+                   font-terminus
+                   "/share/consolefonts/ter-132n"))) ; for HDPI
relevance: 9

name: mingetty
location: gnu/services/base.scm:1190:2
extends: shepherd-root
description: Provide console login using the `mingetty' program.
relevance: 2

name: login
location: gnu/services/base.scm:860:2
extends: pam
```

```
description: Provide a console log-in service as specified by its
+ configuration value, a `login-configuration' object.
relevance: 2

...
```

Como con `guix package --search`, el resultado se obtiene en formato `recutils`, lo que facilita el filtrado de la salida (see *GNU recutils manual*).

`edit` Edit or view the definition of the given service types.

For example, the command below opens your editor, as specified by the `EDITOR` environment variable, on the definition of the `openssh` service type:

```
guix system edit openssh
```

`reconfigure`

Construye el sistema operativo descrito en *archivo*, lo activa, y se constituye como estado actual[20].

> **Nota:** Es altamente recomendable ejecutar `guix pull` antes de la primera ejecución de `guix system reconfigure` (see Section 5.7 [Invocación de guix pull], page 57). No hacerlo puede ocasionar que se obtenga una versión más antigua de Guix una vez que `reconfigure` se haya completado.

Lleva a efecto toda la configuración especificada en *archivo*: cuentas de usuaria, servicios del sistema, lista de paquetes global, programas con setuid, etc. La orden inicia los servicios del sistema especificados en *archivo* que no estén actualmente en ejecución; si un servicio se encuentra en ejecución esta orden prepara su actualización durante la próxima parada (por ejemplo, con `herd stop X` o `herd restart X`).

Esta orden crea una nueva generación cuyo número es el sucesor de la siguiente generación (como lo muestra `guix system list-generations`). Si esa generación ya existe, será sobreescrita. Este comportamiento es el mismo que el de `guix package` (see Section 5.2 [Invocación de guix package], page 36).

También añade una entrada al cargador de arranque para la nueva configuración del sistema operativo—en caso de que no se proporcione la opción `--no-bootloader`. Con GRUB, mueve las entradas de configuraciones antiguas a un submenú, permitiendo la selección de una generación previa del sistema en tiempo de arranque en caso necesario.

Tras la finalización, el nuevo sistema se despliega en `/run/current-system`. Este directorio contiene *metadatos de procedencia*: la lista de canales usados (see Chapter 6 [Canales], page 69) y el *archivo* en sí, cuando esté disponible. Puede visualizar dichos metadatos con la siguiente orden:

```
guix system describe
```

Esta información es útil en caso de que desee inspeccionar posteriormente cómo se construyó está generación en particular. De hecho, asumiendo que *archivo* es

[20] Esta acción (y las acciones relacionadas `switch-generation` y `roll-back`) son usables únicamente en sistemas que ya ejecuten el sistema Guix.

autocontenido, puede construir de nuevo la generación *n* de su sistema operativo con:

```
guix time-machine \
  -C /var/guix/profiles/system-n-link/channels.scm -- \
  system reconfigure \
  /var/guix/profiles/system-n-link/configuration.scm
```

¡Puede pensar en ello como una especie de control de versiones incorporado en Guix! Su sistema no es únicamente un artefacto binario: *transporta sus propias fuentes con él*. See Section 11.19.3 [Referencia de servicios], page 652, para más información sobre el seguimiento de procedencia.

De manera predeterminada, `reconfigure` *evita que instale una version anterior en su sistema operativo*, lo que podría (re)introducir fallos de seguridad y también provocar problemas con servicios que mantienen estado como los sistemas de gestión de bases de datos. Puede modificar este comportamiento con el parámetro `--allow-downgrades`.

`switch-generation`

Cambia a una generación existente del sistema. Esta acción cambia atómicamente el perfil del sistema a la generación del sistema especificada. También redistribuye las entradas de sistema del menú de arranque existentes. Marca como predeterminada la entrada de la generación de sistema especificada y mueve las entradas de otras generaciones a un submenú, si el cargador de arranque lo permite. La próxima vez que se arranque el sistema, se usará la generación de sistema especificada.

El cargador de arranque en sí no se reinstala durante esta orden. Por tanto, el cargador de arranque instalado se usa con un archivo de configuración actualizado.

La generación deseada puede especificarse explícitamente con su numero de generación. Por ejemplo, la siguiente invocación cambiaría a la generación 7 del sistema:

```
guix system switch-generation 7
```

La generación deseada puede especificarse también de forma relativa a la generación actual con la forma `+N` o `-N`, donde `+3` significa "3 generaciones después de la generación actual", y `-1` significa "1 generación antes de la generación actual". Cuando se especifica un valor negativo como `-1` debe ir precedido de `--` para evitar que se analice como una opción. Por ejemplo:

```
guix system switch-generation -- -1
```

Actualmente, el efecto de la invocación de esta acción es *únicamente* cambiar el perfil del sistema a una generación existente y redistribuir las entradas del menú de arranque. Para realmente empezar a usar la generación deseada del sistema, debe reiniciar tras esta acción. En el futuro, se actualizará para hacer lo mismo que `reconfigure`, como activación y desactivación de servicios.

Esta acción fallará si la generación especificada no existe.

`roll-back`
: Cambia a la generación de sistema previa. Tras el siguiente arranque del sistema, usará la generación de sistema precedente. Es la operación inversa de `reconfigure`, y es equivalente a la invocación de `switch-generation` con `-1` como parámetro.

  Actualmente, como con `switch-generation`, debe reiniciar tras la ejecución de esta acción para realmente empezar a usar la generación de sistema precedente.

`delete-generations`
: Elimina generaciones del sistema, haciendo posible su recolección con la basura (see Section 5.6 [Invocación de guix gc], page 53, para información sobre como llevar a cabo la "recolección de basura").

  Esto funciona del mismo modo que '`guix package --delete-generations`' (see Section 5.2 [Invocación de guix package], page 36). Sin parámetros, se eliminan todas las generaciones del sistema excepto la actual:

  ```
  guix system delete-generations
  ```

  También puede seleccionar las generaciones que desea eliminar. El siguiente ejemplo elimina todas las generaciones del sistema que tienen más de dos meses de antigüedad:

  ```
  guix system delete-generations 2m
  ```

  La ejecución de esta orden reinstala automáticamente el cargador de arranque con una lista de entradas del menú actualizada—por ejemplo, el submenú de generaciones antiguas en GRUB no mostrará las generaciones que hayan sido borradas.

`build`
: Construye la derivación del sistema operativo, que incluye todos los archivos de configuración y programas necesarios para el arranque y la ejecución del sistema. Esta acción no instala nada en realidad.

`init`
: Construye el directorio dado con todos los archivos necesarios para ejecutar el sistema operativo especificado en *archivo*. Esto es útil para la instalación inicial de Guix. Por ejemplo:

  ```
  guix system init mi-configuración-del-so.scm /mnt
  ```

  copia a `/mnt` todos los elementos del almacén necesarios para la configuración especificada en `mi-configuración-del-so.scm`. Esto incluye los archivos de configuración, paquetes y demás. También crea otros archivos esenciales necesarios para la correcta operación del sistema—por ejemplo, los directorios `/etc`, `/var` y `/run`, y el archivo `/bin/sh`.

  This command also installs bootloader on the targets specified in `my-os-config`, unless the `--no-bootloader` option was passed.

`vm`
: Build a virtual machine (VM) that contains the operating system declared in *file*, and return a script to run that VM.

  > **Nota:** La acción `vm` y otras presentadas a continuación pueden usar la funcionalidad KVM del núcleo Linux-libre. Específicamente, si la máquina permite la virtualización hardware, debe cargarse el correspondiente módulo KVM del núcleo, debe existir el nodo del

dispositivo `/dev/kvm` y tanto la propia usuaria como las usuarias de construcción del daemon deben tener acceso de lectura y escritura al mismo (see Section 2.2.1 [Configuración del entorno de construcción], page 6).

Los parámetros proporcionados al guión se pasan a QEMU como en el siguiente ejemplo, que activa la red y solicita 1 GiB de RAM para la máquina emulada:

```
$ /gnu/store/...-run-vm.sh -m 1024 -smp 2 -nic user,model=virtio-net-pci
```

It's possible to combine the two steps into one:

```
$ $(guix system vm my-config.scm) -m 1024 -smp 2 -nic user,model=virtio-net-
```

La VM comparte su almacén con el sistema anfitrión.

By default, the root file system of the VM is mounted volatile; the `--persistent` option can be provided to make it persistent instead. In that case, the VM disk-image file will be copied from the store to the `TMPDIR` directory to make it writable.

Sistemas de archivos adicionales pueden compartirse entre la máquina anfitriona y la virtual mediante el uso de las opciones `--share` y `--expose`: la primera especifica un directorio a compartir con acceso de escritura, mientras que la última proporciona solo acceso de lectura al directorio compartido.

El siguiente ejemplo crea una máquina virtual en la que el directorio de la usuaria es accesible en modo solo-lecture, y donde el directorio `/intercambio` esta asociado de forma lectura-escritura con `$HOME/tmp` en el sistema anfitrión:

```
guix system vm mi-configuración.scm \
   --expose=$HOME --share=$HOME/tmp=/intercambio
```

En GNU/Linux, lo predeterminado es arrancar directamente el núcleo; esto posee la ventaja de necesitar únicamente una pequeña imagen del disco raíz pequeña ya el el almacén de la anfitriona puede montarse.

The `--full-boot` option forces a complete boot sequence, starting with the bootloader. This requires more disk space since a root image containing at least the kernel, initrd, and bootloader data files must be created.

The `--image-size` option can be used to specify the size of the image.

The `--no-graphic` option will instruct `guix system` to spawn a headless VM that will use the invoking tty for IO. Among other things, this enables copy-pasting, and scrollback. Use the *`ctrl`*`-a` prefix to issue QEMU commands; e.g. *`ctrl`*`-a` *`h`* prints a help, *`ctrl`*`-a` *`x`* quits the VM, and *`ctrl`*`-a` *`c`* switches between the QEMU monitor and the VM.

`image` The `image` command can produce various image types. The image type can be selected using the `--image-type` option. It defaults to `mbr-hybrid-raw`. When its value is `iso9660`, the `--label` option can be used to specify a volume ID with `image`. By default, the root file system of a disk image is mounted non-volatile; the `--volatile` option can be provided to make it volatile instead. When using `image`, the bootloader installed on the generated image is taken from the provided `operating-system` definition. The following example demonstrates how to generate an image that uses the `grub-efi-bootloader` bootloader and boot it with QEMU:

```
image=$(guix system image --image-type=qcow2 \
        gnu/system/examples/lightweight-desktop.tmpl)
cp $image /tmp/my-image.qcow2
chmod +w /tmp/my-image.qcow2
qemu-system-x86_64 -enable-kvm -hda /tmp/my-image.qcow2 -m 1000 \
                   -bios $(guix build ovmf-x86-64)/share/firmware/ovmf_x64.b
```

When using the `mbr-hybrid-raw` image type, a raw disk image is produced; it can be copied as is to a USB stick, for instance. Assuming `/dev/sdc` is the device corresponding to a USB stick, one can copy the image to it using the following command:

```
# dd if=$(guix system image my-os.scm) of=/dev/sdc status=progress
```

La orden `--list-image-types` muestra todos los tipos de imagen disponibles.

When using the `qcow2` image type, the returned image is in qcow2 format, which the QEMU emulator can efficiently use. See Section 11.18 [Ejecutar Guix en una máquina virtual], page 647, for more information on how to run the image in a virtual machine. The `grub-bootloader` bootloader is always used independently of what is declared in the `operating-system` file passed as argument. This is to make it easier to work with QEMU, which uses the SeaBIOS BIOS by default, expecting a bootloader to be installed in the Master Boot Record (MBR).

When using the `docker` image type, a Docker image is produced. Guix builds the image from scratch, not from a pre-existing Docker base image. As a result, it contains *exactly* what you define in the operating system configuration file. You can then load the image and launch a Docker container using commands like the following:

```
image_id="$(docker load < guix-system-docker-image.tar.gz)"
container_id="$(docker create $image_id)"
docker start $container_id
```

Esta orden arranca un contenedor Docker nuevo a partir de la imagen especificada. El sistema Guix se arrancará de la manera habitual, lo que implica el inicio de cualquier servicio que se haya definido en la configuración del sistema operativo. Puede iniciar una sesión de shell interactiva en el contenedor mediante el uso de `docker exec`:

```
docker exec -ti $container_id /run/current-system/profile/bin/bash --login
```

Dependiendo de lo que ejecute en el contenedor Docker, puede ser necesario proporcionar permisos adicionales al contenedor. Por ejemplo, si pretende construir software mediante el uso de Guix dentro del contenedor Docker, puede tener que proporcionar la opción `--privileged` a `docker create`.

Por último, la opción `--network` afecta a `guix system docker-image`: produce una imagen donde la red supuestamente se comparte con el sistema anfitrión, y por lo tanto no contiene servicios como nscd o NetworkManager.

`container`

Devuelve un guión de la ejecución del sistema operativo declarado en *archivo* dentro de un contenedor. Los contenedores son un conjunto de mecanismos

de aislamiento ligeros que proporciona el núcleo Linux-libre. Los contenedores necesitan sustancialmente menos recursos que máquinas virtuales completas debido a que el núcleo, los objetos compartidos y otros recursos pueden compartirse con el sistema anfitrión; esto también significa que proporcionan un menor aislamiento.

En este momento, el guión debe ejecutarse como root para permitir más de una única usuaria y grupo. El contenedor comparte su almacén con la máquina anfitriona.

Como con la acción `vm` (see [guix system vm], page 637), sistemas de archivos adicionales a compartir entre la máquina anfitriona y el contenedor pueden especificarse mediante el uso de las opciones `--share` y `--expose`:

```
guix system container mi-configuración.scm \
   --expose=$HOME --share=$HOME/tmp=/intercambio
```

The `--share` and `--expose` options can also be passed to the generated script to bind-mount additional directories into the container.

**Nota:** Esta opción requiere Linux-libre 3.19 o posterior.

*opciones* puede contener cualquiera de las opciones de construcción comunes (see Section 9.1.1 [Opciones comunes de construcción], page 181). Además, *opciones* puede contener una de las siguientes:

`--expression=`*expr*

`-e` *expr* Considera el sistema operativo al cual evalúa *expr*. Es una alternativa a la especificación de un archivo que evalúe a un sistema operativo. Se usa para la generación de la imagen de instalación de Guix (see Section 3.9 [Construcción de la imagen de instalación], page 31).

`--system=`*sistema*

`-s` *sistema*

Intenta la construcción para *sistema* en vez de para el tipo de la máquina anfitriona. Funciona como en `guix build` (see Section 9.1 [Invocación de guix build], page 181).

`--target=`*tripleta*

Compilación cruzada para la *tripleta*, que debe ser una tripleta GNU válida, cómo `"aarch64-linux-gnu"` (see Section "Specifying target triplets" in *Autoconf*).

`--derivation`

`-d` Devuelve el nombre de archivo de la derivación del sistema operativo proporcionado sin construir nada.

`--save-provenance`

Como se ha mostrado previamente, `guix system init` y `guix system reconfigure` siempre almacenan información de procedencia a través de un servicio dedicado (see Section 11.19.3 [Referencia de servicios], page 652). No obstante, otras órdenes no hacen esto de manera predeterminada. Si desea, digamos, crear una imagen de máquina virtual que contenga información de procedencia, puede ejecutar:

```
guix system image -t qcow2 --save-provenance config.scm
```

De este modo, la imagen resultante "embeberá sus propias fuentes" de manera efectiva en forma de metadatos en `/run/current-system`. Con dicha información se puede reconstruir la imagen para asegurarse de que realmente contiene lo que dice contener; o se puede usar para derivar una variante de la imagen.

`--image-type=`*tipo*

`-t` *tipo* For the `image` action, create an image with given *type*.

When this option is omitted, `guix system` uses the `mbr-hybrid-raw` image type.

`--file-system-type=iso9660` produce una imagen ISO-9660, que puede ser grabada en CD y DVD.

`--image-size=`*tamaño*

For the `image` action, create an image of the given *size*. *size* may be a number of bytes, or it may include a unit as a suffix (see Section "Block size" in *GNU Coreutils*).

Cuando se omite esta opción, `guix system` calcula una estimación del tamaño de la imagen en función del tamaño del sistema declarado en *archivo*.

`--network`

`-N` Con la acción `container`, permite a los contenedores acceder a la red de la máquina anfitriona, es decir, no crea un espacio de nombres de red.

`--root=`*archivo*

`-r` *archivo*

Hace que *archivo* sea un enlace simbólico al resultado, y lo registra como una raíz del recolector de basura.

`--skip-checks`

Omite las comprobaciones de seguridad previas a la instalación.

Por omisión, `guix system init` y `guix system reconfigure` realizan comprobaciones de seguridad: se aseguran de que los sistemas de archivos que aparecen en la declaración `operating-system` realmente existen (see Section 11.4 [Sistemas de archivos], page 257) y que cualquier módulo del núcleo Linux que pudiese necesitarse durante el arranque se encuentre en `initrd-modules` (see Section 11.14 [Disco en RAM inicial], page 624). El uso de esta opción omite todas estas comprobaciones.

`--allow-downgrades`

Indica a `guix system reconfigure` que permita la instalación de versiones más antiguas.

De manera predeterminada `reconfigure` evita que instale una versión más antigua. Esto se consigue comparando la información de procedencia en su sistema operativo (mostrada por `guix system describe`) con aquella de la orden `guix` (mostrada por `guix describe`). Si las revisiones de `guix` no son descendientes de las usadas en su sistema, `guix system reconfigure` termina con un error. Proporcionar la opción `--allow-downgrades` le permite evitar estas comprobaciones.

> **Nota:** Asegúrese de entender las implicaciones de seguridad antes de usar la opción `--allow-downgrades`.

`--on-error=`*estrategia*
: Aplica *estrategia* cuando ocurre un error durante la lectura de *archivo*. *estrategia* puede ser uno de los siguientes valores:

  `nothing-special`
  : Informa concisamente del error y termina la ejecución. Es la estrategia predeterminada.

  `backtrace`
  : Del mismo modo, pero también muestra la secuencia de llamadas.

  `debug`
  : Informa del error y entra en el depurador de Guile. A partir de ahí, puede ejecutar órdenes como `,bt` para obtener la secuencia de llamadas, `,locals` para mostrar los valores de las variables locales, e inspeccionar el estado del programa de forma más general. See Section "Debug Commands" in *GNU Guile Reference Manual*, para una lista de órdenes de depuración disponibles.

Una vez haya construido, configurado, reconfigurado y re-reconfigurado su instalación de Guix, puede encontrar útil enumerar las generaciones del sistema operativo disponibles en el disco—y que puede seleccionar en el menú de arranque:

`describe`
: Describe the running system generation: its file name, the kernel and bootloader used, etc., as well as provenance information when available.

  The `--list-installed` flag is available, with the same syntax that is used in `guix package --list-installed` (see Section 5.2 [Invocación de guix package], page 36). When the flag is used, the description will include a list of packages that are currently installed in the system profile, with optional filtering based on a regular expression.

  > **Nota:** The *running* system generation—referred to by `/run/current-system`—is not necessarily the *current* system generation—referred to by `/var/guix/profiles/system`: it differs when, for instance, you chose from the bootloader menu to boot an older generation.
  >
  > It can also differ from the *booted* system generation—referred to by `/run/booted-system`—for instance because you reconfigured the system in the meantime.

`list-generations`
: Muestra un resumen de cada generación del sistema operativo disponible en el disco, de manera legible por humanos. Es similar a la opción `--list-generations` de `guix package` (see Section 5.2 [Invocación de guix package], page 36).

  De manera opcional, se puede especificar un patrón, con la misma sintaxis que la usada en `guix package --list-generations`, para restringir la lista de generaciones mostradas. Por ejemplo, la siguiente orden muestra generaciones que tienen hasta 10 días de antigüedad:

  ```
  $ guix system list-generations 10d
  ```

The `--list-installed` flag may also be specified, with the same syntax that is used in `guix package --list-installed`. This may be helpful if trying to determine when a package was added to the system.

¡La orden `guix system` tiene aún más que ofrecer! Las siguientes ordenes le permiten visualizar cual es la relación entre los servicios del sistema:

`extension-graph`

Emit to standard output the *service extension graph* of the operating system defined in *file* (see Section 11.19.1 [Composición de servicios], page 649, for more information on service extensions). By default the output is in Dot/Graphviz format, but you can choose a different format with `--graph-backend`, as with `guix graph` (see Section 9.10 [Invocación de guix graph], page 221):

La orden:

```
$ guix system extension-graph archivo | xdot -
```

muestra las relaciones de extensión entre los servicios.

**Nota:** The `dot` program is provided by the `graphviz` package.

`shepherd-graph`

Emit to standard output the *dependency graph* of shepherd services of the operating system defined in *file*. See Section 11.19.4 [Servicios de Shepherd], page 657, for more information and for an example graph.

Again, the default output format is Dot/Graphviz, but you can pass `--graph-backend` to select a different one.

## 11.17 Invoking `guix deploy`

Ya hemos visto como usar declaraciones `operating-system` para gestionar la configuración de una máquina de manera local. Supongamos no obstante que necesita configurar múltiples máquinas—quizá esté gestionando un servicio en la web que se componga de varios servidores. `guix deploy` le permite usar las mismas declaraciones `operating-system` para gestionar múltiples máquinas remotas como un único "despliegue" lógico.

**Nota:** La funcionalidad descrita en esta sección está todavía en desarrollo y sujeta a cambios. Puede ponerse en contacto con nosotras a través de `guix-devel@gnu.org`.

`guix deploy archivo`

Dicha invocación llevará a cabo en las máquinas el despliegue al cual el *archivo* evalúe. Como ejemplo, *archivo* puede contener una definición como esta:

```
;; Este es un despliegue de Guix con una configuración en
;; mínima ("en los huesos"), sin servidor gráfico X11,
;; en una máquina con un daemon SSH escuchando en
;; localhost:2222. Una configuración como esta puede ser
;; apropiada para máquinas virtuales con puertos redirigidos
;; a la interfaz local de la máquina anfitriona.
(use-service-modules networking ssh)
(use-package-modules bootloaders)
```

```
(define %system
  (operating-system
   (host-name "gnu-deployed")
   (timezone "Etc/UTC")
   (bootloader (bootloader-configuration
                (bootloader grub-bootloader)
                (targets '("/dev/vda"))
                (terminal-outputs '(console))))
   (file-systems (cons (file-system
                        (mount-point "/")
                        (device "/dev/vda1")
                        (type "ext4"))
                       %base-file-systems))
   (services
    (append (list (service dhcp-client-service-type)
                  (service openssh-service-type
                           (openssh-configuration
                            (permit-root-login #t)
                            (allow-empty-passwords? #t))))
            %base-services))))

(list (machine
       (operating-system %sistema)
       (environment managed-host-environment-type)
       (configuration (machine-ssh-configuration
                       (host-name "localhost")
                       (system "x86_64-linux")
                       (user "alicia")
                       (identity "./id_rsa")
                       (port 2222)))))
```

El archivo debe evaluar a una lista de objetos *machine*. Este ejemplo, durante el despliegue, creará una nueva generación en el sistema remoto que implemente la declaración `operating-system` *%system*. `environment` y `configuration` especifican cómo debe aprovisionarse la máquina—es decir, cómo se crean y gestionan los recursos computacionales. El ejemplo previo no crea ningún recurso, ya que `'managed-host` es una máquina que ya está ejecutando el sistema Guix y está disponible a través de la red. Este es un caso particularmente simple; un despliegue más complejo puede implicar, por ejemplo, el arranque de máquinas virtuales a través de un proveedor de servidores privados virtuales (VPS). En dicho caso se usaría un tipo distinto en *environment*.

Tenga en cuenta que primero debe generar un par de claves en la máquina coordinadora para permitir al daemon exportar archivos firmados de archivos en el almacén (see Section 5.11 [Invocación de guix archive], page 66), aunque este paso se realiza de manera automática en el sistema Guix:

```
# guix archive --generate-key
```

Cada máquina de destino debe autorizar a la clave de la máquina maestra para que acepte elementos del almacén que reciba de la coordinadora:

```
# guix archive --authorize < clave-publica-coordinadora.txt
```

La usuaria proporcionada en `user`, en este ejemplo, especifica la cuenta de la usuaria con la que ingresar en el sistema para realizar el despliegue. Su valor predeterminado es `root`, pero el ingreso al sistema como root a través de SSH en algunos casos puede no estar permitido. Para solventar este problema, `guix deploy` puede ingresar al sistema como una usuaria sin privilegios y ejecutar `sudo` para escalar privilegios. Esto funciona únicamente si `sudo` está instalado en el sistema remoto y se puede invocar de manera no interactiva como `user`. Es decir: la línea de `sudoers` que permite a la usuaria `user` la capacidad de usar `sudo` debe contener la etiqueta `NOPASSWD`. Esto se puede conseguir con el siguiente fragmento de la configuración de sistema operativo:

```
(use-modules ...
             (gnu system))                ;para %sudoers-specification

(define %usuaria "nombre")

(operating-system
  ...
  (sudoers-file
     (plain-file "sudoers"
                 (string-append (plain-file-content %sudoers-specification)
                                (format #f "~a ALL = NOPASSWD: ALL~%"
                                        %usuaria)))))
```

Para obtener más información sobre el formato del archivo `sudoers` consulte `man sudoers`.

Once you've deployed a system on a set of machines, you may find it useful to run a command on all of them. The `--execute` or `-x` option lets you do that; the example below runs `uname -a` on all the machines listed in the deployment file:

```
guix deploy file -x -- uname -a
```

One thing you may often need to do after deployment is restart specific services on all the machines, which you can do like so:

```
guix deploy file -x -- herd restart service
```

The `guix deploy -x` command returns zero if and only if the command succeeded on all the machines.

Below are the data types you need to know about when writing a deployment file.

`machine` [Tipo de datos]

Tipo de datos que representa una máquina individual en un despliegue heterogéneo de Guix.

`operating-system`
: El objeto de la configuración de sistema operativo a desplegar.

`environment`
: Un objeto `environment-type` que describe como debe aprovisionarse la máquina.

`configuration` (predeterminado: `#f`)
: Un objeto que describe la configuración para el entorno (`environment`) de la máquina. Si `environment` tiene una configuración predeterminada, puede usarse `#f`. No obstante, si se usa `#f` para un entorno sin configuración predeterminada se emitirá un error.

`machine-ssh-configuration` [Tipo de datos]

Tipo de datos que representa los parámetros del cliente SSH para una máquina con un entorno (`environment`) de tipo gestionado (`managed-host-environment-type`).

`host-name`

`build-locally?` (predeterminado: `#t`)
: Si es falso, las derivaciones del sistema se construirán en la máquina sobre la que se realiza el despliegue.

`system`
: El tipo de sistema que describe la arquitectura de la máquina sobre la que se realiza el despliegue—por ejemplo, `"x86_64-linux"`.

`authorize?` (predeterminado: `#t`)
: Si es verdadero, la clave de firma de la máquina coordinadora debe añadirse al anillo de claves del control de acceso (ACL) de la máquina remota.

`port` (predeterminado: `22`)

`user` (predeterminada: `"root"`)

`identity` (predeterminada: `#f`)
: Cuando se especifica, indica la ruta al archivo que contiene la clave privada de SSH para la identificación con la máquina remota.

`host-key` (predeterminada: `#f`)
: Esta debería ser la clave SSH de la máquina, que puede ser más o menos así:

```
ssh-ed25519 AAAAC3Nz... root@example.org
```

Cuando `host-key` es `#f`, el servidor se identifica con el archivo `~/.ssh/known_hosts`, igual que hace el cliente `ssh` de OpenSSH.

`allow-downgrades?` (predeterminado: `#f`)
: Determina si se permiten instalaciones de versiones potencialmente anteriores.

Al igual que `guix system reconfigure`, `guix deploy` compara la revisión del canal desplegada actualmente en la máquina remota (como muestra `guix system describe`) con aquella que se esté usando en ese momento (como muestra `guix describe`) para determinar si las revisiones que se despliegan son descendientes de aquellas en uso. Cuando no es el caso y el valor de `allow-downgrades?` es falso, emite un error. Esto le permite no instalar accidentalmente una versión anterior en máquinas remotas.

`safety-checks?` (default: `#t`)
: Whether to perform “safety checks” before deployment. This includes verifying that devices and file systems referred to in the operating system

configuration actually exist on the target machine, and making sure that Linux modules required to access storage devices at boot time are listed in the `initrd-modules` field of the operating system.

These safety checks ensure that you do not inadvertently deploy a system that would fail to boot. Be careful before turning them off!

`digital-ocean-configuration` [Tipo de datos]

Tipo de datos que representa el Droplet que debe crearse para la máquina con `environment digital-ocean-environment-type`.

`ssh-key` The path to the SSH private key to use to authenticate with the remote host. In the future, this field may not exist.

`tags` A list of string "tags" that uniquely identify the machine. Must be given such that no two machines in the deployment have the same set of tags.

`region` Descriptor (slug) de región de Digital Ocean, como `"nyc3"`.

`size` Descriptor (slug) de tamaño de Digital Ocean, como `"s-1vcpu-1gb"`

`enable-ipv6?`
Determina si droplet debe crearse con capacidad de usar redes IPv6 o no.

## 11.18 Ejecución de Guix en una máquina virtual

To run Guix in a virtual machine (VM), one can use the pre-built Guix VM image distributed at `https://ftp.gnu.org/gnu/guix/guix-system-vm-image-c7888f5.x86_64-linux.qcow2`. This image is a compressed image in QCOW format. You can pass it to an emulator such as QEMU (`https://qemu.org/`) (see below for details).

Esta imagen arranca en el entorno gráfico Xfce y contiene algunas herramientas usadas de forma común. Puede instalar más software en la imagen mediante la ejecución de `guix package` en un terminal (see Section 5.2 [Invocación de guix package], page 36). También puede reconfigurar el sistema en base a su archivo de configuración inicial, disponible como `/run/current-system/configuration.scm` (see Section 11.2 [Uso de la configuración del sistema], page 244).

Instead of using this pre-built image, one can also build their own image using `guix system image` (see Section 11.16 [Invocación de guix system], page 634).

If you built your own image, you must copy it out of the store (see Section 8.9 [El almacén], page 157) and give yourself permission to write to the copy before you can use it. When invoking QEMU, you must choose a system emulator that is suitable for your hardware platform. Here is a minimal QEMU invocation that will boot the result of `guix system image -t qcow2` on x86_64 hardware:

```
$ qemu-system-x86_64 \
   -nic user,model=virtio-net-pci \
   -enable-kvm -m 2048 \
   -device virtio-blk,drive=myhd \
   -drive if=none,file=guix-system-vm-image-c7888f5.x86_64-linux.qcow2,id=myhd
```

Aquí está el significado de cada una de esas opciones:

`qemu-system-x86_64`
: Esto especifica la plataforma hardware a emular. Debe corresponder con el anfitrión.

`-nic user,model=virtio-net-pci`
: Activa la pila de red en espacio de usuaria sin privilegios. El SO anfitrión puede acceder a la máquina virtualizada pero no al revés. Este es el modo más simple de poner la máquina en red. `model` especifica que dispositivo de red emular: `virtio-net-pci` es un dispositivo especial para sistemas operativos virtualizados y recomendado para la mayor parte de usos. Asumiendo que su plataforma de hardware es x86_64, puede obtener una lista de adaptadores de red disponibles ejecutando `qemu-system-x86_64 -nic model=help`.

`-enable-kvm`
: Si su sistema tiene extensiones de virtualización por hardware, la activación de la implementación de máquinas virtuales (KVM) del núcleo Linux hará que la ejecución sea más rápida.

`-m 2048` RAM disponible para el sistema operativo virtualizado, en mebibytes. El valor predeterminado es 128 MiB, que puede ser insuficiente para algunas operaciones.

`-device virtio-blk,drive=midisco`
: Crea un dispositivo `virtio-blk` llamado "midisco". `virtio-blk` es un mecanismo de "paravirtualización" de dispositivos de bloques que permite a QEMU obtener un mejor rendimiento que se emulase una unidad de disco completa. Véase la documentación de QEMU y KVM para más información.

`-drive if=none,file=/tmp/imagen-qemu,id=midisco`
: Use our QCOW image, the `guix-system-vm-image-c7888f5.x86_64-linux.qcow2` file, as the backing store of the "myhd" drive.

The default `run-vm.sh` script that is returned by an invocation of `guix system vm` does not add a `-nic user` flag by default. To get network access from within the vm add the `(dhcp-client-service)` to your system definition and start the VM using `$(guix system vm config.scm) -nic user`. An important caveat of using `-nic user` for networking is that `ping` will not work, because it uses the ICMP protocol. You'll have to use a different command to check for network connectivity, for example `guix download`.

### 11.18.1 Conexión a través de SSH

Para activar SSH dentro de una máquina virtual debe añadir un servidor SSH como `(openssh-service-type)` en su máquina virtual (see Section 11.10.5 [Servicios de red], page 314). Además debe que redirigir el puerto SSH, el 22 por omisión, a la máquina anfitriona. Puede hacerlo con

```
$(guix system vm config.scm) -nic user,model=virtio-net-pci,hostfwd=tcp::10022-:22
```

Para conectarse a la máquina virtual puede ejecutar

```
ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -p 10022 localhost
```

La `-p` indica a `ssh` el puerto al que se debe conectar. `-o UserKnownHostsFile=/dev/null` evita que `ssh` se queje cada vez que modifique su archivo `config.scm` y la orden `-o StrictHostKeyChecking=no` evita que tenga que autorizar la conexión a una máquina desconocida cada vez que se conecte.

> **Nota:** If you find the above '`hostfwd`' example not to be working (e.g., your SSH client hangs attempting to connect to the mapped port of your VM), make sure that your Guix System VM has networking support, such as by using the `dhcp-client-service-type` service type.

### 11.18.2 Uso de `virt-viewer` con Spice

Como alternativa al cliente gráfico predeterminado de `qemu` puede usar `remote-viewer` del paquete `virt-viewer`. Para conectarse proporcione la opción `-spice port=5930,disable-ticketing` a `qemu`. Véase la sección previa para más información sobre cómo hacer esto.

Spice also allows you to do some nice stuff like share your clipboard with your VM. To enable that you'll also have to pass the following flags to `qemu`:

```
-device virtio-serial-pci,id=virtio-serial0,max_ports=16,bus=pci.0,addr=0x5
-chardev spicevmc,name=vdagent,id=vdagent
-device virtserialport,nr=1,bus=virtio-serial0.0,chardev=vdagent,\
name=com.redhat.spice.0
```

También deber añadir a su definición de sistema el servicio `(spice-vdagent-service)` (see Section 11.10.37 [Servicios misceláneos], page 603).

## 11.19 Definición de servicios

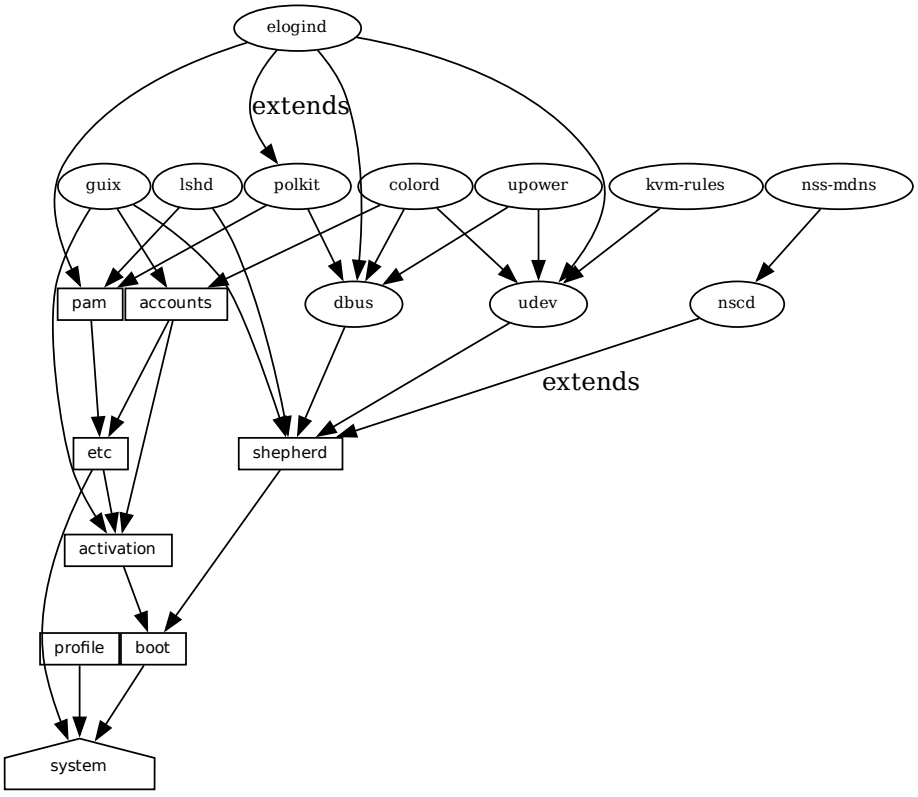
Las secciones anteriores muestran los servicios disponibles y cómo se pueden combinar en una declaración `operating-system`. ¿Pero cómo las definimos en primer lugar? ¿Y qué es un servicio en cualquier caso?

### 11.19.1 Composición de servicios

Definimos un *servicio* como, de manera genérica, algo que extiende la funcionalidad del sistema operativo. Habitualmente un servicio es un proceso—un *daemon*—iniciado cuando el sistema arranca: un servidor de shell seguro, un servidor Web, el daemon de construcción de Guix, etc. A veces un servicio es un daemon cuya ejecución puede ser iniciada por otro daemon—por ejemplo, un servidor FTP iniciado por `inetd` o un servicio D-Bus activado por `dbus-daemon`. De manera ocasional, un servicio no se puede asociar a un daemon. Por ejemplo, el servicio "account" recopila cuentas de usuaria y se asegura que existen cuando el sistema se ejecuta; el servicio "udev" recopila reglas de gestión de dispositivos y los pone a disposición del daemon eudev; el servicio `/etc` genera el contenido del directorio `/etc` del sistema.

Los servicios de Guix se conectan a través de *extensiones*. Por ejemplo, el servicio de shell seguro *extiende* Shepherd—el sistema de inicio, el cual se ejecuta como PID 1—proporcionando las líneas de órdenes para arrancar y parar el daemon de shell seguro (see Section 11.10.5 [Servicios de red], page 314); el servicio UPower extiende el servicio D-Bus proporcionando su especificación `.service`, y extiende el servicio udev al que proporciona reglas de gestión de dispositivos (see Section 11.10.9 [Servicios de escritorio], page 363); el servicio del daemon de Guix extiende Shepherd proporcionando las líneas de órdenes para arrancar y parar el daemon, y extiende el servicio de cuentas proporcionando una lista de cuentas de usuarias de construcción que necesita (see Section 11.10.1 [Servicios base], page 276).

Al fin y al cabo, los servicios y sus relaciones de "extensión" forman un grafo acíclico dirigido (GAD). Si representamos los servicios como cajas y las extensiones como flechas, un sistema típico puede proporcionar algo de este estilo:




En la base, podemos ver el *servicio del sistema*, el cual produce el directorio que contiene todo lo necesario para ejecutar y arrancar el sistema, como es devuelto por la orden `guix system build`. See Section 11.19.3 [Referencia de servicios], page 652, para aprender acerca de otros servicios mostrados aquí. See [system-extension-graph], page 643, para información sobre cómo generar esta representación para una definición particular de sistema operativo.

Técnicamente, las desarrolladoras pueden definir *tipos de servicio* para expresar estas relaciones. Puede haber cualquier número de servicios de un tipo dado en el sistema—por ejemplo, un sistema que ejecuta dos instancias del shell seguro GNU (lsh) tiene dos instancias de *lsh-service-type*, con parámetros diferentes.

La siguiente sección describe la interfaz programática para tipos de servicio y servicios.

### 11.19.2 Tipos de servicios y servicios

Un *tipo de servicio* es un nodo en el GAD descrito previamente. Empecemos con un ejemplo simple, el tipo de servicio para el daemon de construcción Guix (see Section 2.3 [Invocación de guix-daemon], page 12):

```
(define guix-service-type
  (service-type
   (name 'guix)
   (extensions
    (list (service-extension shepherd-root-service-type guix-shepherd-service)
          (service-extension account-service-type guix-accounts)
          (service-extension activation-service-type guix-activation)))
   (default-value (guix-configuration))))
```

Define tres cosas:

1. Un nombre, cuyo único propósito es facilitar la inspección y la depuración.
2. Una lista de *extensiones de servicio*, donde cada extensión designa el tipo de servicio a extender y un procedimiento que, dados los parámetros del servicio, devuelve una lista de objetos para extender el servicio de dicho tipo.

   Cada tipo de servicio tiene al menos una extensión de servicio. La única excepción es el *tipo de servicio de arranque*, que es el último servicio.
3. De manera opcional, un valor predeterminado para instancias de este tipo.

En este ejemplo, `guix-service-type` extiende tres servicios:

`shepherd-root-service-type`
: El procedimiento `guix-shepherd-service` define cómo se extiende el servicio de Shepherd. Es decir, devuelve un objeto `<shepherd-service>` que define cómo se arranca y para `guix-daemon` (see Section 11.19.4 [Servicios de Shepherd], page 657).

`account-service-type`
: `guix-accounts` crea la implementación de esta extensión para este servicio, la cual devuelve una lista de objetos `user-group` y `user-account` que representan las cuentas de usuarias de construcción (see Section 2.3 [Invocación de guix-daemon], page 12).

`activation-service-type`
: Aquí `guix-activation` es un procedimiento que devuelve una expresión-G, que es un fragmento de código a ejecutar en "tiempo de activación"—por ejemplo, cuando el servicio se arranca.

Un servicio de este tipo se puede instanciar de esta manera:

```
(service guix-service-type
         (guix-configuration
           (build-accounts 5)
           (extra-options '("--gc-keep-derivations"))))
```

El segundo parámetro a la forma `service` es un valor que representa los parámetros de esta instancia específica del servicio. See [guix-configuration-type], page 285, para información acerca del tipo de datos `guix-configuration`. Cuando se omite el valor, se usa el valor predeterminado por `guix-service-type`:

```
(service guix-service-type)
```

`guix-service-type` es bastante simple puesto que extiende otros servicios pero no es extensible a su vez.

El tipo de servicio para un servicio *extensible* puede tener esta forma:

```
(define udev-service-type
  (service-type (name 'udev)
                (extensions
                 (list (service-extension shepherd-root-service-type
                                          udev-shepherd-service)))

                (compose concatenate)       ;concatenate the list of rules
                (extend (lambda (config rules)
                          (udev-configuration
                           (inherit config)
                           (rules (append (udev-configuration-rules config)
                                          rules)))))))
```

This is the service type for the eudev device management daemon (`https://github.com/eudev-project/eudev`). Compared to the previous example, in addition to an extension of `shepherd-root-service-type`, we see two new fields:

`compose` Este es el procedimiento para *componer* la lista de extensiones en servicios de este tipo.

Los servicios pueden extender el servicio udev proporcionandole una lista de reglas; componemos estas extensiones mediante una simple concatenación.

`extend` Este procedimiento define cómo el valor del servicio se *extiende* con la composición de la extensión.

Las extensiones de udev se componen en una lista de reglas, pero el valor del servicio udev es en sí un registro `<udev-configuration>`. Por tanto aquí extendemos el registro agregando la lista de reglas que contiene al final de la lista de reglas que se contribuyeron.

`description`
Es una cadena que proporciona una descripción del tipo de servicio. Dicha cadena puede contener lenguaje de marcado Texinfo (see Section "Overview" in *GNU Texinfo*). La orden `guix system search` busca estas cadenas y las muestra (see Section 11.16 [Invocación de guix system], page 634).

Puede haber únicamente una instancia de un tipo de servicio extensible como `udev-service-type`. Si hubiese más, las especificaciones `service-extension` serían ambiguas.

¿Todavía aquí? La siguiente sección proporciona una referencia de la interfaz programática de los servicios.

### 11.19.3 Referencia de servicios

Ya hemos echado un vistazo a los tipos de servicio (see Section 11.19.2 [Tipos de servicios y servicios], page 650). Esta sección proporciona referencias sobre cómo manipular servicios y tipos de servicio. Esta interfaz se proporciona en el módulo `(gnu services)`.

`service` *tipo* [*valor*] [Procedimiento]
Devuelve un nuevo servicio de *tipo*, un objeto `<service-type>` (véase a continuación). *valor* puede ser cualquier objeto; representa los parámetros de esta instancia de servicio particular.

Cuando se omite *valor*, se usa el valor predeterminado especificado por *tipo*; si *type* no especifica ningún valor, se produce un error.

Por ejemplo, esto:

```
(service openssh-service-type)
```

es equivalente a esto:

```
(service openssh-service-type
         (openssh-configuration))
```

En ambos casos el resultado es una instancia de `openssh-service-type` con la configuración predeterminada.

`service?` *obj* [Procedimiento]
Devuelve verdadero si *obj* es un servicio.

`service-kind` *servicio* [Procedimiento]
Devuelve el tipo de *servicio*—es decir, un objeto `<service-type>`.

`service-value` *servicio* [Procedimiento]
Devuelve el valor asociado con *servicio*. Representa sus parámetros.

Este es un ejemplo de creación y manipulación de un servicio:

```
(define s
  (service nginx-service-type
           (nginx-configuration
            (nginx nginx)
            (log-directory log-directory)
            (run-directory run-directory)
            (file config-file))))

(service? s)
⇒ #t

(eq? (service-kind s) nginx-service-type)
⇒ #t
```

La forma `modify-services` proporciona una manera fácil de cambiar los parámetros de algunos servicios de una lista como `%base-services` (see Section 11.10.1 [Servicios base], page 276). Evalúa a una lista de servicios. Por supuesto, siempre puede usar operaciones estándar sobre listas como `map` y `fold` para hacerlo (see Section "SRFI-1" in *GNU Guile Reference Manual*); `modify-services` proporciona simplemente una forma más concisa para este patrón común.

`modify-services` *servicios* (*tipo variable* `=>` *cuerpo*) . . . [Special Form]
Modifica los servicios listados en *servicios* de acuerdo a las cláusulas proporcionadas. Cada cláusula tiene la forma:

```
(tipo variable => cuerpo)
```

donde *tipo* es un tipo de servicio—por ejemplo, `guix-service-type`—y *variable* es un identificador que se asocia dentro del *cuerpo* a los parámetros del servicio—por ejemplo, una instancia `guix-configuration`—del servicio original de dicho *ipo*.

El *cuerpo* deve evaluar a los nuevos parámetros del servicio, que serán usados para configurar el nuevo servicio. Este nuevo servicio reemplaza el original en la lista resultante. Debido a que los parámetros de servicio de un servicio se crean mediante el uso de `define-record-type*`, puede escribir un breve *cuerpo* que evalúe a los nuevos parámetros del servicio mediante el uso de la característica `inherit` que proporciona `define-record-type*` para heredar los valores antiguos.

Clauses can also have the following form:

```
(delete type)
```

Such a clause removes all services of the given *type* from *services*.

See Section 11.2 [Uso de la configuración del sistema], page 244, para ejemplos de uso.

A continuación se procede con la interfaz programática de los tipos de servicios. Es algo que debe conocer para escribir definiciones de nuevos servicios, pero no es cuando busque formas de personalizar su declaración `operating-system`.

`service-type` [Tipo de datos]

Esta es la representación de un *tipo de servicio* (see Section 11.19.2 [Tipos de servicios y servicios], page 650).

`name` Es un símbolo, usado únicamente para simplificar la inspección y la depuración.

`extensions`
Una lista no vacía de objetos `<service-extension>` (véase a continuación).

`compose` (predeterminado: `#f`)
Si es `#f`, entonces el tipo de servicio denota servicios que no pueden extenderse—es decir, servicios que no pueden recibir "valores" de otros servicios.

En otro caso, debe ser un procedimiento de un único parámetro. El procedimiento es invocado en `fold-services` y se le proporciona una lista de valores recibidos de las extensiones. Puede devolver un valor único.

`extend` (predeterminado: `#f`)
Si es `#f`, los servicios de este tipo no pueden extenderse.

En otro caso, debe ser un procedimiento que acepte dos parámetros: `fold-services` lo invoca, proporcionandole el valor inicial del servicio como el primer parámetro y el resultado de aplicar `compose` a los valores de las extensiones como segundo parámetro. Debe devolver un valor que es un parámetro válido para la instancia del servicio.

`description`
Una cadena, que posiblemente usa el lenguaje de marcas Texinfo, que describe en un par de frases el servicio. Esta cadena permite la búsqueda del servicio a través de `guix system search` (see Section 11.16 [Invocación de guix system], page 634).

`default-value` (predeterminado: `&no-default-value`)
: El valor predeterminado asociado a instancias de este tipo de servicio. Esto permite a las usuarias usar `service` sin su segundo parámetro:

```
(service tipo)
```

El servicio devuelto en este caso tiene el valor predeterminado especificado por *tipo*.

See Section 11.19.2 [Tipos de servicios y servicios], page 650, para ejemplos.

`service-extension` *tipo-deseado calcula* [Procedimiento]
: Devuelve una nueva extensión para servicios del tipo *tipo-deseado*. *calcula* debe ser un procedimiento de un único parámetro: es llamado en `fold-services`, proporcionandole el valor asociado con el servicio que proporciona la extensión; debe devolver un valor válido para el servicio deseado.

`service-extension?` *obj* [Procedimiento]
: Devuelve verdadero si *obj* es una expresión-G.

De manera ocasional, puede desear simplemente extender un servicio existente. Esto implica la creación de un nuevo tipo de servicio y la especificación de la extensión deseada, lo cual puede ser engorroso; el procedimiento `simple-service` proporciona un atajo para ello.

`simple-service` *nombre deseado valor* [Procedimiento]
: Devuelve un servicio que extiende *deseado* con *valor*. Esto funciona creando una instancia única del tipo de servicio *nombre*, de la cual el servicio devuelto es una instancia.

Por ejemplo, esto extiende mcron (see Section 11.10.2 [Ejecución de tareas programadas], page 297) con una tarea adicional:

```
(simple-service 'mi-tarea-mcron mcron-service-type
                #~(job '(next-hour (3)) "guix gc -F 2G"))
```

En el núcleo de la abstracción de los servicios se encuentra el procedimiento `fold-services`, que es responsable de la "compilación" de una lista de servicios en un único directorio que contiene todo lo necesario para arrancar y ejecutar el sistema—el directorio mostrado por la orden `guix system build` (see Section 11.16 [Invocación de guix system], page 634). En esencia, propaga las extensiones de servicios a través del grafo de servicios, actualizando los parámetros de cada nodo en el camino, hasta que alcanza el nodo raíz.

`fold-services` *servicios* [*#:target-type system-service-type*] [Procedimiento]
: Recorre *servicios* propagando sus extensiones hasta la raíz del tipo *target-type*; devuelve el servicio raíz tratado de la manera apropiada.

Por último, el módulo `(gnu services)` también define varios tipos esenciales de servicios, algunos de los cuales se enumeran a continuación.

`system-service-type` [Variable]
: Esta es la raíz del grafo de servicios. Produce el directorio del sistema como lo devuelve la orden `guix system build`.

`boot-service-type` [Variable]
El tipo del "servicio de arranque", que produce un *guión de arranque*. El guión de arranque es lo que ejecuta el disco inicial en RAM cuando se arranca.

`etc-service-type` [Variable]
El tipo del servicio `/etc`. Este servicio se usa para crear los archivos en `/etc` y puede extenderse proporcionandole pares nombre/archivo como estas:

```
(list `("issue" ,(plain-file "issue" "¡Bienvenida!\n")))
```

En este ejemplo, el efecto sería la adición de un archivo `/etc/issue` que apunte al archivo proporcionado.

`setuid-program-service-type` [Variable]
Type for the "setuid-program service". This service collects lists of executable file names, passed as gexps, and adds them to the set of setuid and setgid programs on the system (see Section 11.11 [Programas con setuid], page 620).

`profile-service-type` [Variable]
Tipo del servicio que genera el *perfil del sistema*—es decir, los programas en `/run/current-system/profile`. Otros servicios pueden extenderlo proporcionandole listas de paquetes a añadir al perfil del sistema.

`provenance-service-type` [Variable]
Es el tipo del servicio que registra los *metadatos de procedencia* en el sistema mismo. Crea varios archivos en `/run/current-system`:

`channels.scm`
: Es un "archivo de canales" que se le puede proporcionar a `guix pull -C` o `guix time-machine -C`, y que describe los canales usados para construir el sistema, si dicha información estaba disponible (see Chapter 6 [Canales], page 69).

`configuration.scm`
: Est es el archivo que se proporciona como valor para el servicio `provenance-service-type`. De manera predeterminada, `guix system reconfigure` proporciona automáticamente el archivo de configuración del SO que recibió en la línea de órdenes.

`provenance`
: Contiene la misma información que los otros dos archivos, pero en un formato que se puede procesar más fácilmente.

En general, estas dos piezas de información (canales y el archivo de configuración) son suficientes para reproducir el sistema operativo "desde las fuentes".

> **Advertencias:** Esta información es necesaria para reconstruir su sistema operativo, pero no siempre es suficiente. En particular, `configuration.scm` en sí es insuficiente si no está autocontenido—si hace referencia a módulos externos de Guile o a archivos adicionales. Si desea que `configuration.scm` sea autocontenido, le recomendamos que los módulos o archivos a los que haga referencia sean parte de un canal.

Además, la proveniencia de los metadatos es "silenciosa" en el sentido de que no cambia los bits que contiene su sistema, *excepto por los bits de los metadatos en sí*. Dos configuraciones de SO diferentes o conjuntos de canales pueden llevar al mismo sistema, bit a bit; cuando se usa `provenance-service-type`, estos dos sistemas tendrán distintos metadatos y por lo tanto distintos nombres de archivo en el almacén, lo que hace no tan trivial dicha comparación.

Este servicio se añade automáticamente a la configuración de su sistema operativo cuando usa `guix system reconfigure`, `guix system init` o `guix deploy`.

`linux-loadable-module-service-type` [Variable]

Type of the service that collects lists of packages containing kernel-loadable modules, and adds them to the set of kernel-loadable modules.

This service type is intended to be extended by other service types, such as below:

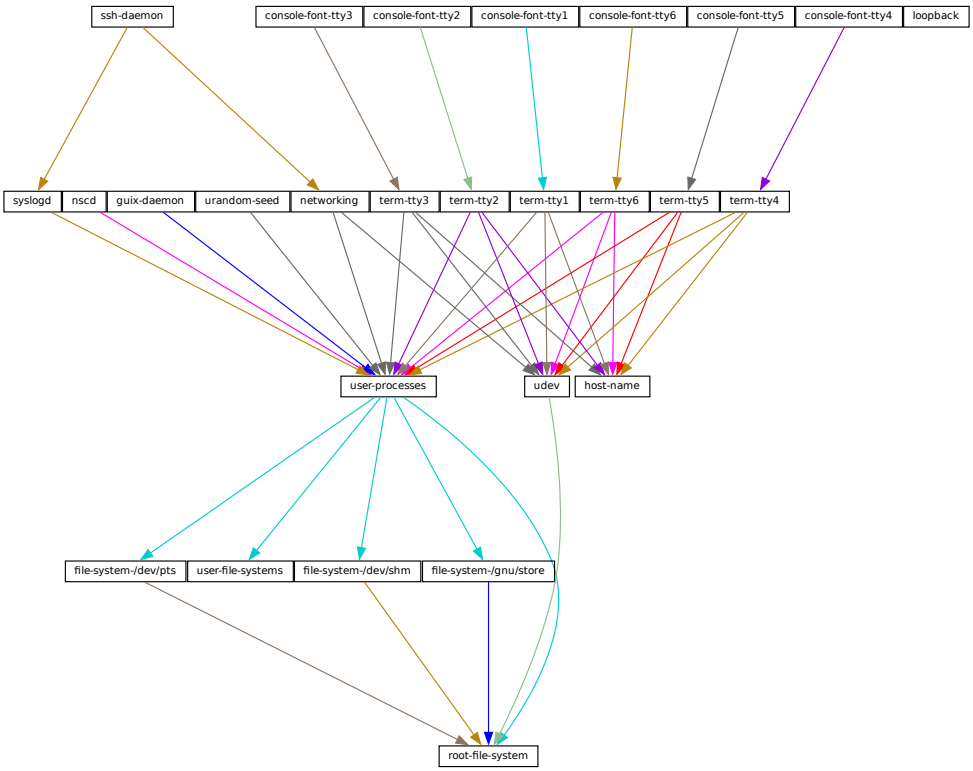
```
(simple-service 'installing-module
                linux-loadable-module-service-type
                (list module-to-install-1
                      module-to-install-2))
```

This does not actually load modules at bootup, only adds it to the kernel profile so that it *can* be loaded by other means.

### 11.19.4 Servicios de Shepherd

El módulo `(gnu services shepherd)` proporciona una forma de definir servicios gestionados por GNU Shepherd, que es el sistema de inicio—el primer proceso que se inicia cuando el sistema arranca, también conocido como PID 1 (see Section "Introduction" in *The GNU Shepherd Manual*).

Los servicios en Shepherd pueden depender de otros servicios. Por ejemplo, el daemon SSH puede tener que arrancarse tras el arranque del daemon syslog, lo cual a su vez puede suceder únicamente tras el montaje de todos los sistemas de archivos. El sistema operativo simple definido previamente (see Section 11.2 [Uso de la configuración del sistema], page 244) genera un grafo de servicios como este:




En realidad puede generar dicho grafo para cualquier definición de sistema operativo mediante el uso de la orden `guix system shepherd-graph` (see [system-shepherd-graph], page 643).

`%shepherd-root-service` es un objeto de servicio que representa el PID 1, del tipo `shepherd-root-service-type`; puede ser extendido proporcionandole listas de objetos `<shepherd-service>`.

`shepherd-service` [Tipo de datos]

El tipo de datos que representa un servicio gestionado por Shepherd.

`provision`

Una lista de símbolos que indican lo que proporciona el servicio.

Esto son nombres que pueden proporcionarse a `herd start`, `herd status` y órdenes similares (see Section "Invoking herd" in *The GNU Shepherd Manual*). See Section "Defining Services" in *The GNU Shepherd Manual*, para más detalles.

`requirement` (predeterminada: `'()`)

Lista de símbolos que indican los servicios Shepherd de los que este depende.

`one-shot?` (predeterminado: `#f`)
: Si este servicio es *one-shot*. Los servicios "one-shot" finalizan inmediatamente después de que su acción `start` se complete. See Section "Slots of services" in *The GNU Shepherd Manual*, para más información.

`respawn?` (predeterminado: `#t`)
: Indica si se debe reiniciar el servicio cuando se para, por ejemplo cuando el proceso subyacente muere.

`respawn-limit` (default: `#f`)
: Set a limit on how many times and how frequently a service may be restarted by Shepherd before it is disabled. See Section "Defining Services" in *The GNU Shepherd Manual*, for details.

`respawn-delay` (default: `#f`)
: When true, this is the delay in seconds before restarting a failed service.

`start`
`stop` (predeterminado: `#~(const #f)`)
: Los campos `start` y `stop` hacen referencia a las características de Shepherd de arranque y parada de procesos respectivamente (see Section "Service De- and Constructors" in *The GNU Shepherd Manual*). Se proporcionan como expresiones-G que se expandirán en el archivo de configuración de Shepherd (see Section 8.12 [Expresiones-G], page 167).

`actions` (predeterminadas: `'()`)
: Esta es la lista de objetos `shepherd-action` (véase a continuación) que definen las *acciones* permitidas por el servicio, además de las acciones estándar `start` y `stop`. Las acciones que se listan aquí estarán disponibles como ordenes de `herd`:

```
herd acción servicio [parámetros...]
```

`auto-start?` (predeterminado: `#t`)
: Determina si Shepherd debe iniciar este servicio de manera automática. Si es `#f` el servicio debe iniciarse manualmente con `herd start`.

`documentación`
: Una cadena de documentación, que se mostrará al ejecutar:

```
herd doc nombre-del-servicio
```

donde *nombre-del-servicio* es uno de los símbolos en `provision` (see Section "Invoking herd" in *The GNU Shepherd Manual*).

`modules` (predeterminados: `%default-modules`)
: Esta es la lista de módulos que deben estar dentro del ámbito cuando `start` y `stop` son evaluados.

The example below defines a Shepherd service that spawns `syslogd`, the system logger from the GNU Networking Utilities (see Section "syslogd invocation" in *GNU Inetutils*):

```
(let ((config (plain-file "syslogd.conf" "...")))
  (shepherd-service
    (documentation "Run the syslog daemon (syslogd).")
```

```
(provision '(syslogd))
(requirement '(user-processes))
(start #~(make-forkexec-constructor
          (list #$(file-append inetutils "/libexec/syslogd")
                "--rcfile" #$config)
          #:pid-file "/var/run/syslog.pid"))
(stop #~(make-kill-destructor))))
```

Key elements in this example are the `start` and `stop` fields: they are *staged* code snippets that use the `make-forkexec-constructor` procedure provided by the Shepherd and its dual, `make-kill-destructor` (see Section "Service De- and Constructors" in *The GNU Shepherd Manual*). The `start` field will have `shepherd` spawn `syslogd` with the given option; note that we pass `config` after `--rcfile`, which is a configuration file declared above (contents of this file are omitted). Likewise, the `stop` field tells how this service is to be stopped; in this case, it is stopped by making the `kill` system call on its PID. Code staging is achieved using G-expressions: `#~` stages code, while `#$` "escapes" back to host code (see Section 8.12 [Expresiones-G], page 167).

`shepherd-action` [Tipo de datos]

Este es el tipo de datos que define acciones adicionales implementadas por un servicio Shepherd (vea previamente).

`name` Símbolo que nombra la acción.

`documentación`
Esta es una cadena de documentación para la acción. Puede verse ejecutando:

```
herd doc servicio action acción
```

`procedure`
Debe ser una expresión-G que evalúa a un procedimiento de al menos un parámetro, el cual es el "valor de ejecución" del servicio (see Section "Slots of services" in *The GNU Shepherd Manual*).

El siguiente ejemplo define una acción llamada `di-hola` que saluda amablemente a la usuaria:

```
(shepherd-action
  (name 'di-hola)
  (documentation "¡Di hola!")
  (procedure #~(lambda (running . args)
                 (format #t "¡Hola, compa! parámetros: ~s\n"
                         args)
                 #t)))
```

Asumiendo que esta acción se añade al servicio `ejemplo`, puede ejecutar:

```
# herd di-hola ejemplo
¡Hola, compa! parámetros: ()
# herd di-hola ejemplo a b c
¡Hola, compa! parámetros: ("a" "b" "c")
```

Esta, como puede ver, es una forma un tanto sofisticada de decir hola. See Section "Defining Services" in *The GNU Shepherd Manual*, para más información sobre acciones.

`shepherd-configuration-action` [Procedure]

Return a `configuration` action to display *file*, which should be the name of the service's configuration file.

It can be useful to equip services with that action. For example, the service for the Tor anonymous router (see Section 11.10.5 [Servicios de red], page 314) is defined roughly like this:

```
(let ((torrc (plain-file "torrc" ...)))
  (shepherd-service
    (provision '(tor))
    (requirement '(user-processes loopback syslogd))

    (start #~(make-forkexec-constructor
              (list #$(file-append tor "/bin/tor") "-f" #$torrc)
              #:user "tor" #:group "tor"))
    (stop #~(make-kill-destructor))
    (actions (list (shepherd-configuration-action torrc)))
    (documentation "Run the Tor anonymous network overlay.")))
```

Thanks to this action, administrators can inspect the configuration file passed to `tor` with this shell command:

```
cat $(herd configuration tor)
```

This can come in as a handy debugging tool!

`shepherd-root-service-type` [Variable]

El tipo de servicio para el "servicio raíz" de Shepherd—es decir, PID 1.

This is the service type that extensions target when they want to create shepherd services (see Section 11.19.2 [Tipos de servicios y servicios], page 650, for an example). Each extension must pass a list of `<shepherd-service>`. Its value must be a `shepherd-configuration`, as described below.

`shepherd-configuration` [Data Type]

This data type represents the Shepherd's configuration.

`shepherd (default: shepherd)`
: The Shepherd package to use.

`services (default: '())`
: A list of `<shepherd-service>` to start. You should probably use the service extension mechanism instead (see Section 11.19.4 [Servicios de Shepherd], page 657).

The following example specifies the Shepherd package for the operating system:

```
(operating-system
  ;; ...
  (services (append (list openssh-service-type))
```

```
            ;; ...
            %desktop-services)
      ;; ...
      ;; Use own Shepherd package.
      (essential-services
       (modify-services (operating-system-default-essential-services
                         this-operating-system)
         (shepherd-root-service-type config => (shepherd-configuration
                                                (inherit config)
                                                (shepherd my-shepherd))))))
```

`%shepherd-root-service` [Variable]

Este servicio representa el PID 1.

### 11.19.5 Complex Configurations

Some programs might have rather complex configuration files or formats, and to make it easier to create Scheme bindings for these configuration files, you can use the utilities defined in the `(gnu services configuration)` module.

The main utility is the `define-configuration` macro, a helper used to define a Scheme record type (see Section "Record Overview" in *GNU Guile Reference Manual*). The fields from this Scheme record can be serialized using *serializers*, which are procedures that take some kind of Scheme value and translates them into another Scheme value or Section 8.12 [Expresiones-G], page 167.

`define-configuration` *name clause1 clause2 . . .* [Macro]

Create a record type named *`name`* that contains the fields found in the clauses.

A clause has the following form:

```
(field-name
 type-decl
 documentation
 option*
 ...)
```

*field-name* is an identifier that denotes the name of the field in the generated record.

*type-decl* is either *`type`* for fields that require a value to be set or `(`*`type default-value`*`)` otherwise.

*type* is the type of the value corresponding to *field-name*; since Guile is untyped, a predicate procedure—*`type`*`?`—will be called on the value corresponding to the field to ensure that the value is of the correct type. This means that if say, *type* is `package`, then a procedure named `package?` will be applied on the value to make sure that it is indeed a `<package>` object.

*default-value* is the default value corresponding to the field; if none is specified, the user is forced to provide a value when creating an object of the record type.

*documentation* is a string formatted with Texinfo syntax which should provide a description of what setting this field does.

*option** is one of the following subclauses:

`empty-serializer`
: Exclude this field from serialization.

`(serializer` *`serializer`*`)`
: *serializer* is the name of a procedure which takes two arguments, the first is the name of the field, and the second is the value corresponding to the field. The procedure should return a string or Section 8.12 [Expresiones-G], page 167, that represents the content that will be serialized to the configuration file. If none is specified, a procedure of the name `serialize-`*`type`* will be used.

  An example of a simple serializer procedure:

```
(define (serialize-boolean field-name value)
  (let ((value (if value "true" "false")))
    #~(string-append '#$field-name " = " #$value)))
```

`(sanitizer` *`sanitizer`*`)`
: *sanitizer* is a procedure which takes one argument, a user-supplied value, and returns a "sanitized" value for the field. If no sanitizer is specified, a default sanitizer is used, which raises an error if the value is not of type *type*.

  An example of a sanitizer for a field that accepts both strings and symbols looks like this:

```
(define (sanitize-foo value)
  (cond ((string? value) value)
        ((symbol? value) (symbol->string value))
        (else (error "bad value"))))
```

In some cases multiple different configuration records might be defined in the same file, but their serializers for the same type might have to be different, because they have different configuration formats. For example, the `serialize-boolean` procedure for the Getmail service would have to be different from the one for the Transmission service. To make it easier to deal with this situation, one can specify a serializer prefix by using the `prefix` literal in the `define-configuration` form. This means that one doesn't have to manually specify a custom *serializer* for every field.

```
(define (foo-serialize-string field-name value)
  ...)

(define (bar-serialize-string field-name value)
  ...)

(define-configuration foo-configuration
  (label
   string
   "The name of label.")
  (prefix foo-))
```

```
(define-configuration bar-configuration
  (ip-address
   string
   "The IPv4 address for this device.")
  (prefix bar-))
```

However, in some cases you might not want to serialize any of the values of the record, to do this, you can use the `no-serialization` literal. There is also the `define-configuration/no-serialization` macro which is a shorthand of this.

```
;; Nothing will be serialized to disk.
(define-configuration foo-configuration
  (field
   (string "test")
   "Some documentation.")
  (no-serialization))

;; The same thing as above.
(define-configuration/no-serialization bar-configuration
  (field
   (string "test")
   "Some documentation."))
```

`define-maybe` *type* [Macro]

Sometimes a field should not be serialized if the user doesn't specify a value. To achieve this, you can use the `define-maybe` macro to define a "maybe type"; if the value of a maybe type is left unset, or is set to the `%unset-value` value, then it will not be serialized.

When defining a "maybe type", the corresponding serializer for the regular type will be used by default. For example, a field of type `maybe-string` will be serialized using the `serialize-string` procedure by default, you can of course change this by specifying a custom serializer procedure. Likewise, the type of the value would have to be a string, or left unspecified.

```
(define-maybe string)

(define (serialize-string field-name value)
  ...)

(define-configuration baz-configuration
  (name
   ;; If set to a string, the `serialize-string' procedure will be used
   ;; to serialize the string.  Otherwise this field is not serialized.
   maybe-string
   "The name of this module."))
```

Like with `define-configuration`, one can set a prefix for the serializer name by using the `prefix` literal.

```
(define-maybe integer
  (prefix baz-))
```

```
(define (baz-serialize-integer field-name value)
  ...)
```

There is also the `no-serialization` literal, which when set means that no serializer will be defined for the "maybe type", regardless of whether its value is set or not. `define-maybe/no-serialization` is a shorthand for specifying the `no-serialization` literal.

```
(define-maybe/no-serialization symbol)

(define-configuration/no-serialization test-configuration
  (mode
   maybe-symbol
   "Docstring."))
```

`maybe-value-set?` *value* [Procedure]

Predicate to check whether a user explicitly specified the value of a maybe field.

`serialize-configuration` *configuration fields* [Procedure]

Return a G-expression that contains the values corresponding to the *fields* of *configuration*, a record that has been generated by `define-configuration`. The G-expression can then be serialized to disk by using something like `mixed-text-file`.

Once you have defined a configuration record, you will most likely also want to document it so that other people know to use it. To help with that, there are two procedures, both of which are documented below.

`generate-documentation` *documentation documentation-name* [Procedure]

Generate a Texinfo fragment from the docstrings in *documentation*, a list of `(label fields sub-documentation ...)`. *label* should be a symbol and should be the name of the configuration record. *fields* should be a list of all the fields available for the configuration record.

*sub-documentation* is a `(field-name configuration-name)` tuple. *field-name* is the name of the field which takes another configuration record as its value, and *configuration-name* is the name of that configuration record.

*sub-documentation* is only needed if there are nested configuration records. For example, the `getmail-configuration` record (see Section 11.10.13 [Servicios de correo], page 391) accepts a `getmail-configuration-file` record in one of its `rcfile` field, therefore documentation for `getmail-configuration-file` is nested in `getmail-configuration`.

```
(generate-documentation
  `((getmail-configuration ,getmail-configuration-fields
     (rcfile getmail-configuration-file))
    ...)
  'getmail-configuration)
```

*documentation-name* should be a symbol and should be the name of the configuration record.

`configuration->documentation` *configuration-symbol* [Procedure]
Take *configuration-symbol*, the symbol corresponding to the name used when defining a configuration record with `define-configuration`, and print the Texinfo documentation of its fields. This is useful if there aren't any nested configuration records since it only prints the documentation for the top-level fields.

As of right now, there is no automated way to generate documentation for configuration records and put them in the manual. Instead, every time you make a change to the docstrings of a configuration record, you have to manually call `generate-documentation` or `configuration->documentation`, and paste the output into the `doc/guix.texi` file.

Below is an example of a record type created using `define-configuration` and friends.

```
(use-modules (gnu services)
             (guix gexp)
             (gnu services configuration)
             (srfi srfi-26)
             (srfi srfi-1))

;; Turn field names, which are Scheme symbols into strings
(define (uglify-field-name field-name)
  (let ((str (symbol->string field-name)))
    ;; field? -> is-field
    (if (string-suffix? "?" str)
        (string-append "is-" (string-drop-right str 1))
        str)))

(define (serialize-string field-name value)
  #~(string-append #$(uglify-field-name field-name) " = " #$value "\n"))

(define (serialize-integer field-name value)
  (serialize-string field-name (number->string value)))

(define (serialize-boolean field-name value)
  (serialize-string field-name (if value "true" "false")))

(define (serialize-contact-name field-name value)
  #~(string-append "\n[" #$value "]\n"))

(define (list-of-contact-configurations? lst)
  (every contact-configuration? lst))

(define (serialize-list-of-contact-configurations field-name value)
  #~(string-append #$@(map (cut serialize-configuration <>
                                contact-configuration-fields)
                           value)))

(define (serialize-contacts-list-configuration configuration)
  (mixed-text-file
```

```
   "contactrc"
   #~(string-append "[Owner]\n"
                    #$(serialize-configuration
                       configuration contacts-list-configuration-fields))))

(define-maybe integer)
(define-maybe string)

(define-configuration contact-configuration
  (name
   string
   "The name of the contact."
   serialize-contact-name)
  (phone-number
   maybe-integer
   "The person's phone number.")
  (email
   maybe-string
   "The person's email address.")
  (married?
   boolean
   "Whether the person is married."))

(define-configuration contacts-list-configuration
  (name
   string
   "The name of the owner of this contact list.")
  (email
   string
   "The owner's email address.")
  (contacts
   (list-of-contact-configurations '())
   "A list of @code{contact-configuation} records which contain
information about all your contacts."))
```

A contacts list configuration could then be created like this:

```
(define my-contacts
  (contacts-list-configuration
   (name "Alice")
   (email "alice@example.org")
   (contacts
    (list (contact-configuration
           (name "Bob")
           (phone-number 1234)
           (email "bob@gnu.org")
           (married? #f))
          (contact-configuration
```

```
            (name "Charlie")
            (phone-number 0000)
            (married? #t))))))
```

After serializing the configuration to disk, the resulting file would look like this:

```
[owner]
name = Alice
email = alice@example.org

[Bob]
phone-number = 1234
email = bob@gnu.org
is-married = false

[Charlie]
phone-number = 0
is-married = true
```

# 12 System Troubleshooting Tips

Guix System allows rebooting into a previous generation should the last one be malfunctioning, which makes it quite robust against being broken irreversibly. This feature depends on GRUB being correctly functioning though, which means that if for whatever reasons your GRUB installation becomes corrupted during a system reconfiguration, you may not be able to easily boot into a previous generation. A technique that can be used in this case is to *chroot* into your broken system and reconfigure it from there. Such technique is explained below.

## 12.1 Chrooting into an existing system

This section details how to *chroot* to an already installed Guix System with the aim of reconfiguring it, for example to fix a broken GRUB installation. The process is similar to how it would be done on other GNU/Linux systems, but there are some Guix System particularities such as the daemon and profiles that make it worthy of explaining here.

1. Obtain a bootable image of Guix System. It is recommended the latest development snapshot so the kernel and the tools used are at least as as new as those of your installed system; it can be retrieved from the https://ci.guix.gnu.org (`https://ci.guix.gnu.org/search/latest/ISO-9660?query=spec:images+status:success+system:x86_64-linux+image.iso`) URL. Follow the see Section 3.3 [Instalación desde memoria USB y DVD], page 22, section for copying it to a bootable media.
2. Boot the image, and proceed with the graphical text-based installer until your network is configured. Alternatively, you could configure the network manually by following the [manual-installation-networking], page 26, section. If you get the error '`RTNETLINK answers: Operation not possible due to RF-kill`', try '`rfkill list`' followed by '`rfkill unblock 0`', where '`0`' is your device identifier (ID).
3. Switch to a virtual console (tty) if you haven't already by pressing simultaneously the *`Control`* + *`Alt`* + *`F4`* keys. Mount your file system at `/mnt`. Assuming your root partition is `/dev/sda2`, you would do:

   ```
   mount /dev/sda2 /mnt
   ```

4. Mount special block devices and Linux-specific directories:

   ```
   mount --rbind /proc /mnt/proc
   mount --rbind /sys /mnt/sys
   mount --rbind /dev /mnt/dev
   ```

   If your system is EFI-based, you must also mount the ESP partition. Assuming it is `/dev/sda1`, you can do so with:

   ```
   mount /dev/sda1 /mnt/boot/efi
   ```

5. Enter your system via chroot:

   ```
   chroot /mnt /bin/sh
   ```

6. Source the system profile as well as your *user* profile to setup the environment, where *user* is the user name used for the Guix System you are attempting to repair:

   ```
   source /var/guix/profiles/system/profile/etc/profile
   ```

```
source /home/user/.guix-profile/etc/profile
```

To ensure you are working with the Guix revision you normally would as your normal user, also source your current Guix profile:

```
source /home/user/.config/guix/current/etc/profile
```

7. Start a minimal `guix-daemon` in the background:

```
guix-daemon --build-users-group=guixbuild --disable-chroot &
```

8. Edit your Guix System configuration if needed, then reconfigure with:

```
guix system reconfigure your-config.scm
```

9. Finally, you should be good to reboot the system to test your fix.

# 13 Home Configuration

Guix supports declarative configuration of *home environments* by utilizing the configuration mechanism described in the previous chapter (see Section 11.19 [Definición de servicios], page 649), but for user's dotfiles and packages. It works both on Guix System and foreign distros and allows users to declare all the packages and services that should be installed and configured for the user. Once a user has written a file containing a `home-environment` record, such a configuration can be *instantiated* by an unprivileged user with the `guix home` command (see Section 13.4 [Invoking guix home], page 702).

The user's home environment usually consists of three basic parts: software, configuration, and state. Software in mainstream distros are usually installed system-wide, but with GNU Guix most software packages can be installed on a per-user basis without needing root privileges, and are thus considered part of the user's *home environment*. Packages on their own are not very useful in many cases, because often they require some additional configuration, usually config files that reside in `XDG_CONFIG_HOME` (`~/.config` by default) or other directories. Everything else can be considered state, like media files, application databases, and logs.

Using Guix for managing home environments provides a number of advantages:

- All software can be configured in one language (Guile Scheme), this gives users the ability to share values between configurations of different programs.
- A well-defined home environment is self-contained and can be created in a declarative and reproducible way—there is no need to grab external binaries or manually edit some configuration file.
- After every `guix home reconfigure` invocation, a new home environment generation will be created. This means that users can rollback to a previous home environment generation so they don't have to worry about breaking their configuration.
- It is possible to manage stateful data with Guix Home, this includes the ability to automatically clone Git repositories on the initial setup of the machine, and periodically running commands like `rsync` to sync data with another host. This functionality is still in an experimental stage, though.

## 13.1 Declaring the Home Environment

The home environment is configured by providing a `home-environment` declaration in a file that can be passed to the `guix home` command (see Section 13.4 [Invoking guix home], page 702). The easiest way to get started is by generating an initial configuration with `guix home import`:

```
guix home import ~/src/guix-config
```

The `guix home import` command reads some of the "dot files" such as `~/.bashrc` found in your home directory and copies them to the given directory, `~/src/guix-config` in this case; it also reads the contents of your profile, `~/.guix-profile`, and, based on that, it populates `~/src/guix-config/home-configuration.scm` with a Home configuration that resembles your current configuration.

A simple setup can include Bash and a custom text configuration, like in the example below. Don't be afraid to declare home environment parts, which overlaps with your current

dot files: before installing any configuration files, Guix Home will back up existing config files to a separate place in the home directory.

> **Nota:** It is highly recommended that you manage your shell or shells with Guix Home, because it will make sure that all the necessary scripts are sourced by the shell configuration file. Otherwise you will need to do it manually. (see Section 13.2 [Configuring the Shell], page 673).

```
(use-modules (gnu home)
             (gnu home services)
             (gnu home services shells)
             (gnu services)
             (gnu packages admin)
             (guix gexp))


(home-environment
 (packages (list htop))
 (services
  (list
   (service home-bash-service-type
            (home-bash-configuration
             (guix-defaults? #t)
             (bash-profile (list (plain-file "bash-profile" "\
export HISTFILE=$XDG_CACHE_HOME/.bash_history")))))

   (simple-service 'test-config
                   home-xdg-configuration-files-service-type
                   (list `("test.conf"
                           ,(plain-file "tmp-file.txt"
                                        "the content of
                                          ~/.config/test.conf")))))))
```

The `packages` field should be self-explanatory, it will install the list of packages into the user's profile. The most important field is `services`, it contains a list of *home services*, which are the basic building blocks of a home environment.

There is no daemon (at least not necessarily) related to a home service, a home service is just an element that is used to declare part of home environment and extend other parts of it. The extension mechanism discussed in the previous chapter (see Section 11.19 [Definición de servicios], page 649) should not be confused with Shepherd services (see Section 11.19.4 [Servicios de Shepherd], page 657). Using this extension mechanism and some Scheme code that glues things together gives the user the freedom to declare their own, very custom, home environments.

Once the configuration looks good, you can first test it in a throw-away "container":

```
guix home container config.scm
```

The command above spawns a shell where your home environment is running. The shell runs in a container, meaning it's isolated from the rest of the system, so it's a good way to

try out your configuration—you can see if configuration bits are missing or misbehaving, if daemons get started, and so on. Once you exit that shell, you're back to the prompt of your original shell "in the real world".

Once you have a configuration file that suits your needs, you can reconfigure your home by running:

```
guix home reconfigure config.scm
```

This "builds" your home environment and creates `~/.guix-home` pointing to it. Voilà!

> **Nota:** Make sure the operating system has elogind, systemd, or a similar mechanism to create the XDG run-time directory and has the `XDG_RUNTIME_DIR` variable set. Failing that, the `on-first-login` script will not execute anything, and processes like user Shepherd and its descendants will not start.

If you're using Guix System, you can embed your home configuration in your system configuration such that `guix system reconfigure` will deploy both the system *and* your home at once! See [guix-home-service-type], page 594, for how to do that.

## 13.2 Configuring the Shell

This section is safe to skip if your shell or shells are managed by Guix Home. Otherwise, read it carefully.

There are a few scripts that must be evaluated by a login shell to activate the home environment. The shell startup files only read by login shells often have `profile` suffix. For more information about login shells see Section "Invoking Bash" in *The GNU Bash Reference Manual* and see Section "Bash Startup Files" in *The GNU Bash Reference Manual*.

The first script that needs to be sourced is `setup-environment`, which sets all the necessary environment variables (including variables declared by the user) and the second one is `on-first-login`, which starts Shepherd for the current user and performs actions declared by other home services that extends `home-run-on-first-login-service-type`.

Guix Home will always create `~/.profile`, which contains the following lines:

```
HOME_ENVIRONMENT=$HOME/.guix-home
. $HOME_ENVIRONMENT/setup-environment
$HOME_ENVIRONMENT/on-first-login
```

This makes POSIX compliant login shells activate the home environment. However, in most cases this file won't be read by most modern shells, because they are run in non POSIX mode by default and have their own `*profile` startup files. For example Bash will prefer `~/.bash_profile` in case it exists and only if it doesn't will it fallback to `~/.profile`. Zsh (if no additional options are specified) will ignore `~/.profile`, even if `~/.zprofile` doesn't exist.

To make your shell respect `~/.profile`, add `. ~/.profile` or `source ~/.profile` to the startup file for the login shell. In case of Bash, it is `~/.bash_profile`, and in case of Zsh, it is `~/.zprofile`.

> **Nota:** This step is only required if your shell is *not* managed by Guix Home. Otherwise, everything will be done automatically.

## 13.3 Home Services

A *home service* is not necessarily something that has a daemon and is managed by Shepherd (see Section "Jump Start" in *The GNU Shepherd Manual*), in most cases it doesn't. It's a simple building block of the home environment, often declaring a set of packages to be installed in the home environment profile, a set of config files to be symlinked into `XDG_CONFIG_HOME` (`~/.config` by default), and environment variables to be set by a login shell.

There is a service extension mechanism (see Section 11.19.1 [Composición de servicios], page 649) which allows home services to extend other home services and utilize capabilities they provide; for example: declare mcron jobs (see *GNU Mcron*) by extending Section 13.3.3 [Mcron Home Service], page 684; declare daemons by extending Section 13.3.5 [Shepherd Home Service], page 686; add commands, which will be invoked on by the Bash by extending Section 13.3.2 [Shells Home Services], page 679.

A good way to discover available home services is using the `guix home search` command (see Section 13.4 [Invoking guix home], page 702). After the required home services are found, include its module with the `use-modules` form (see Section "Using Guile Modules" in *The GNU Guile Reference Manual*), or the `#:use-modules` directive (see Section "Creating Guile Modules" in *The GNU Guile Reference Manual*) and declare a home service using the `service` function, or extend a service type by declaring a new service with the `simple-service` procedure from `(gnu services)`.

### 13.3.1 Essential Home Services

There are a few essential home services defined in `(gnu home services)`, they are mostly for internal use and are required to build a home environment, but some of them will be useful for the end user.

`home-environment-variables-service-type` [Variable]

The service of this type will be instantiated by every home environment automatically by default, there is no need to define it, but someone may want to extend it with a list of pairs to set some environment variables.

```
(list ("ENV_VAR1" . "value1")
      ("ENV_VAR2" . "value2"))
```

The easiest way to extend a service type, without defining a new service type is to use the `simple-service` helper from `(gnu services)`.

```
(simple-service 'some-useful-env-vars-service
home-environment-variables-service-type
`(("LESSHISTFILE" . "$XDG_CACHE_HOME/.lesshst")
                  ("SHELL" . ,(file-append zsh "/bin/zsh"))
                  ("USELESS_VAR" . #f)
                  ("_JAVA_AWT_WM_NONREPARENTING" . #t)
                  ("LITERAL_VALUE" . ,(literal-string "${abc}"))))
```

If you include such a service in you home environment definition, it will add the following content to the `setup-environment` script (which is expected to be sourced by the login shell):

```
export LESSHISTFILE="$XDG_CACHE_HOME/.lesshst"
export SHELL="/gnu/store/2hsg15n644f0glrcbkb1kqknmmqdar03-zsh-5.8/bin/zsh"
```

```
export _JAVA_AWT_WM_NONREPARENTING
export LITERAL_VALUE='${abc}'
```

Notice that `literal-string` above lets us declare that a value is to be interpreted as a *literal string*, meaning that "special characters" such as the dollar sign will not be interpreted by the shell.

> **Nota:** Make sure that module `(gnu packages shells)` is imported with `use-modules` or any other way, this namespace contains the definition of the `zsh` package, which is used in the example above.

The association list (see Section "Association Lists" in *The GNU Guile Reference manual*) is a data structure containing key-value pairs, for `home-environment-variables-service-type` the key is always a string, the value can be a string, string-valued gexp (see Section 8.12 [Expresiones-G], page 167), file-like object (see Section 8.12 [Expresiones-G], page 167) or boolean. For gexps, the variable will be set to the value of the gexp; for file-like objects, it will be set to the path of the file in the store (see Section 8.9 [El almacén], page 157); for `#t`, it will export the variable without any value; and for `#f`, it will omit variable.

`home-profile-service-type` [Variable]

The service of this type will be instantiated by every home environment automatically, there is no need to define it, but you may want to extend it with a list of packages if you want to install additional packages into your profile. Other services, which need to make some programs available to the user will also extend this service type.

The extension value is just a list of packages:

```
(list htop vim emacs)
```

The same approach as `simple-service` (see Section 11.19.3 [Referencia de servicios], page 652) for `home-environment-variables-service-type` can be used here, too. Make sure that modules containing the specified packages are imported with `use-modules`. To find a package or information about its module use `guix search` (see Section 5.2 [Invocación de guix package], page 36). Alternatively, `specification->package` can be used to get the package record from a string without importing its related module.

There are few more essential services, but users are not expected to extend them.

`home-service-type` [Variable]

The root of home services DAG, it generates a folder, which later will be symlinked to `~/.guix-home`, it contains configurations, profile with binaries and libraries, and some necessary scripts to glue things together.

`home-run-on-first-login-service-type` [Variable]

The service of this type generates a Guile script, which is expected to be executed by the login shell. It is only executed if the special flag file inside `XDG_RUNTIME_DIR` hasn't been created, this prevents redundant executions of the script if multiple login shells are spawned.

It can be extended with a gexp. However, to autostart an application, users *should not* use this service, in most cases it's better to extend `home-shepherd-service-type` with a Shepherd service (see Section 11.19.4 [Servicios de Shepherd], page 657),

or extend the shell's startup file with the required command using the appropriate service type.

`home-files-service-type` [Variable]

The service of this type allows to specify a list of files, which will go to `~/.guix-home/files`, usually this directory contains configuration files (to be more precise it contains symlinks to files in `/gnu/store`), which should be placed in `$XDG_CONFIG_DIR` or in rare cases in `$HOME`. It accepts extension values in the following format:

```
`((".sway/config" ,sway-file-like-object)
  (".tmux.conf" ,(local-file "./tmux.conf")))
```

Each nested list contains two values: a subdirectory and file-like object. After building a home environment `~/.guix-home/files` will be populated with appropriate content and all nested directories will be created accordingly, however, those files won't go any further until some other service will do it. By default a `home-symlink-manager-service-type`, which creates necessary symlinks in home folder to files from `~/.guix-home/files` and backs up already existing, but clashing configs and other things, is a part of essential home services (enabled by default), but it's possible to use alternative services to implement more advanced use cases like read-only home. Feel free to experiment and share your results.

It is often the case that Guix Home users already have a setup for versioning their user configuration files (also known as *dot files*) in a single directory, and some way of automatically deploy changes to their user home.

The `home-dotfiles-service-type` from `(gnu home services dotfiles)` is designed to ease the way into using Guix Home for this kind of users, allowing them to point the service to their dotfiles directory without migrating them to Guix native configurations.

Please keep in mind that it is advisable to keep your dotfiles directories under version control, for example in the same repository where you'd track your Guix Home configuration.

There are two supported dotfiles directory layouts, for now. The `'plain` layout, which is structured as follows:

```
~$ tree -a ./dotfiles/
dotfiles/
 .gitconfig
 .gnupg
    gpg-agent.conf
    gpg.conf
 .guile
 .config
    guix
       channels.scm
    nixpkgs
        config.nix
 .nix-channels
 .tmux.conf
 .vimrc
```

This tree structure is installed as is to the home directory upon `guix home reconfigure`.

The `'stow` layout, which must follow the layout suggested by GNU Stow (`https://www.gnu.org/software/stow/`) presents an additional application specific directory layer, just like:

```
~$ tree -a ./dotfiles/
dotfiles/
 git
    .gitconfig
 gpg
    .gnupg
        gpg-agent.conf
        gpg.conf
 guile
    .guile
 guix
    .config
        guix
            channels.scm
 nix
    .config
       nixpkgs
           config.nix
    .nix-channels
 tmux
    .tmux.conf
 vim
     .vimrc

13 directories, 10 files
```

For an informal specification please refer to the Stow manual (see *Introduction*). This tree structure is installed following GNU Stow's logic to the home directory upon `guix home reconfigure`.

A suitable configuration with a `'plain` layout could be:

```
(home-environment
  ;; ...
  (services
    (service home-dotfiles-service-type
             (home-dotfiles-configuration
               (directories '("./dotfiles"))))))
```

The expected home directory state would then be:

```
.
 .config
    guix
       channels.scm
    nixpkgs
        config.nix
```

```
.gitconfig
.gnupg
   gpg-agent.conf
   gpg.conf
.guile
.nix-channels
.tmux.conf
.vimrc
```

`home-dotfiles-service-type` [Variable]

Return a service which is very similiar to `home-files-service-type` (and actually extends it), but designed to ease the way into using Guix Home for users that already track their dotfiles under some kind of version control. This service allows users to point Guix Home to their dotfiles directory and have their files automatically provisioned to their home directory, without migrating all of their dotfiles to Guix native configurations.

`home-dotfiles-configuration` [Data Type]

Available `home-dotfiles-configuration` fields are:

`source-directory` (default: `(current-source-directory)`) (type: string)
: The path where dotfile directories are resolved. By default dotfile directories are resolved relative the source location where `home-dotfiles-configuration` appears.

`layout` (default: `'plain`) (type: symbol)
: The intended layout of the specified `directory`. It can be either `'stow` or `'plain`.

`directories` (default: `'()`) (type: list-of-strings)
: The list of dotfiles directories where `home-dotfiles-service-type` will look for application dotfiles.

`packages` (type: maybe-list-of-strings)
: The names of a subset of the GNU Stow package layer directories. When provided the `home-dotfiles-service-type` will only provision dotfiles from this subset of applications. This field will be ignored if `layout` is set to `'plain`.

`excluded` (default: `'(".*~" ".*\\.swp" "\\.git" "\\.gitignore")`) (type: list-of-strings)
: The list of file patterns `home-dotfiles-service-type` will exclude while visiting each one of the `directories`.

`home-xdg-configuration-files-service-type` [Variable]

The service is very similar to `home-files-service-type` (and actually extends it), but used for defining files, which will go to `~/.guix-home/files/.config`, which will be symlinked to `$XDG_CONFIG_DIR` by `home-symlink-manager-service-type` (for example) during activation. It accepts extension values in the following format:

```
`(("sway/config" ,sway-file-like-object)
```

```
;; -> ~/.guix-home/files/.config/sway/config
;; -> $XDG_CONFIG_DIR/sway/config (by symlink-manager)
("tmux/tmux.conf" ,(local-file "./tmux.conf")))
```

`home-activation-service-type` [Variable]

The service of this type generates a guile script, which runs on every `guix home reconfigure` invocation or any other action, which leads to the activation of the home environment.

`home-symlink-manager-service-type` [Variable]

The service of this type generates a guile script, which will be executed during activation of home environment, and do a few following steps:

1. Reads the content of `files/` directory of current and pending home environments.
2. Cleans up all symlinks created by symlink-manager on previous activation. Also, sub-directories, which become empty also will be cleaned up.
3. Creates new symlinks the following way: It looks `files/` directory (usually defined with `home-files-service-type`, `home-xdg-configuration-files-service-type` and maybe some others), takes the files from `files/.config/` subdirectory and put respective links in `XDG_CONFIG_DIR`. For example symlink for `files/.config/sway/config` will end up in `$XDG_CONFIG_DIR/sway/config`. The rest files in `files/` outside of `files/.config/` subdirectory will be treated slightly different: symlink will just go to `$HOME`. `files/.some-program/config` will end up in `$HOME/.some-program/config`.
4. If some sub-directories are missing, they will be created.
5. If there is a clashing files on the way, they will be backed up.

symlink-manager is a part of essential home services and is enabled and used by default.

### 13.3.2 Shells

Shells play a quite important role in the environment initialization process, you can configure them manually as described in section Section 13.2 [Configuring the Shell], page 673, but the recommended way is to use home services listed below. It's both easier and more reliable.

Each home environment instantiates `home-shell-profile-service-type`, which creates a `~/.profile` startup file for all POSIX-compatible shells. This file contains all the necessary steps to properly initialize the environment, but many modern shells like Bash or Zsh prefer their own startup files, that's why the respective home services (`home-bash-service-type` and `home-zsh-service-type`) ensure that `~/.profile` is sourced by `~/.bash_profile` and `~/.zprofile`, respectively.

#### Shell Profile Service

`home-shell-profile-configuration` [Data Type]

Available `home-shell-profile-configuration` fields are:

`profile` (default: `'()`) (type: text-config)

`home-shell-profile` is instantiated automatically by `home-environment`, DO NOT create this service manually, it can only

be extended. `profile` is a list of file-like objects, which will go to `~/.profile`. By default `~/.profile` contains the initialization code which must be evaluated by the login shell to make home-environment's profile available to the user, but other commands can be added to the file if it is really necessary. In most cases shell's configuration files are preferred places for user's customizations. Extend home-shell-profile service only if you really know what you do.

## Bash Home Service

`home-bash-configuration` [Data Type]

Available `home-bash-configuration` fields are:

`package` (default: `bash`) (type: package)
The Bash package to use.

`guix-defaults?` (default: `#t`) (type: boolean)
Add sane defaults like reading `/etc/bashrc` and coloring the output of `ls` to the top of the `.bashrc` file.

`environment-variables` (default: `'()`) (type: alist)
Association list of environment variables to set for the Bash session. The rules for the `home-environment-variables-service-type` apply here (see Section 13.3.1 [Essential Home Services], page 674). The contents of this field will be added after the contents of the `bash-profile` field.

`aliases` (default: `'()`) (type: alist)
Association list of aliases to set for the Bash session. The aliases will be defined after the contents of the `bashrc` field has been put in the `.bashrc` file. The alias will automatically be quoted, so something like this:

```
'(("ls" . "ls -alF"))
```

turns into

```
alias ls="ls -alF"
```

`bash-profile` (default: `'()`) (type: text-config)
List of file-like objects, which will be added to `.bash_profile`. Used for executing user's commands at start of login shell (In most cases the shell started on tty just after login). `.bash_login` won't be ever read, because `.bash_profile` always present.

`bashrc` (default: `'()`) (type: text-config)
List of file-like objects, which will be added to `.bashrc`. Used for executing user's commands at start of interactive shell (The shell for interactive usage started by typing `bash` or by terminal app or any other program).

`bash-logout` (default: `'()`) (type: text-config)
List of file-like objects, which will be added to `.bash_logout`. Used for executing user's commands at the exit of login shell. It won't be read in some cases (if the shell terminates by exec'ing another process for example).

You can extend the Bash service by using the `home-bash-extension` configuration record, whose fields must mirror that of `home-bash-configuration` (see [home-bash-configuration], page 680). The contents of the extensions will be added to the end of the corresponding Bash configuration files (see Section "Bash Startup Files" in *The GNU Bash Reference Manual*.

For example, here is how you would define a service that extends the Bash service such that `~/.bash_profile` defines an additional environment variable, `PS1`:

```
(define bash-fancy-prompt-service
  (simple-service 'bash-fancy-prompt
                  home-bash-service-type
                  (home-bash-extension
                   (environment-variables
                    '(("PS1" . "\\u \\wλ "))))))
```

You would then add `bash-fancy-prompt-service` to the list in the `services` field of your `home-environment`. The reference of `home-bash-extension` follows.

`home-bash-extension` [Data Type]

Available `home-bash-extension` fields are:

`environment-variables` (default: `'()`) (type: alist)
: Additional environment variables to set. These will be combined with the environment variables from other extensions and the base service to form one coherent block of environment variables.

`aliases` (default: `'()`) (type: alist)
: Additional aliases to set. These will be combined with the aliases from other extensions and the base service.

`bash-profile` (default: `'()`) (type: text-config)
: Additional text blocks to add to `.bash_profile`, which will be combined with text blocks from other extensions and the base service.

`bashrc` (default: `'()`) (type: text-config)
: Additional text blocks to add to `.bashrc`, which will be combined with text blocks from other extensions and the base service.

`bash-logout` (default: `'()`) (type: text-config)
: Additional text blocks to add to `.bash_logout`, which will be combined with text blocks from other extensions and the base service.

## Zsh Home Service

`home-zsh-configuration` [Data Type]

Available `home-zsh-configuration` fields are:

`package` (default: `zsh`) (type: package)
: The Zsh package to use.

`xdg-flavor?` (default: `#t`) (type: boolean)
: Place all the configs to `$XDG_CONFIG_HOME/zsh`. Makes `~/.zshenv` to set `ZDOTDIR` to `$XDG_CONFIG_HOME/zsh`. Shell startup process will continue with `$XDG_CONFIG_HOME/zsh/.zshenv`.

`environment-variables` (default: `'()`) (type: alist)
: Association list of environment variables to set for the Zsh session.

`zshenv` (default: `'()`) (type: text-config)
: List of file-like objects, which will be added to `.zshenv`. Used for setting user's shell environment variables. Must not contain commands assuming the presence of tty or producing output. Will be read always. Will be read before any other file in `ZDOTDIR`.

`zprofile` (default: `'()`) (type: text-config)
: List of file-like objects, which will be added to `.zprofile`. Used for executing user's commands at start of login shell (In most cases the shell started on tty just after login). Will be read before `.zlogin`.

`zshrc` (default: `'()`) (type: text-config)
: List of file-like objects, which will be added to `.zshrc`. Used for executing user's commands at start of interactive shell (The shell for interactive usage started by typing `zsh` or by terminal app or any other program).

`zlogin` (default: `'()`) (type: text-config)
: List of file-like objects, which will be added to `.zlogin`. Used for executing user's commands at the end of starting process of login shell.

`zlogout` (default: `'()`) (type: text-config)
: List of file-like objects, which will be added to `.zlogout`. Used for executing user's commands at the exit of login shell. It won't be read in some cases (if the shell terminates by exec'ing another process for example).

## Inputrc Profile Service

The GNU Readline package (`https://tiswww.cwru.edu/php/chet/readline/rltop.html`) includes Emacs and vi editing modes, with the ability to customize the configuration with settings in the `~/.inputrc` file. With the `gnu home services shells` module, you can setup your readline configuration in a predictable manner, as shown below. For more information about configuring an `~/.inputrc` file, see Section "Readline Init File" in *GNU Readline*.

`home-inputrc-service-type` [Variable]
: This is the service to setup various `.inputrc` configurations. The settings in `.inputrc` are read by all programs which are linked with GNU Readline.

  Here is an example of a service and its configuration that you could add to the `services` field of your `home-environment`:

```
(service home-inputrc-service-type
         (home-inputrc-configuration
           (key-bindings
            `(("Control-l" . "clear-screen")))
           (variables
            `(("bell-style" . "visible")
              ("colored-completion-prefix" . #t)
              ("editing-mode" . "vi")
```

```
   ("show-mode-in-prompt" . #t)))
 (conditional-constructs
  `(("$if mode=vi" .
     ,(home-inputrc-configuration
        (variables
         `(("colored-stats" . #t)
           ("enable-bracketed-paste" . #t)))))
    ("$else" .
     ,(home-inputrc-configuration
        (variables
         `(("show-all-if-ambiguous" . #t)))))
    ("endif" . #t)
    ("$include" . "/etc/inputrc")
    ("$include" . ,(file-append
                    (specification->package "readline")
                    "/etc/inputrc"))))))
```

The example above starts with a combination of `key-bindings` and `variables`. The `conditional-constructs` show how it is possible to add conditionals and includes. In the example above `colored-stats` is only enabled if the editing mode is `vi` style, and it also reads any additional configuration located in `/etc/inputrc` or in `/gnu/store/...-readline/etc/inputrc`.

The value associated with a `home-inputrc-service-type` instance must be a `home-inputrc-configuration` record, as described below.

`home-inputrc-configuration` [Data Type]

Available `home-inputrc-configuration` fields are:

`key-bindings` (default: `'()`) (type: alist)

Association list of readline key bindings to be added to the `~/.inputrc` file.

```
'((\"Control-l\" . \"clear-screen\"))
```

turns into

```
Control-l: clear-screen
```

`variables` (default: `'()`) (type: alist)

Association list of readline variables to set.

```
'((\"bell-style\" . \"visible\")
  (\"colored-completion-prefix\" . #t))
```

turns into

```
set bell-style visible
set colored-completion-prefix on
```

`conditional-constructs` (default: `'()`) (type: alist)

Association list of conditionals to add to the initialization file. This includes `$if`, `else`, `endif` and `include` and they receive a value of another `home-inputrc-configuration`.

```
(conditional-constructs
```

```
`((\"$if mode=vi\" .
    ,(home-inputrc-configuration
       (variables
        `((\"show-mode-in-prompt\" . #t)))))
  (\"$else\" .
    ,(home-inputrc-configuration
       (key-bindings
        `((\"Control-l\" . \"clear-screen\")))))
  (\"$endif\" . #t)))
```

turns into

```
$if mode=vi
set show-mode-in-prompt on
$else
Control-l: clear-screen
$endif
```

`extra-content` (default: `""`) (type: text-config)
: Extra content appended as-is to the configuration file. Run `man readline` for more information about all the configuration options.

### 13.3.3 Scheduled User's Job Execution

The `(gnu home services mcron)` module provides an interface to GNU mcron, a daemon to run jobs at scheduled times (see *GNU mcron*). The information about system's mcron is applicable here (see Section 11.10.2 [Ejecución de tareas programadas], page 297), the only difference for home services is that they have to be declared in a `home-environment` record instead of an `operating-system` record.

`home-mcron-service-type` [Variable]
: This is the type of the `mcron` home service, whose value is a `home-mcron-configuration` object. It allows to manage scheduled tasks.

  This service type can be the target of a service extension that provides additional job specifications (see Section 11.19.1 [Composición de servicios], page 649). In other words, it is possible to define services that provide additional mcron jobs to run.

`home-mcron-configuration` [Data Type]
: Available `home-mcron-configuration` fields are:

  `mcron` (default: `mcron`) (type: file-like)
  : El paquete mcron usado.

  `jobs` (default: `'()`) (type: list-of-gexps)
  : This is a list of gexps (see Section 8.12 [Expresiones-G], page 167), where each gexp corresponds to an mcron job specification (see Section "Syntax" in *GNU mcron*).

  `log?` (default: `#t`) (type: boolean)
  : Log messages to standard output.

  `log-format` (default: `"~1@*~a ~a: ~a~%"`) (type: string)
  : `(ice-9 format)` format string for log messages. The default value produces messages like "'*pid name*: *message*"' (see Section "Invoking

mcron" in *GNU mcron*). Each message is also prefixed by a timestamp by GNU Shepherd.

### 13.3.4 Power Management Home Services

The `(gnu home services pm)` module provides home services pertaining to battery power.

`home-batsignal-service-type` [Variable]

Service for `batsignal`, a program that monitors battery levels and warns the user through desktop notifications when their battery is getting low. You can also configure a command to be run when the battery level passes a point deemed "dangerous". This service is configured with the `home-batsignal-configuration` record.

`home-batsignal-configuration` [Data Type]

Data type representing the configuration for batsignal.

`warning-level` (default: `15`)
: The battery level to send a warning message at.

`warning-message` (default: `#f`)
: The message to send as a notification when the battery level reaches the `warning-level`. Setting to `#f` uses the default message.

`critical-level` (default: `5`)
: The battery level to send a critical message at.

`critical-message` (default: `#f`)
: The message to send as a notification when the battery level reaches the `critical-level`. Setting to `#f` uses the default message.

`danger-level` (default: `2`)
: The battery level to run the `danger-command` at.

`danger-command` (default: `#f`)
: The command to run when the battery level reaches the `danger-level`. Setting to `#f` disables running the command entirely.

`full-level` (default: `#f`)
: The battery level to send a full message at. Setting to `#f` disables sending the full message entirely.

`full-message` (default: `#f`)
: The message to send as a notification when the battery level reaches the `full-level`. Setting to `#f` uses the default message.

`batteries` (default: `'()`)
: The batteries to monitor. Setting to `'()` tries to find batteries automatically.

`poll-delay` (default: `60`)
: The time in seconds to wait before checking the batteries again.

`icon` (default: `#f`)
: A file-like object to use as the icon for battery notifications. Setting to `#f` disables notification icons entirely.

`notifications?` (default: `#t`)
: Whether to send any notifications.

`notifications-expire?` (default: `#f`)
: Whether notifications sent expire after a time.

`notification-command` (default: `#f`)
: Command to use to send messages. Setting to `#f` sends a notification through `libnotify`.

`ignore-missing?` (default: `#f`)
: Whether to ignore missing battery errors.

### 13.3.5 Managing User Daemons

The `(gnu home services shepherd)` module supports the definitions of per-user Shepherd services (see Section "Introduction" in *The GNU Shepherd Manual*). You extend `home-shepherd-service-type` with new services; Guix Home then takes care of starting the `shepherd` daemon for you when you log in, which in turns starts the services you asked for.

`home-shepherd-service-type` [Variable]
: The service type for the userland Shepherd, which allows one to manage long-running processes or one-shot tasks. User's Shepherd is not an init process (PID 1), but almost all other information described in (see Section 11.19.4 [Servicios de Shepherd], page 657) is applicable here too.

  This is the service type that extensions target when they want to create shepherd services (see Section 11.19.2 [Tipos de servicios y servicios], page 650, for an example). Each extension must pass a list of `<shepherd-service>`. Its value must be a `home-shepherd-configuration`, as described below.

`home-shepherd-configuration` [Data Type]
: This data type represents the Shepherd's configuration.

  `shepherd (default: shepherd)`
  : The Shepherd package to use.

  `auto-start? (default: #t)`
  : Whether or not to start Shepherd on first login.

  `services (default: '())`
  : A list of `<shepherd-service>` to start. You should probably use the service extension mechanism instead (see Section 11.19.4 [Servicios de Shepherd], page 657).

### 13.3.6 Secure Shell

The OpenSSH package (`https://www.openssh.com`) includes a client, the `ssh` command, that allows you to connect to remote machines using the SSH (secure shell) protocol. With the `(gnu home services ssh)` module, you can set up OpenSSH so that it works in a predictable fashion, almost independently of state on the local machine. To do that, you instantiate `home-openssh-service-type` in your Home configuration, as explained below.

`home-openssh-service-type` [Variable]

This is the type of the service to set up the OpenSSH client. It takes care of several things:

- providing a `~/.ssh/config` file based on your configuration so that `ssh` knows about hosts you regularly connect to and their associated parameters;
- providing a `~/.ssh/authorized_keys`, which lists public keys that the local SSH server, `sshd`, may accept to connect to this user account;
- optionally providing a `~/.ssh/known_hosts` file so that `ssh` can authenticate hosts you connect to.

Here is an example of a service and its configuration that you could add to the `services` field of your `home-environment`:

```
(service home-openssh-service-type
         (home-openssh-configuration
          (hosts
           (list (openssh-host (name "ci.guix.gnu.org")
                               (user "charlie"))
                 (openssh-host (name "chbouib")
                               (host-name "chbouib.example.org")
                               (user "supercharlie")
                               (port 10022))))
          (authorized-keys (list (local-file "alice.pub")))))
```

The example above lists two hosts and their parameters. For instance, running `ssh chbouib` will automatically connect to `chbouib.example.org` on port 10022, logging in as user '`supercharlie`'. Further, it marks the public key in `alice.pub` as authorized for incoming connections.

The value associated with a `home-openssh-service-type` instance must be a `home-openssh-configuration` record, as describe below.

`home-openssh-configuration` [Data Type]

This is the datatype representing the OpenSSH client and server configuration in one's home environment. It contains the following fields:

`hosts` (default: `'()`)

A list of `openssh-host` records specifying host names and associated connection parameters (see below). This host list goes into `~/.ssh/config`, which `ssh` reads at startup.

`known-hosts` (default: `*unspecified*`)

This must be either:

- `*unspecified*`, in which case `home-openssh-service-type` leaves it up to `ssh` and to the user to maintain the list of known hosts at `~/.ssh/known_hosts`, or
- a list of file-like objects, in which case those are concatenated and emitted as `~/.ssh/known_hosts`.

The `~/.ssh/known_hosts` contains a list of host name/host key pairs that allow `ssh` to authenticate hosts you connect to and to detect possible

impersonation attacks. By default, `ssh` updates it in a *TOFU, trust-on-first-use* fashion, meaning that it records the host's key in that file the first time you connect to it. This behavior is preserved when `known-hosts` is set to `*unspecified*`.

If you instead provide a list of host keys upfront in the `known-hosts` field, your configuration becomes self-contained and stateless: it can be replicated elsewhere or at another point in time. Preparing this list can be relatively tedious though, which is why `*unspecified*` is kept as a default.

`authorized-keys` (default: `#false`)
The default `#false` value means: Leave any `~/.ssh/authorized_keys` file alone. Otherwise, this must be a list of file-like objects, each of which containing an SSH public key that should be authorized to connect to this machine.

Concretely, these files are concatenated and made available as `~/.ssh/authorized_keys`. If an OpenSSH server, `sshd`, is running on this machine, then it *may* take this file into account: this is what `sshd` does by default, but be aware that it can also be configured to ignore it.

`add-keys-to-agent` (default: `no`)
This string specifies whether keys should be automatically added to a running ssh-agent. If this option is set to `yes` and a key is loaded from a file, the key and its passphrase are added to the agent with the default lifetime, as if by `ssh-add`. If this option is set to `ask`, `ssh` will require confirmation. If this option is set to `confirm`, each use of the key must be confirmed. If this option is set to `no`, no keys are added to the agent. Alternately, this option may be specified as a time interval to specify the key's lifetime in `ssh-agent`, after which it will automatically be removed. The argument must be `no`, `yes`, `confirm` (optionally followed by a time interval), `ask` or a time interval.

`openssh-host` [Data Type]

Available `openssh-host` fields are:

`name` (type: string)
Name of this host declaration. A `openssh-host` must define only `name` or `match-criteria`. Use host-name `\"*\"` for top-level options.

`host-name` (type: maybe-string)
Host name—e.g., `"foo.example.org"` or `"192.168.1.2"`.

`match-criteria` (type: maybe-match-criteria)
When specified, this string denotes the set of hosts to which the entry applies, superseding the `host-name` field. Its first element must be all or one of `ssh-match-keywords`. The rest of the elements are arguments for the keyword, or other criteria. A `openssh-host` must define only `name` or `match-criteria`. Other host configuration options will apply to all hosts matching `match-criteria`.

`address-family` (type: maybe-address-family)
: Address family to use when connecting to this host: one of `AF_INET` (for IPv4 only), `AF_INET6` (for IPv6 only). Additionally, the field can be left unset to allow any address family.

`identity-file` (type: maybe-string)
: The identity file to use—e.g., `"/home/charlie/.ssh/id_ed25519"`.

`port` (type: maybe-natural-number)
: TCP port number to connect to.

`user` (type: maybe-string)
: User name on the remote host.

`forward-x11?` (type: maybe-boolean)
: Whether to forward remote client connections to the local X11 graphical display.

`forward-x11-trusted?` (type: maybe-boolean)
: Whether remote X11 clients have full access to the original X11 graphical display.

`forward-agent?` (type: maybe-boolean)
: Whether the authentication agent (if any) is forwarded to the remote machine.

`compression?` (type: maybe-boolean)
: Whether to compress data in transit.

`proxy` (type: maybe-proxy-command-or-jump-list)
: The command to use to connect to the server or a list of SSH hosts to jump through before connecting to the server. The field may be set to either a `proxy-command` or a list of `proxy-jump` records.

  As an example, a `proxy-command` to connect via an HTTP proxy at 192.0.2.0 would be constructed with: `(proxy-command "nc -X connect -x 192.0.2.0:8080 %h %p")`.

  `proxy-jump` [Data Type]
  : Available `proxy-jump` fields are:

    `user` (type: maybe-string)
    : User name on the remote host.

    `host-name` (type: string)
    : Host name—e.g., `foo.example.org` or `192.168.1.2`.

    `port` (type: maybe-natural-number)
    : TCP port number to connect to.

`host-key-algorithms` (type: maybe-string-list)
: The list of accepted host key algorithms—e.g., `'("ssh-ed25519")`.

`accepted-key-types` (type: maybe-string-list)
: The list of accepted user public key types.

`extra-content` (default: `""`) (type: raw-configuration-string)
: Extra content appended as-is to this `Host` block in `~/.ssh/config`.

The `parcimonie` service runs a daemon that slowly refreshes a GnuPG public key from a keyserver. It refreshes one key at a time; between every key update parcimonie sleeps a random amount of time, long enough for the previously used Tor circuit to expire. This process is meant to make it hard for an attacker to correlate the multiple key update.

As an example, here is how you would configure `parcimonie` to refresh the keys in your GnuPG keyring, as well as those keyrings created by Guix, such as when running `guix import`:

```
(service home-parcimonie-service-type
         (home-parcimonie-configuration
           (refresh-guix-keyrings? #t)))
```

This assumes that the Tor anonymous routing daemon is already running on your system. On Guix System, this can be achieved by setting up `tor-service-type` (see Section 11.10.5 [Servicios de red], page 314).

The service reference is given below.

`parcimonie-service-type` [Variable]
: This is the service type for `parcimonie` (Parcimonie's web site (`https://salsa.debian.org/intrigeri/parcimonie`)). Its value must be a `home-parcimonie-configuration`, as shown below.

`home-parcimonie-configuration` [Data Table]
: Available `home-parcimonie-configuration` fields are:

  `parcimonie` (default: `parcimonie`) (type: file-like)
  : The parcimonie package to use.

  `verbose?` (default: `#f`) (type: boolean)
  : Whether to have more verbose logging from the service.

  `gnupg-already-torified?` (default: `#f`) (type: boolean)
  : Whether GnuPG is already configured to pass all traffic through Tor (`https://torproject.org`).

  `refresh-guix-keyrings?` (default: `#f`) (type: boolean)
  : Guix creates a few keyrings in the *$XDG_CONFIG_DIR*, such as when running `guix import` (see Section 9.5 [Invocación de guix import], page 198). Setting this to `#t` will also refresh any keyrings which Guix has created.

  `extra-content` (default: `#f`) (type: raw-configuration-string)
  : Raw content to add to the parcimonie command.

The OpenSSH package (`https://www.openssh.com`) includes a daemon, the `ssh-agent` command, that manages keys to connect to remote machines using the SSH (secure shell) protocol. With the `(gnu home services ssh)` service, you can configure the OpenSSH ssh-agent to run upon login. See Section 13.3.7 [GNU Privacy Guard], page 691, for an alternative to OpenSSH's `ssh-agent`.

Here is an example of a service and its configuration that you could add to the `services` field of your `home-environment`:

```
(service home-ssh-agent-service-type
         (home-ssh-agent-configuration
          (extra-options '("-t" "1h30m"))))
```

`home-ssh-agent-service-type` [Variable]
This is the type of the `ssh-agent` home service, whose value is a `home-ssh-agent-configuration` object.

`home-ssh-agent-configuration` [Data Type]
Available `home-ssh-agent-configuration` fields are:

`openssh` (default: `openssh`) (type: file-like)
The OpenSSH package to use.

`socket-directory` (default: `XDG_RUNTIME_DIR/ssh-agent"`) (type: gexp)
The directory to write the ssh-agent's `socket` file.

`extra-options` (predeterminadas: `'()`)
Extra options will be passed to `ssh-agent`, please run `man ssh-agent` for more information.

### 13.3.7 GNU Privacy Guard

The `(gnu home services gnupg)` module provides services that help you set up the GNU Privacy Guard, also known as GnuPG or GPG, in your home environment.

The `gpg-agent` service configures and sets up GPG's agent, the program that is responsible for managing OpenPGP private keys and, optionally, OpenSSH (secure shell) private keys (see Section "Invoking GPG-AGENT" in *Using the GNU Privacy Guard*).

As an example, here is how you would configure `gpg-agent` with SSH support such that it uses the Emacs-based Pinentry interface when prompting for a passphrase:

```
(service home-gpg-agent-service-type
         (home-gpg-agent-configuration
          (pinentry-program
           (file-append pinentry-emacs "/bin/pinentry-emacs"))
          (ssh-support? #t)))
```

The service reference is given below.

`home-gpg-agent-service-type` [Variable]
This is the service type for `gpg-agent` (see Section "Invoking GPG-AGENT" in *Using the GNU Privacy Guard*). Its value must be a `home-gpg-agent-configuration`, as shown below.

`home-gpg-agent-configuration` [Data Type]
Available `home-gpg-agent-configuration` fields are:

`gnupg` (default: `gnupg`) (type: file-like)
The GnuPG package to use.

`pinentry-program` (type: file-like)
: Pinentry program to use. Pinentry is a small user interface that `gpg-agent` delegates to anytime it needs user input for a passphrase or PIN (personal identification number) (see *Using the PIN-Entry*).

`ssh-support?` (default: `#f`) (type: boolean)
: Whether to enable SSH (secure shell) support. When true, `gpg-agent` acts as a drop-in replacement for OpenSSH's `ssh-agent` program, taking care of OpenSSH secret keys and directing passphrase requests to the chosen Pinentry program.

`default-cache-ttl` (default: `600`) (type: integer)
: Time a cache entry is valid, in seconds.

`max-cache-ttl` (default: `7200`) (type: integer)
: Maximum time a cache entry is valid, in seconds. After this time a cache entry will be expired even if it has been accessed recently.

`default-cache-ttl-ssh` (default: `1800`) (type: integer)
: Time a cache entry for SSH keys is valid, in seconds.

`max-cache-ttl-ssh` (default: `7200`) (type: integer)
: Maximum time a cache entry for SSH keys is valid, in seconds.

`extra-content` (default: `""`) (type: raw-configuration-string)
: Raw content to add to the end of `~/.gnupg/gpg-agent.conf`.

### 13.3.8 Desktop Home Services

The `(gnu home services desktop)` module provides services that you may find useful on "desktop" systems running a graphical user environment such as Xorg.

`home-x11-service-type` [Variable]
: This is the service type representing the X Window graphical display server (also referred to as "X11").

  X Window is necessarily started by a system service; on Guix System, starting it is the responsibility of `gdm-service-type` and similar services (see Section 11.10.7 [Sistema X Window], page 340). At the level of Guix Home, as an unprivileged user, we cannot start X Window; all we can do is check whether it is running. This is what this service does.

  As a user, you probably don't need to worry or explicitly instantiate `home-x11-service-type`. Services that require an X Window graphical display, such as `home-redshift-service-type` below, instantiate it and depend on its corresponding `x11-display` Shepherd service (see Section 13.3.5 [Shepherd Home Service], page 686).

  When X Window is running, the `x11-display` Shepherd service starts and sets the `DISPLAY` environment variable of the `shepherd` process, using its original value if it was already set; otherwise, it fails to start.

  The service can also be forced to use a given value for `DISPLAY`, like so:

```
herd start x11-display :3
```

  In the example above, `x11-display` is instructed to set `DISPLAY` to `:3`.

`home-redshift-service-type` [Variable]

This is the service type for Redshift (`https://github.com/jonls/redshift`), a program that adjusts the display color temperature according to the time of day. Its associated value must be a `home-redshift-configuration` record, as shown below.

A typical configuration, where we manually specify the latitude and longitude, might look like this:

```
(service home-redshift-service-type
         (home-redshift-configuration
          (location-provider 'manual)
          (latitude 35.81)    ;northern hemisphere
          (longitude -0.80))) ;west of Greenwich
```

`home-redshift-configuration` [Data Type]

Available `home-redshift-configuration` fields are:

`redshift` (default: `redshift`) (type: file-like)
Redshift package to use.

`location-provider` (default: `geoclue2`) (type: symbol)
Geolocation provider—`'manual` or `'geoclue2`. In the former case, you must also specify the `latitude` and `longitude` fields so Redshift can determine daytime at your place. In the latter case, the Geoclue system service must be running; it will be queried for location information.

`adjustment-method` (default: `randr`) (type: symbol)
Color adjustment method.

`daytime-temperature` (default: `6500`) (type: integer)
Daytime color temperature (kelvins).

`nighttime-temperature` (default: `4500`) (type: integer)
Nighttime color temperature (kelvins).

`daytime-brightness` (type: maybe-inexact-number)
Daytime screen brightness, between 0.1 and 1.0, or left unspecified.

`nighttime-brightness` (type: maybe-inexact-number)
Nighttime screen brightness, between 0.1 and 1.0, or left unspecified.

`latitude` (type: maybe-inexact-number)
Latitude, when `location-provider` is `'manual`.

`longitude` (type: maybe-inexact-number)
Longitude, when `location-provider` is `'manual`.

`dawn-time` (type: maybe-string)
Custom time for the transition from night to day in the morning—`"HH:MM"` format. When specified, solar elevation is not used to determine the daytime/nighttime period.

`dusk-time` (type: maybe-string)
Likewise, custom time for the transition from day to night in the evening.

`extra-content` (default: `""`) (type: raw-configuration-string)
: Extra content appended as-is to the Redshift configuration file. Run `man redshift` for more information about the configuration file format.

`home-dbus-service-type` [Variable]
This is the service type for running a session-specific D-Bus, for unprivileged applications that require D-Bus to be running.

`home-dbus-configuration` [Data Type]
The configuration record for `home-dbus-service-type`.

`dbus` (default: `dbus`)
: The package providing the `/bin/dbus-daemon` command.

`home-unclutter-service-type` [Variable]
This is the service type for Unclutter, a program that runs on the background of an X11 session and detects when the X pointer hasn't moved for a specified idle timeout, after which it hides the cursor so that you can focus on the text underneath. Its associated value must be a `home-unclutter-configuration` record, as shown below.

A typical configuration, where we manually specify the idle timeout (in seconds), might look like this:

```
(service home-unclutter-service-type
         (home-unclutter-configuration
          (idle-timeout 2)))
```

`home-unclutter-configuration` [Data Type]
The configuration record for `home-unclutter-service-type`.

`unclutter` (default: `unclutter`) (type: file-like)
: Unclutter package to use.

`idle-timeout` (default: `5`) (type: integer)
: A timeout in seconds after which to hide cursor.

`home-xmodmap-service-type` [Variable]
This is the service type for the xmodmap (`https://gitlab.freedesktop.org/xorg/app/xmodmap`) utility to modify keymaps and pointer button mappings under the Xorg display server. Its associated value must be a `home-xmodmap-configuration` record, as shown below.

The `key-map` field takes a list of objects, each of which is either a *statement* (a string) or an *assignment* (a pair of strings). As an example, the snippet below swaps around the *`Caps_Lock`* and the *`Control_L`* keys, by first removing the keysyms (on the right-hand side) from the corresponding modifier maps (on the left-hand side), re-assigning them by swapping each other out, and finally adding back the keysyms to the modifier maps.

```
(service home-xmodmap-service-type
         (home-xmodmap-configuration
          (key-map '(("remove Lock" . "Caps_Lock")
                     ("remove Control" . "Control_L")
```

```
                                ("keysym Control_L" . "Caps_Lock")
                                ("keysym Caps_Lock" . "Control_L")
                                ("add Lock" . "Caps_Lock")
                                ("add Control" . "Control_L")))))
```

`home-xmodmap-configuration` [Data Type]

The configuration record for `home-xmodmap-service-type`. Its available fields are:

`xmodmap` (default: `xmodmap`) (type: file-like)
: The `xmodmap` package to use.

`key-map` (default: `'()`) (type: list)
: The list of expressions to be read by `xmodmap` on service startup.

`home-startx-command-service-type` [Variable]

Add `startx` to the home profile putting it onto `PATH`.

The value for this service is a `<xorg-configuration>` object which is passed to the `xorg-start-command-xinit` procedure producing the `startx` used. Default value is `(xorg-configuration)`.

### 13.3.9 Guix Home Services

The `(gnu home services guix)` module provides services for user-specific Guix configuration.

`home-channels-service-type` [Variable]

This is the service type for managing `$XDG_CONFIG_HOME/guix/channels.scm`, the file that controls the channels received on `guix pull` (see Chapter 6 [Canales], page 69). Its associated value is a list of `channel` records, defined in the `(guix channels)` module.

Generally, it is better to extend this service than to directly configure it, as its default value is the default guix channel(s) defined by `%default-channels`. If you configure this service directly, be sure to include a guix channel. See Section 6.1 [Especificación de canales adicionales], page 69, and Section 6.2 [Uso de un canal de Guix personalizado], page 69, for more details.

A typical extension for adding a channel might look like this:

```
(simple-service 'variant-packages-service
                home-channels-service-type
                (list
                 (channel
                  (name 'variant-packages)
                  (url "https://example.org/variant-packages.git"))))
```

### 13.3.10 Fonts Home Services

The `(gnu home services fontutils)` module provides services for user-specific Fontconfig setup. The Fontconfig (`https://www.freedesktop.org/wiki/Software/fontconfig`) library is used by many applications to access fonts on the system.

`home-fontconfig-service-type` [Variable]

This is the service type for generating configurations for Fontconfig. Its associated value is a list of either strings (or gexps) pointing to fonts locations, or SXML (see Section "SXML" in *GNU Guile Reference Manual*) fragments to be converted into XML and put inside the main `fontconfig` node.

Generally, it is better to extend this service than to directly configure it, as its default value is the default Guix Home's profile font installation path (`~/.guix-home/profile/share/fonts`). If you configure this service directly, be sure to include the above directory.

Here's how you'd extend it to include fonts installed with the Nix package manager, and to prefer your favourite monospace font:

```
(simple-service 'additional-fonts-service
                home-fontconfig-service-type
                (list "~/.nix-profile/share/fonts"
                      '(alias
                        (family "monospace")
                        (prefer
                         (family "Liberation Mono")))))
```

### 13.3.11 Sound Home Services

The `(gnu home services sound)` module provides services related to sound support.

#### PulseAudio RTP Streaming Services

The following services dynamically reconfigure the PulseAudio sound server (`https://pulseaudio.org`): they let you toggle broadcast of audio output over the network using the RTP (real-time transport protocol) and, correspondingly, playback of sound received over RTP. Once `home-pulseaudio-rtp-sink-service-type` is among your home services, you can start broadcasting audio output by running this command:

```
herd start pulseaudio-rtp-sink
```

You can then run a PulseAudio-capable mixer, such as `pavucontrol` or `pulsemixer` (both from the same-named package) to control which audio stream(s) should be sent to the RTP "sink".

By default, audio is broadcasted to a multicast address: any device on the LAN (local area network) receives it and may play it. Using multicast in this way puts a lot of pressure on the network and degrades its performance, so you may instead prefer sending to specifically one device. The first way to do that is by specifying the IP address of the target device when starting the service:

```
herd start pulseaudio-rtp-sink 192.168.1.42
```

The other option is to specify this IP address as the one to use by default in your home environment configuration:

```
(service home-pulseaudio-rtp-sink-service-type
         "192.168.1.42")
```

On the device where you intend to receive and play the RTP stream, you can use `home-pulseaudio-rtp-source-service-type`, like so:

```
(service home-pulseaudio-rtp-source-service-type)
```

This will then let you start the receiving module for PulseAudio:

```
herd start pulseaudio-rtp-source
```

Again, by default it will listen on the multicast address. If, instead, you'd like it to listen for direct incoming connections, you can do that by running:

```
(service home-pulseaudio-rtp-source-service-type
         "0.0.0.0")
```

The reference of these services is given below.

`home-pulseaudio-rtp-sink-service-type` [Variable]
`home-pulseaudio-rtp-source-service-type` [Variable]

This is the type of the service to send, respectively receive, audio streams over RTP (real-time transport protocol).

The value associated with this service is the IP address (a string) where to send, respectively receive, the audio stream. By default, audio is sent/received on multicast address `%pulseaudio-rtp-multicast-address`.

This service defines one Shepherd service: `pulseaudio-rtp-sink`, respectively `pulseaudio-rtp-source`. The service is not started by default, so you have to explicitly start it when you want to turn it on, as in this example:

```
herd start pulseaudio-rtp-sink
```

Stopping the Shepherd service turns off broadcasting.

`%pulseaudio-rtp-multicast-address` [Variable]

This is the multicast address used by default by the two services above.

## PipeWire Home Service

PipeWire (`https://pipewire.org`) provides a low-latency, graph-based audio and video processing service. In addition to its native protocol, it can also be used as a replacement for both JACK and PulseAudio.

While PipeWire provides the media processing and API, it does not, directly, know about devices such as sound cards, nor how you might want to connect applications, hardware, and media processing filters. Instead, PipeWire relies on a *session manager* to specify all these relationships. While you may use any session manager you wish, for most people the WirePlumber (`https://pipewire.pages.freedesktop.org/wireplumber/`) session manager, a reference implementation provided by the PipeWire project itself, suffices, and that is the one `home-pipewire-service-type` uses.

PipeWire can be used as a replacement for PulseAudio by setting `enable-pulseaudio?` to `#t` in `home-pipewire-configuration`, so that existing PulseAudio clients may use it without any further configuration.

In addition, JACK clients may connect to PipeWire by using the `pw-jack` program, which comes with PipeWire. Simply prefix the command with `pw-jack` when you run it, and audio data should go through PipeWire:

```
pw-jack mpv -ao=jack sound-file.wav
```

For more information on PulseAudio emulation, see `https://gitlab.freedesktop.org/pipewire/pipewire/-/wikis/Config-PulseAudio`, for JACK, see `https://gitlab.freedesktop.org/pipewire/pipewire/-/wikis/Config-JACK`.

As PipeWire does not use `dbus` to start its services on demand (as PulseAudio does), `home-pipewire-service-type` uses Shepherd to start services when logged in, provisioning the `pipewire`, `wireplumber`, and, if configured, `pipewire-pulseaudio` services. See Section 13.3.5 [Shepherd Home Service], page 686.

`home-pipewire-service-type` [Variable]

This provides the service definition for `pipewire`, which will run on login. Its value is a `home-pipewire-configuration` object.

To start the service, add it to the `service` field of your `home-environment`, such as:

```
(service home-pipewire-service-type)
```

`home-pipewire-configuration` [Data Type]

Available `home-pipewire-configuration` fields are:

`pipewire` (default: `pipewire`) (type: file-like)
The PipeWire package to use.

`wireplumber` (default: `wireplumber`) (type: file-like)
The WirePlumber package to use.

`enable-pulseaudio?` (default: `#t`) (type: boolean)
When true, enable PipeWire's PulseAudio emulation support, allowing PulseAudio clients to use PipeWire transparently.

### 13.3.12 Mail Home Services

The `(gnu home services mail)` module provides services that help you set up the tools to work with emails in your home environment.

MSMTP (`https://marlam.de/msmtp`) is a SMTP (Simple Mail Transfer Protocol) client. It sends mail to a predefined SMTP server that takes care of proper delivery.

The service reference is given below.

`home-msmtp-service-type` [Variable]

This is the service type for `msmtp`. Its value must be a `home-msmtp-configuration`, as shown below. It provides the `~/.config/msmtp/config` file.

As an example, here is how you would configure `msmtp` for a single account:

```
(service home-msmtp-service-type
         (home-msmtp-configuration
          (accounts
           (list
            (msmtp-account
             (name "alice")
             (configuration
              (msmtp-configuration
               (host "mail.example.org")
               (port 587)
               (user "alice")
               (password-eval "pass Mail/alice")))))))))
```

`home-msmtp-configuration` [Data Type]

Available `home-msmtp-configuration` fields are:

`defaults` (type: msmtp-configuration)
The configuration that will be set as default for all accounts.

`accounts` (default: `'()`) (type: list-of-msmtp-accounts)
A list of `msmtp-account` records which contain information about all your accounts.

`default-account` (type: maybe-string)
Set the default account.

`extra-content` (default: `""`) (type: string)
Extra content appended as-is to the configuration file. Run `man msmtp` for more information about the configuration file format.

`msmtp-account` [Data Type]

Available `msmtp-account` fields are:

`name` (type: string)
The unique name of the account.

`configuration` (type: msmtp-configuration)
The configuration for this given account.

`msmtp-configuration` [Data Type]

Available `msmtp-configuration` fields are:

`auth?` (type: maybe-boolean)
Enable or disable authentication.

`tls?` (type: maybe-boolean)
Enable or disable TLS (also known as SSL) for secured connections.

`tls-starttls?` (type: maybe-boolean)
Choose the TLS variant: start TLS from within the session ('on', default), or tunnel the session through TLS ('off').

`tls-trust-file` (type: maybe-string)
Activate server certificate verification using a list of trusted Certification Authorities (CAs).

`log-file` (type: maybe-string)
Enable logging to the specified file. An empty argument disables logging. The file name '-' directs the log information to standard output.

`host` (type: maybe-string)
The SMTP server to send the mail to.

`port` (type: maybe-integer)
The port that the SMTP server listens on. The default is 25 (`"smtp"`), unless TLS without STARTTLS is used, in which case it is 465 (`"smtps"`).

`user` (type: maybe-string)
Set the user name for authentication.

`from` (type: maybe-string)
: Set the envelope-from address.

`password-eval` (type: maybe-string)
: Set the password for authentication to the output (stdout) of the command cmd.

`extra-content` (default: `""`) (type: string)
: Extra content appended as-is to the configuration block. Run `man msmtp` for more information about the configuration file format.

### 13.3.13 Messaging Home Services

The ZNC bouncer (`https://znc.in`) can be run as a daemon to manage your IRC presence. With the `(gnu home services messaging)` service, you can configure ZNC to run upon login.

You will have to provide a `~/.znc/configs/znc.conf` separately.

Here is an example of a service and its configuration that you could add to the `services` field of your `home-environment`:

```
(service home-znc-service-type)
```

`home-znc-service-type` [Variable]
: This is the type of the ZNC home service, whose value is a `home-znc-configuration` object.

`home-znc-configuration` [Data Type]
: Available `home-znc-configuration` fields are:

  `znc` (default: `znc`) (type: file-like)
  : The ZNC package to use.

  `extra-options` (predeterminadas: `'()`)
  : Extra options will be passed to `znc`, please run `man znc` for more information.

### 13.3.14 Media Home Services

The Kodi media center (`https://kodi.tv`) can be run as a daemon on a media server. With the `(gnu home services kodi)` service, you can configure Kodi to run upon login.

Here is an example of a service and its configuration that you could add to the `services` field of your `home-environment`:

```
(service home-kodi-service-type
  (home-kodi-configuration
   (extra-options '("--settings="<settings-file>"))))
```

`home-kodi-service-type` [Variable]
: This is the type of the Kodi home service, whose value is a `home-kodi-configuration` object.

`home-kodi-configuration` [Data Type]

Available `home-kodi-configuration` fields are:

`kodi` (default: `kodi`) (type: file-like)
The Kodi package to use.

`extra-options` (predeterminadas: `'()`)
Extra options will be passed to `kodi`, please run `man kodi` for more information.

### 13.3.15 Networking Home Services

This section lists services somewhat networking-related that you may use with Guix Home.

The `(gnu home services syncthing)` module provides a service to set up the https://syncthing.net (`Syncthing`) continuous file backup service.

`home-syncthing-service-type` [Variable]

This is the service type for the `syncthing` daemon; it is the Home counterpart of the `syncthing-service-type` system service (see Section 11.10.5 [Servicios de red], page 314). The value for this service type is a `syncthing-configuration`.

Here is how you would set it up with the default configuration:

```
(service home-syncthing-service-type)
```

For a custom configuration, wrap you `syncthing-configuration` in `for-home`, as in this example:

```
(service home-syncthing-service-type
         (for-home
          (syncthing-configuration (logflags 5))))
```

For details about `syncthing-configuration`, check out the documentation of the system service (see Section 11.10.5 [Servicios de red], page 314).

### 13.3.16 Miscellaneous Home Services

This section lists Home services that lack a better place.

#### Servicio de diccionario

The `(gnu home services dict)` module provides the following service:

`home-dicod-service-type` [Variable]

Tipo de servicio que ejecuta el daemon `dicod`, una implementación del servidor DICT (see Section “Dicod” in *GNU Dico Manual*).

Puede añadir `open localhost` en su archivo `~/.dico` para hacer que `localhost` sea el servidor predeterminado de su cliente `dico` (see Section “Initialization File” in *GNU Dico Manual*).

This service is a direct mapping of the `dicod-service-type` system service (see Section 11.10.37 [Servicios misceláneos], page 603). You can use it like this:

```
(service home-dicod-service-type)
```

You may specify a custom configuration by providing a `dicod-configuration` record, exactly like for `dicod-service-type`, but wrapping it in `for-home`:

```
(service home-dicod-service-type
         (for-home
          (dicod-configuration ...)))
```

## 13.4 Invoking `guix home`

Once you have written a home environment declaration (see Section 13.1 [Declaring the Home Environment], page 671, it can be *instantiated* using the `guix home` command. The synopsis is:

```
guix home options... action file
```

*file* must be the name of a file containing a `home-environment` declaration. *action* specifies how the home environment is instantiated, but there are few auxiliary actions which don't instantiate it. Currently the following values are supported:

`search` Display available home service type definitions that match the given regular expressions, sorted by relevance:

```
$ guix home search shell
name: home-shell-profile
location: gnu/home/services/shells.scm:100:2
extends: home-files
description: Create `~/.profile', which is used for environment initializati
+ This service type can be extended with a list of file-like objects.
relevance: 6

name: home-fish
location: gnu/home/services/shells.scm:640:2
extends: home-files home-profile
description: Install and configure Fish, the friendly interactive shell.
relevance: 3

name: home-zsh
location: gnu/home/services/shells.scm:290:2
extends: home-files home-profile
description: Install and configure Zsh.
relevance: 1

name: home-bash
location: gnu/home/services/shells.scm:508:2
extends: home-files home-profile
description: Install and configure GNU Bash.
relevance: 1

...
```

As for `guix search`, the result is written in `recutils` format, which makes it easy to filter the output (see *GNU recutils manual*).

`container`

Spawn a shell in an isolated environment—a *container*—containing your home as specified by *file*.

For example, this is how you would start an interactive shell in a container with your home:

```
guix home container config.scm
```

This is a throw-away container where you can lightheartedly fiddle with files; any changes made within the container, any process started—all this disappears as soon as you exit that shell.

As with `guix shell`, several options control that container:

`--network`
`-N` Enable networking within the container (it is disabled by default).

`--expose=`*fuente*`[=`*destino*`]`
`--share=`*fuente*`[=`*destino*`]`
As with `guix shell`, make directory *source* of the host system available as *target* inside the container—read-only if you pass `--expose`, and writable if you pass `--share` (see Section 7.1 [Invoking guix shell], page 79).

Additionally, you can run a command in that container, instead of spawning an interactive shell. For instance, here is how you would check which Shepherd services are started in a throw-away home container:

```
guix home container config.scm -- herd status
```

The command to run in the container must come after `--` (double hyphen).

`edit` Edit or view the definition of the given Home service types.

For example, the command below opens your editor, as specified by the `EDITOR` environment variable, on the definition of the `home-mcron` service type:

```
guix home edit home-mcron
```

`reconfigure`

Build the home environment described in *file*, and switch to it. Switching means that the activation script will be evaluated and (in basic scenario) symlinks to configuration files generated from `home-environment` declaration will be created in `~`. If the file with the same path already exists in home folder it will be moved to `~/`*timestamp*`-guix-home-legacy-configs-backup`, where *timestamp* is a current UNIX epoch time.

> **Nota:** It is highly recommended to run `guix pull` once before you run `guix home reconfigure` for the first time (see Section 5.7 [Invocación de guix pull], page 57).

This effects all the configuration specified in *file*. The command starts Shepherd services specified in *file* that are not currently running; if a service is currently running, this command will arrange for it to be upgraded the next time it is stopped (e.g. by `herd stop` *service* or `herd restart` *service*).

This command creates a new generation whose number is one greater than the current generation (as reported by `guix home list-generations`). If that

generation already exists, it will be overwritten. This behavior mirrors that of `guix package` (see Section 5.2 [Invocación de guix package], page 36).

Upon completion, the new home is deployed under `~/.guix-home`. This directory contains *provenance meta-data*: the list of channels in use (see Chapter 6 [Canales], page 69) and *file* itself, when available. You can view the provenance information by running:

```
guix home describe
```

This information is useful should you later want to inspect how this particular generation was built. In fact, assuming *file* is self-contained, you can later rebuild generation *n* of your home environment with:

```
guix time-machine \
  -C /var/guix/profiles/per-user/USER/guix-home-n-link/channels.scm -- \
  home reconfigure \
  /var/guix/profiles/per-user/USER/guix-home-n-link/configuration.scm
```

You can think of it as some sort of built-in version control! Your home is not just a binary artifact: *it carries its own source*.

> **Nota:** If you're using Guix System, [guix-home-service-type], page 594, on how to embed your home configuration in your system configuration such that `guix system reconfigure` deploys both your system and your home.

`switch-generation`

Switch to an existing home generation. This action atomically switches the home profile to the specified home generation.

The target generation can be specified explicitly by its generation number. For example, the following invocation would switch to home generation 7:

```
guix home switch-generation 7
```

La generación deseada puede especificarse también de forma relativa a la generación actual con la forma `+N` o `-N`, donde `+3` significa "3 generaciones después de la generación actual", y `-1` significa "1 generación antes de la generación actual". Cuando se especifica un valor negativo como `-1` debe ir precedido de `--` para evitar que se analice como una opción. Por ejemplo:

```
guix home switch-generation -- -1
```

Esta acción fallará si la generación especificada no existe.

`roll-back`

Switch to the preceding home generation. This is the inverse of `reconfigure`, and it is exactly the same as invoking `switch-generation` with an argument of `-1`.

`delete-generations`

Delete home generations, making them candidates for garbage collection (see Section 5.6 [Invocación de guix gc], page 53, for information on how to run the "garbage collector").

This works in the same way as '`guix package --delete-generations`' (see Section 5.2 [Invocación de guix package], page 36). With no arguments, all home generations but the current one are deleted:

```
guix home delete-generations
```

You can also select the generations you want to delete. The example below deletes all the home generations that are more than two months old:

```
guix home delete-generations 2m
```

`build` Build the derivation of the home environment, which includes all the configuration files and programs needed. This action does not actually install anything.

`describe` Describe the current home generation: its file name, as well as provenance information when available.

To show installed packages in the current home generation's profile, the `--list-installed` flag is provided, with the same syntax that is used in `guix package --list-installed` (see Section 5.2 [Invocación de guix package], page 36). For instance, the following command shows a table of all the packages with "emacs" in their name that are installed in the current home generation's profile:

```
guix home describe --list-installed=emacs
```

`list-generations`
List a summary of each generation of the home environment available on disk, in a human-readable way. This is similar to the `--list-generations` option of `guix package` (see Section 5.2 [Invocación de guix package], page 36).

De manera opcional, se puede especificar un patrón, con la misma sintaxis que la usada en `guix package --list-generations`, para restringir la lista de generaciones mostradas. Por ejemplo, la siguiente orden muestra generaciones que tienen hasta 10 días de antigüedad:

```
guix home list-generations 10d
```

The `--list-installed` flag may also be specified, with the same syntax that is used in `guix home describe`. This may be helpful if trying to determine when a package was added to the home profile.

`import` Generate a *home environment* from the packages in the default profile and configuration files found in the user's home directory. The configuration files will be copied to the specified directory, and a `home-configuration.scm` will be populated with the home environment. Note that not every home service that exists is supported (see Section 13.3 [Home Services], page 674).

```
$ guix home import ~/guix-config
guix home: '/home/alice/guix-config' populated with all the Home configurati
```

And there's more! `guix home` also provides the following sub-commands to visualize how the services of your home environment relate to one another:

`extension-graph`
Emit to standard output the *service extension graph* of the home environment defined in *file* (see Section 11.19.1 [Composición de servicios], page 649, for more information on service extensions). By default the output is in Dot/Graphviz

format, but you can choose a different format with `--graph-backend`, as with `guix graph` (see Section 9.10 [Invocación de guix graph], page 221):

La orden:

```
guix home extension-graph file | xdot -
```

muestra las relaciones de extensión entre los servicios.

`shepherd-graph`
: Emit to standard output the *dependency graph* of shepherd services of the home environment defined in *file*. See Section 11.19.4 [Servicios de Shepherd], page 657, for more information and for an example graph.

  Again, the default output format is Dot/Graphviz, but you can pass `--graph-backend` to select a different one.

*opciones* puede contener cualquiera de las opciones de construcción comunes (see Section 9.1.1 [Opciones comunes de construcción], page 181). Además, *opciones* puede contener una de las siguientes:

`--expression=`*expr*
`-e` *expr*
: Consider the home-environment *expr* evaluates to. This is an alternative to specifying a file which evaluates to a home environment.

`--allow-downgrades`
: Instruct `guix home reconfigure` to allow system downgrades.

  Just like `guix system`, `guix home reconfigure`, by default, prevents you from downgrading your home to older or unrelated revisions compared to the channel revisions that were used to deploy it—those shown by `guix home describe`. Using `--allow-downgrades` allows you to bypass that check, at the risk of downgrading your home—be careful!

# 14 Documentación

In most cases packages installed with Guix come with documentation. There are two main documentation formats: “Info”, a browsable hypertext format used for GNU software, and “manual pages” (or “man pages”), the linear documentation format traditionally found on Unix. Info manuals are accessed with the `info` command or with Emacs, and man pages are accessed using `man`.

Puede buscar documentación de software instalado en su sistema por palabras clave. Por ejemplo, la siguiente orden busca información sobre “TLS” en manuales Info:

```
$ info -k TLS
"(emacs)Network Security" -- STARTTLS
"(emacs)Network Security" -- TLS
"(gnutls)Core TLS API" -- gnutls_certificate_set_verify_flags
"(gnutls)Core TLS API" -- gnutls_certificate_set_verify_function
...
```

The command below searches for the same keyword in man pages[1]:

```
$ man -k TLS
SSL (7)              - OpenSSL SSL/TLS library
certtool (1)         - GnuTLS certificate tool
...
```

Estas búsquedas son completamente locales en su máquina de modo que tiene la garantía de que la documentación que encuentre corresponde con lo que está realmente instalado, puede acceder a ella sin conexión a la red, y se respeta su privacidad.

Una vez tenga estos resultados, puede ver la documentación relevante mediante la ejecución de, digamos:

```
$ info "(gnutls)Core TLS API"
```

o:

```
$ man certtool
```

Los manuales Info contienen secciones e índices, así como enlaces como aquellos encontrados en páginas Web. El lector `info` (see *Stand-alone GNU Info*) y su contraparte en Emacs (see Section “Misc Help” in *The GNU Emacs Manual*) proporcionan combinaciones de teclas intuitivas para la navegación en los manuales. See Section “Getting Started” in *Info: An Introduction*, para una introducción a la navegación en Info.

[1] The database searched by `man -k` is only created in profiles that contain the `man-db` package.

# 15 Platforms

The packages and systems built by Guix are intended, like most computer programs, to run on a CPU with a specific instruction set, and under a specific operating system. Those programs are often also targeting a specific kernel and system library. Those constraints are captured by Guix in `platform` records.

## 15.1 `platform` Reference

The `platform` data type describes a *platform*: an ISA (instruction set architecture), combined with an operating system and possibly additional system-wide settings such as the ABI (application binary interface).

`platform` [Data Type]

This is the data type representing a platform.

`target` This field specifies the platform's GNU triplet as a string (see Section "Specifying Target Triplets" in *Autoconf*).

`system` This string is the system type as it is known to Guix and passed, for instance, to the `--system` option of most commands.

It usually has the form `"cpu-kernel"`, where *cpu* is the target CPU and *kernel* the target operating system kernel.

It can be for instance `"aarch64-linux"` or `"armhf-linux"`. You will encounter system types when you perform native builds (see Section 10.2 [Native Builds], page 240).

`linux-architecture` (default: `#false`)
This optional string field is only relevant if the kernel is Linux. In that case, it corresponds to the ARCH variable used when building Linux, `"mips"` for instance.

`rust-target` (default: `#false`)
This optional string field is used to determine which rust target is best supported by this platform. For example, the base level system targeted by `armhf-linux` system is closest to `armv7-unknown-linux-gnueabihf`.

`glibc-dynamic-linker`
This field is the name of the GNU C Library dynamic linker for the corresponding system, as a string. It can be `"/lib/ld-linux-armhf.so.3"`.

## 15.2 Supported Platforms

The `(guix platforms ...)` modules export the following variables, each of which is bound to a `platform` record.

`armv7-linux` [Variable]

Platform targeting ARM v7 CPU running GNU/Linux.

`aarch64-linux` [Variable]

Platform targeting ARM v8 CPU running GNU/Linux.

`mips64-linux` [Variable]
Platform targeting MIPS little-endian 64-bit CPU running GNU/Linux.

`powerpc-linux` [Variable]
Platform targeting PowerPC big-endian 32-bit CPU running GNU/Linux.

`powerpc64le-linux` [Variable]
Platform targeting PowerPC little-endian 64-bit CPU running GNU/Linux.

`riscv64-linux` [Variable]
Platform targeting RISC-V 64-bit CPU running GNU/Linux.

`i686-linux` [Variable]
Platform targeting x86 CPU running GNU/Linux.

`x86_64-linux` [Variable]
Platform targeting x86 64-bit CPU running GNU/Linux.

`x86_64-linux-x32` [Variable]
Platform targeting x86 64-bit CPU running GNU/Linux with the run-time using the X32 ABI.

`i686-mingw` [Variable]
Platform targeting x86 CPU running Windows, with run-time support from MinGW.

`x86_64-mingw` [Variable]
Platform targeting x86 64-bit CPU running Windows, with run-time support from MinGW.

`i586-gnu` [Variable]
Platform targeting x86 CPU running GNU/Hurd (also referred to as "GNU").

`avr` [Variable]
Platform targeting AVR CPUs without an operating system, with run-time support from AVR Libc.

`or1k-elf` [Variable]
Platform targeting OpenRISC 1000 CPU without an operating system and without a C standard library.

`xtensa-ath9k-elf` [Variable]
Platform targeting Xtensa CPU used in the Qualcomm Atheros AR7010 and AR9271 USB 802.11n NICs (Network Interface Controllers).

# 16 Creating System Images

When it comes to installing Guix System for the first time on a new machine, you can basically proceed in three different ways. The first one is to use an existing operating system on the machine to run the `guix system init` command (see Section 11.16 [Invocación de guix system], page 634). The second one, is to produce an installation image (see Section 3.9 [Construcción de la imagen de instalación], page 31). This is a bootable system which role is to eventually run `guix system init`. Finally, the third option would be to produce an image that is a direct instantiation of the system you wish to run. That image can then be copied on a bootable device such as an USB drive or a memory card. The target machine would then directly boot from it, without any kind of installation procedure.

The `guix system image` command is able to turn an operating system definition into a bootable image. This command supports different image types, such as `mbr-hybrid-raw`, `iso9660` and `docker`. Any modern `x86_64` machine will probably be able to boot from an `iso9660` image. However, there are a few machines out there that require specific image types. Those machines, in general using `ARM` processors, may expect specific partitions at specific offsets.

This chapter explains how to define customized system images and how to turn them into actual bootable images.

## 16.1 `image` Reference

The `image` record, described right after, allows you to define a customized bootable system image.

`image` [Data Type]

This is the data type representing a system image.

`name` (default: `#false`)
: The image name as a symbol, `'my-iso9660` for instance. The name is optional and it defaults to `#false`.

`format`
: The image format as a symbol. The following formats are supported:

  - `disk-image`, a raw disk image composed of one or multiple partitions.
  - `compressed-qcow2`, a compressed qcow2 image composed of one or multiple partitions.
  - `docker`, a Docker image.
  - `iso9660`, an ISO-9660 image.
  - `tarball`, a tar.gz image archive.
  - `wsl2`, a WSL2 image.

`platform` (default: `#false`)
: The `platform` record the image is targeting (see Chapter 15 [Platforms], page 708), `aarch64-linux` for instance. By default, this field is set to `#false` and the image will target the host platform.

`size` (default: `'guess`)
: The image size in bytes or `'guess`. The `'guess` symbol, which is the default, means that the image size will be inferred based on the image content.

`operating-system`
: The image's `operating-system` record that is instantiated.

`partition-table-type` (default: `'mbr`)
: The image partition table type as a symbol. Possible values are `'mbr` and `'gpt`. It default to `'mbr`.

`partitions` (default: `'()`)
: The image partitions as a list of `partition` records (see Section 16.1.1 [partition Reference], page 711).

`compression?` (default: `#true`)
: Whether the image content should be compressed, as a boolean. It defaults to `#true` and only applies to `'iso9660` image formats.

`volatile-root?` (default: `#true`)
: Whether the image root partition should be made volatile, as a boolean. This is achieved by using a RAM backed file system (overlayfs) that is mounted on top of the root partition by the initrd. It defaults to `#true`. When set to `#false`, the image root partition is mounted as read-write partition by the initrd.

`shared-store?` (default: `#false`)
: Whether the image's store should be shared with the host system, as a boolean. This can be useful when creating images dedicated to virtual machines. When set to `#false`, which is the default, the image's `operating-system` closure is copied to the image. Otherwise, when set to `#true`, it is assumed that the host store will be made available at boot, using a `9p` mount for instance.

`shared-network?` (default: `#false`)
: Whether to use the host network interfaces within the image, as a boolean. This is only used for the `'docker` image format. It defaults to `#false`.

`substitutable?` (default: `#true`)
: Whether the image derivation should be substitutable, as a boolean. It defaults to `true`.

### 16.1.1 `partition` Reference

In `image` record may contain some partitions.

`partition` [Data Type]
: This is the data type representing an image partition.

`size` (default: `'guess`)
: The partition size in bytes or `'guess`. The `'guess` symbol, which is the default, means that the partition size will be inferred based on the partition content.

`offset` (default: `0`)
: The partition's start offset in bytes, relative to the image start or the previous partition end. It defaults to `0` which means that there is no offset applied.

`file-system` (default: `"ext4"`)
: The partition file system as a string, defaulting to `"ext4"`. The supported values are `"vfat"`, `"fat16"`, `"fat32"` and `"ext4"`. `"vfat"`, `"fat16"` and `"fat32"` partitions without the `'esp` flag are by default LBA compatible.

`file-system-options` (default: `'()`)
: The partition file system creation options that should be passed to the partition creation tool, as a list of strings. This is only supported when creating `"ext4"` partitions.

  See the `"extended-options"` man page section of the `"mke2fs"` tool for a more complete reference.

`label`
: The partition label as a mandatory string, `"my-root"` for instance.

`uuid` (default: `#false`)
: The partition UUID as an `uuid` record (see Section 11.4 [Sistemas de archivos], page 257). By default it is `#false`, which means that the partition creation tool will attribute a random UUID to the partition.

`flags` (predeterminadas: `'()`)
: The partition flags as a list of symbols. Possible values are `'boot` and `'esp`. The `'boot` flags should be set if you want to boot from this partition. Exactly one partition should have this flag set, usually the root one. The `'esp` flag identifies a UEFI System Partition.

`initializer` (default: `#false`)
: The partition initializer procedure as a gexp. This procedure is called to populate a partition. If no initializer is passed, the `initialize-root-partition` procedure from the `(gnu build image)` module is used.

## 16.2 Instantiate an Image

Let's say you would like to create an MBR image with three distinct partitions:

- The ESP (EFI System Partition), a partition of 40 MiB at offset 1024 KiB with a vfat file system.
- an ext4 partition of 50 MiB data file, and labeled "data".
- an ext4 bootable partition containing the `%simple-os` operating-system.

You would then write the following image definition in a `my-image.scm` file for instance.

```
(use-modules (gnu)
             (gnu image)
             (gnu tests)
             (gnu system image)
             (guix gexp))
```

```
(define MiB (expt 2 20))

(image
 (format 'disk-image)
 (operating-system %simple-os)
 (partitions
  (list
   (partition
    (size (* 40 MiB))
    (offset (* 1024 1024))
    (label "GNU-ESP")
    (file-system "vfat")
    (flags '(esp))
    (initializer (gexp initialize-efi-partition)))
   (partition
    (size (* 50 MiB))
    (label "DATA")
    (file-system "ext4")
    (initializer #~(lambda* (root . rest)
                     (mkdir root)
                     (call-with-output-file
                         (string-append root "/data")
                       (lambda (port)
                         (format port "my-data"))))))
   (partition
    (size 'guess)
    (label root-label)
    (file-system "ext4")
    (flags '(boot))
    (initializer (gexp initialize-root-partition))))))
```

Note that the first and third partitions use generic initializers procedures, initialize-efi-partition and initialize-root-partition respectively. The initialize-efi-partition installs a GRUB EFI loader that is loading the GRUB bootloader located in the root partition. The initialize-root-partition instantiates a complete system as defined by the `%simple-os` operating-system.

You can now run:

```
guix system image my-image.scm
```

to instantiate the `image` definition. That produces a disk image which has the expected structure:

```
$ parted $(guix system image my-image.scm) print
...
Model:  (file)
Disk /gnu/store/yhylv1bp5b2ypb97pd3bbhz6jk5nbhxw-disk-image: 1714MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
```

```
Number  Start   End     Size    Type     File system  Flags
 1      1049kB  43.0MB  41.9MB  primary  fat16        esp
 2      43.0MB  95.4MB  52.4MB  primary  ext4
 3      95.4MB  1714MB  1619MB  primary  ext4         boot
```

The size of the `boot` partition has been inferred to `1619MB` so that it is large enough to host the `%simple-os` operating-system.

You can also use existing `image` record definitions and inherit from them to simplify the `image` definition. The `(gnu system image)` module provides the following `image` definition variables.

`mbr-disk-image` [Variable]

An MBR disk-image composed of a single ROOT partition. The ROOT partition starts at a 1 MiB offset so that the bootloader can install itself in the post-MBR gap.

`mbr-hybrid-disk-image` [Variable]

An MBR disk-image composed of two partitions: a 64 bits ESP partition and a ROOT boot partition. The ESP partition starts at a 1 MiB offset so that a BIOS compatible bootloader can install itself in the post-MBR gap. The image can be used by `x86_64` and `i686` machines supporting only legacy BIOS booting. The ESP partition ensures that it can also be used by newer machines relying on UEFI booting, hence the *hybrid* denomination.

`efi-disk-image` [Variable]

A GPT disk-image composed of two partitions: a 64 bits ESP partition and a ROOT boot partition. This image can be used on most `x86_64` and `i686` machines, supporting BIOS or UEFI booting.

`efi32-disk-image` [Variable]

Same as `efi-disk-image` but with a 32 bits EFI partition.

`iso9660-image` [Variable]

An ISO-9660 image composed of a single bootable partition. This image can also be used on most `x86_64` and `i686` machines.

`docker-image` [Variable]

A Docker image that can be used to spawn a Docker container.

Using the `efi-disk-image` we can simplify our previous `image` declaration this way:

```
(use-modules (gnu)
             (gnu image)
             (gnu tests)
             (gnu system image)
             (guix gexp)
             (ice-9 match))

(define MiB (expt 2 20))
```

```
(define data
  (partition
   (size (* 50 MiB))
   (label "DATA")
   (file-system "ext4")
   (initializer #~(lambda* (root . rest)
                    (mkdir root)
                    (call-with-output-file
                        (string-append root "/data")
                      (lambda (port)
                        (format port "my-data")))))))

(image
 (inherit efi-disk-image)
 (operating-system %simple-os)
 (partitions
  (match (image-partitions efi-disk-image)
    ((esp root)
     (list esp data root)))))
```

This will give the exact same `image` instantiation but the `image` declaration is simpler.

## 16.3 image-type Reference

The `guix system image` command can, as we saw above, take a file containing an `image` declaration as argument and produce an actual disk image from it. The same command can also handle a file containing an `operating-system` declaration as argument. In that case, how is the `operating-system` turned into an image?

That's where the `image-type` record intervenes. This record defines how to transform an `operating-system` record into an `image` record.

`image-type` [Data Type]

This is the data type representing an image-type.

`name` The image-type name as a mandatory symbol, `'efi32-raw` for instance.

`constructor`
The image-type constructor, as a mandatory procedure that takes an `operating-system` record as argument and returns an `image` record.

There are several `image-type` records provided by the `(gnu system image)` and the `(gnu system images ...)` modules.

`mbr-raw-image-type` [Variable]

Build an image based on the `mbr-disk-image` image.

`mbr-hybrid-raw-image-type` [Variable]

Build an image based on the `mbr-hybrid-disk-image` image.

`efi-raw-image-type` [Variable]

Build an image based on the `efi-disk-image` image.

`efi32-raw-image-type` [Variable]
Build an image based on the `efi32-disk-image` image.

`qcow2-image-type` [Variable]
Build an image based on the `mbr-disk-image` image but with the `compressed-qcow2` image format.

`iso-image-type` [Variable]
Build a compressed image based on the `iso9660-image` image.

`uncompressed-iso-image-type` [Variable]
Build an image based on the `iso9660-image` image but with the `compression?` field set to `#false`.

`docker-image-type` [Variable]
Build an image based on the `docker-image` image.

`raw-with-offset-image-type` [Variable]
Build an MBR image with a single partition starting at a `1024KiB` offset. This is useful to leave some room to install a bootloader in the post-MBR gap.

`pinebook-pro-image-type` [Variable]
Build an image that is targeting the Pinebook Pro machine. The MBR image contains a single partition starting at a `9MiB` offset. The `u-boot-pinebook-pro-rk3399-bootloader` bootloader will be installed in this gap.

`rock64-image-type` [Variable]
Build an image that is targeting the Rock64 machine. The MBR image contains a single partition starting at a `16MiB` offset. The `u-boot-rock64-rk3328-bootloader` bootloader will be installed in this gap.

`novena-image-type` [Variable]
Build an image that is targeting the Novena machine. It has the same characteristics as `raw-with-offset-image-type`.

`pine64-image-type` [Variable]
Build an image that is targeting the Pine64 machine. It has the same characteristics as `raw-with-offset-image-type`.

`hurd-image-type` [Variable]
Build an image that is targeting a `i386` machine running the Hurd kernel. The MBR image contains a single ext2 partitions with specific `file-system-options` flags.

`hurd-qcow2-image-type` [Variable]
Build an image similar to the one built by the `hurd-image-type` but with the `format` set to `'compressed-qcow2`.

`wsl2-image-type` [Variable]
Build an image for the WSL2 (Windows Subsystem for Linux 2). It can be imported by running:

```
wsl --import Guix ./guix ./wsl2-image.tar.gz
wsl -d Guix
```

So, if we get back to the `guix system image` command taking an `operating-system` declaration as argument. By default, the `mbr-raw-image-type` is used to turn the provided `operating-system` into an actual bootable image.

To use a different `image-type`, the `--image-type` option can be used. The `--list-image-types` option will list all the supported image types. It turns out to be a textual listing of all the `image-types` variables described just above (see Section 11.16 [Invocación de guix system], page 634).

## 16.4 Image Modules

Let's take the example of the Pine64, an ARM based machine. To be able to produce an image targeting this board, we need the following elements:

- An `operating-system` record containing at least an appropriate kernel (`linux-libre-arm64-generic`) and bootloader `u-boot-pine64-lts-bootloader`) for the Pine64.
- Possibly, an `image-type` record providing a way to turn an `operating-system` record to an `image` record suitable for the Pine64.
- An actual `image` that can be instantiated with the `guix system image` command.

The `(gnu system images pine64)` module provides all those elements: `pine64-barebones-os`, `pine64-image-type` and `pine64-barebones-raw-image` respectively.

The module returns the `pine64-barebones-raw-image` in order for users to be able to run:

```
guix system image gnu/system/images/pine64.scm
```

Now, thanks to the `pine64-image-type` record declaring the `'pine64-raw image-type`, one could also prepare a `my-pine.scm` file with the following content:

```
(use-modules (gnu system images pine64))
(operating-system
  (inherit pine64-barebones-os)
  (timezone "Europe/Athens"))
```

to customize the `pine64-barebones-os`, and run:

```
$ guix system image --image-type=pine64-raw my-pine.scm
```

Note that there are other modules in the `gnu/system/images` directory targeting `Novena`, `Pine64`, `PinebookPro` and `Rock64` machines.

# 17 Instalación de archivos de depuración

Los programas binarios, como los producidos por los compiladores GCC por ejemplo, se escriben típicamente en el formato ELF, con una sección que contiene *información de depuración*. La información de depuración es lo que permite que el depurador, GDB, asocie código binario a código fuente; es necesaria para depurar un programa compilado en condiciones adecuadas.

Este capítulo explica cómo usar la información de depuración separada cuando los paquetes la proporcionan, y cómo reconstruir paquetes con información de depuración cuando esta falte.

## 17.1 Información separada para depuración

El problema con la información de depuración es que ocupa un espacio considerable en el disco. Por ejemplo, la información de depuración de la biblioteca C de GNU ocupa más de 60 MiB. Por tanto, como usuaria, mantener toda la información de depuración de todos los programas instalados no es habitualmente una opción. No obstante, el ahorro de espacio no debe ser impedir la depuración—especialmente en el sistema GNU, que debería facilitar a sus usuarias ejercitar su libertad de computación (see Section 1.2 [Distribución GNU], page 2).

Afortunadamente, las utilidades binarias GNU (Binutils) y GDB proporcionan un mecanismo que permite a las usuarias obtener lo mejor de ambos mundos: la información de depuración puede extraerse de los binarios y almacenarse en archivos separados. GDB es capaz entonces de cargar la información de depuración desde esos archivos, cuando estén disponibles (see Section "Separate Debug Files" in *Debugging with GDB*).

La distribución GNU toma ventaja de este hecho almacenando la información de depuración en el subdirectorio `lib/debug` de una salida separada del paquete llamada `debug` (see Section 5.4 [Paquetes con múltiples salidas], page 51). Las usuarias pueden elegir si instalan la salida `debug` de un paquete cuando la necesitan. Por ejemplo, la siguiente orden instala la información de depuración para la biblioteca C de GNU y para GNU Guile.

```
guix install glibc:debug guile:debug
```

Se debe decir entonces a GDB que busque los archivos de depuración en el perfil de la usuaria, proporcionando un valor a la variable `debug-file-directory` (considere hacerlo en el archivo `~/.gdbinit`, see Section "Startup" in *Debugging with GDB*):

```
(gdb) set debug-file-directory ~/.guix-profile/lib/debug
```

A partir de ese momento GDB obtendrá la información de depuración de los archivos `.debug` dentro de la carpeta `~/.guix-profile/lib/debug`.

Below is an alternative GDB script which is useful when working with other profiles. It takes advantage of the optional Guile integration in GDB. This snippet is included by default on Guix System in the `~/.gdbinit` file.

```
guile
(use-modules (gdb))
(execute (string-append "set debug-file-directory "
                        (or (getenv "GDB_DEBUG_FILE_DIRECTORY")
                            "~/.guix-profile/lib/debug")))
```

```
end
```

Además, probablemente desee que GDB sea capaz de mostrar el código fuente que está depurando. Para hacerlo, tiene que desempaquetar el código fuente del paquete de su interés (obtenido con `guix build --source`, see Section 9.1 [Invocación de guix build], page 181) e indicar a GDB cual es el directorio de fuentes mediante el uso de la orden `directory` (see Section "Source Path" in *Debugging with GDB*).

El mecanismo de la salida `debug` en Guix se implementa por el sistema de construcción `gnu-build-system` (see Section 8.5 [Sistemas de construcción], page 123). Ahora mismo necesita una activación explícita—la información de depuración está disponible únicamente para paquetes con definiciones que declaren explícitamente una salida `debug`. Para comprobar si un paquete tiene una salida `debug`, use `guix package --list-available` (see Section 5.2 [Invocación de guix package], page 36).

Siga leyendo para ver cómo tratar con paquetes que carezcan de la salida `debug`.

## 17.2 Reconstrucción de la información para depuración

Como vimos anteriormente, algunos paquetes, pero no todos, proporcionan información de depuración en una salida llamada `debug`. ¿Qué puede hacer cuando dicha información de depuración no está disponible? La opción `--with-debug-info` proporciona una solución para ello: le permite reconstruir uno o más paquetes para los que la información de depuración no esté disponible—y únicamente esos—e injertarlos en la aplicación que esté depurando. Por lo tanto, aunque no es tan rápido como instalar una salida `debug`, no tiene un coste elevado.

Vamos a ilustrarlo con un ejemplo. Supongamos que está sufriendo un error en Inkscape y desearía que está pasando en GLib, una biblioteca que se encuentra enraizada profundamente en su grafo de dependencias. Además GLib no tiene una salida `debug` y la pila de llamadas que GDB muestra es completamente deprimente:

```
(gdb) bt
#0  0x00007ffff5f92190 in g_getenv ()
   from /gnu/store/...-glib-2.62.6/lib/libglib-2.0.so.0
#1  0x00007ffff608a7d6 in gobject_init_ctor ()
   from /gnu/store/...-glib-2.62.6/lib/libgobject-2.0.so.0
#2  0x00007ffff7fe275a in call_init (l=<optimized out>, argc=argc@entry=1, argv=argv@e
    env=env@entry=0x7fffffffcfe8) at dl-init.c:72
#3  0x00007ffff7fe2866 in call_init (env=0x7fffffffcfe8, argv=0x7fffffffcfd8, argc=1,
    at dl-init.c:118
```

Para hacer frente a esta situación debe instalar Inkscape enlazado con una variante de GLib que contenga información de depuración:

```
guix install inkscape --with-debug-info=glib
```

Esta vez la depuración será mucho más agradable:

```
$ gdb --args sh -c 'exec inkscape'
...
(gdb) b g_getenv
Function "g_getenv" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
```

```
Breakpoint 1 (g_getenv) pending.
(gdb) r
Starting program: /gnu/store/...-profile/bin/sh -c exec\ inkscape
...
(gdb) bt
#0  g_getenv (variable=variable@entry=0x7ffff60c7a2e "GOBJECT_DEBUG") at ../glib-2.62.
#1  0x00007ffff608a7d6 in gobject_init () at ../glib-2.62.6/gobject/gtype.c:4380
#2  gobject_init_ctor () at ../glib-2.62.6/gobject/gtype.c:4493
#3  0x00007ffff7fe275a in call_init (l=<optimized out>, argc=argc@entry=3, argv=argv@e
    env=env@entry=0x7fffffffd0a8) at dl-init.c:72
...
```

¡Así mejor!

Tenga en cuenta que puede haber paquetes en los que la opción `--with-debug-info` no tenga el efecto deseado. See Section 9.1.2 [Opciones de transformación de paquetes], page 184, para obtener más información.

# 18 Using TEX and LATEX

Guix provides packages for the TEX, LATEX, ConTeXt, LuaTeX, and related typesetting systems, taken from the TEX Live distribution (`https://www.tug.org/texlive/`). However, because TEX Live is so huge and because finding one's way in this maze is tricky, so this section provides some guidance on how to deploy the relevant packages to compile TEX and LATEX documents.

TEX Live currently comes in two mutually exclusive flavors in Guix:

- The "monolithic" `texlive` package: it comes with *every single TEX Live package* (roughly 4,200), but it is huge—more than 4 GiB for a single package!
- A "modular" TEX Live distribution, in which you only install the packages, always prefixed with '`texlive-`', you need.

So to insist, these two flavors cannot be combined[1]. If in the modular setting your document does not compile, the solution is not to add the monolithic `texlive` package, but to add the set of missing packages from the modular distribution.

Building a coherent system that provides all the essential tools and, at the same time, satisfies all of its internal dependencies can be a difficult task. It is therefore recommended to start with sets of packages, called *collections*, and *schemes*, the name for collections of collections. The following command lists available schemes and collections (see [Invoking guix package], page 42):

```
guix search texlive-\(scheme\|collection\) | recsel -p name,description
```

If needed, you may then complete your system with individual packages, particularly when they belong to a large collection you're not otherwise interested in.

For instance, the following manifest is a reasonable, yet frugal starting point for a French LATEX user:

```
(specifications->manifest
 '("rubber"

   "texlive-scheme-basic"
   "texlive-collection-latexrecommended"
   "texlive-collection-fontsrecommended"
   "texlive-babel-french"

   ;; From "latexextra" collection.
   "texlive-tabularray"
   ;; From "binextra" collection.
   "texlive-texdoc"))
```

If you come across a document that does not compile in such a basic setting, the main difficulty is finding the missing packages. In this case, `pdflatex` and similar commands tend to fail with obscure error messages along the lines of:

```
doc.tex: File `tikz.sty' not found.
```

[1] No rule without exception! As the monolithic TEX Live does not contain the `biber` executable, it is okay to combine it with `texlive-biber`, which does.

```
doc.tex:7: Emergency stop.
```

or, for a missing font:

```
kpathsea: Running mktexmf phvr7t
! I can't find file `phvr7t'.
```

How do you determine what the missing package is? In the first case, you will find the answer by running:

```
$ guix search texlive tikz
name: texlive-pgf
version: 59745
...
```

In the second case, `guix search` turns up nothing. Instead, you can search the TEX Live package database using the `tlmgr` command:

```
$ tlmgr info phvr7t
tlmgr: cannot find package phvr7t, searching for other matches:

Packages containing `phvr7t' in their title/description:

Packages containing files matching `phvr7t':
helvetic:
        texmf-dist/fonts/tfm/adobe/helvetic/phvr7t.tfm
        texmf-dist/fonts/tfm/adobe/helvetic/phvr7tn.tfm
        texmf-dist/fonts/vf/adobe/helvetic/phvr7t.vf
        texmf-dist/fonts/vf/adobe/helvetic/phvr7tn.vf
tex4ht:
        texmf-dist/tex4ht/ht-fonts/alias/adobe/helvetic/phvr7t.htf
```

The file is available in the TEX Live `helvetic` package, which is known in Guix as `texlive-helvetic`. Quite a ride, but you found it!

# 19 Actualizaciones de seguridad

De manera ocasional, vulnerabilidades importantes de seguridad se descubren en los paquetes de software y deben aplicarse parches. Las desarrolladoras de Guix tratan de seguir las vulnerabilidades conocidas y aplicar parches tan pronto como sea posible en la rama `master` de Guix (todavía no proporcionamos una rama "estable" que contenga únicamente actualizaciones de seguridad). La herramienta `guix lint` ayuda a las desarrolladoras a encontrar versiones vulnerables de paquetes de software en la distribución:

```
$ guix lint -c cve
gnu/packages/base.scm:652:2: glibc@2.21: probablemente vulnerable a CVE-2015-1781, CVE-2015-7547
gnu/packages/gcc.scm:334:2: gcc@4.9.3: probablemente vulnerable a CVE-2015-5276
gnu/packages/image.scm:312:2: openjpeg@2.1.0: probablemente vulnerable a CVE-2016-1923, CVE-2016-1924
...
```

See Section 9.8 [Invocación de guix lint], page 216, para más información.

Guix sigue una disciplina funcional de gestión de paquetes (see Chapter 1 [Introducción], page 1), lo que implica que, cuando se cambia un paquete, *todos los paquetes que dependen de él* deben ser reconstruidos. Esto puede ralentizar de manera significativa el despliegue de correcciones en paquetes básicos como libc o Bash, ya que básicamente la distribución al completo debe reconstruirse. El uso de binarios preconstruidos ayuda (see Section 5.3 [Sustituciones], page 46), pero el despliegue aún puede tomar más tiempo del deseado.

Para afrontar esto, Guix implementa *injertos*, un mecanismo que permite un rápido despliegue de actualizaciones críticas sin los costes asociados con una reconstrucción completa de la distribución. La idea es reconstruir únicamente el paquete que hace falta parchear, y entonces "injertarlo" en los paquetes explícitamente instalados por la usuaria y que previamente hacían referencia al paquete original. El coste de realizar un injerto es menor que una reconstrucción completa de la cadena de dependencias.

Por ejemplo, supongamos que es necesario incorporar una actualización de seguridad en Bash. Las desarrolladoras de Guix proporcionarán una definición de paquete para la versión "corregida" de Bash, digamos `bash-fixed`, de la manera habitual (see Section 8.2 [Definición de paquetes], page 102). Una vez hecho, la definición original del paquete es aumentada con un campo `replacement` que apunta al paquete que contiene la corrección del error:

```
(define bash
  (package
    (name "bash")
    ;; ...
    (replacement bash-fixed)))
```

De ahí en adelante, cualquier paquete que dependa directa o indirectamente de Bash—como informa de ello `guix gc --requisites` (see Section 5.6 [Invocación de guix gc], page 53)—que se instale se "reescribe" automáticamente para hacer referencia a `bash-fixed` en vez de `bash`. Este proceso de injerto toma un tiempo proporcional al tamaño del paquete, normalmente menos de un minuto para un paquete "medio" en una máquina reciente. El injertado es recursivo: cuando una dependencia indirecta requiere un injerto, el injerto se "propagará" hasta el paquete que la usuaria esté instalando.

Actualmente, la longitud del nombre y la versión del injerto y aquella del paquete que reemplaza (*bash-fixed* y *bash* en el ejemplo previo) debe ser igual. Esta restricción viene

principalmente del hecho de que el injertado funciona mediante la aplicación de parches en archivos, incluyendo archivos binarios, directamente. Otras restricciones pueden ser aplicables: por ejemplo, durante la adición de un injerto a un paquete que proporciona una biblioteca compartida, la biblioteca compartida y su reemplazo deben tener el mismo `SONAME` y deben ser compatibles a nivel binario.

La opción de línea de órdenes `--no-grafts` le permite anular voluntariamente el proceso de injerto (see Section 9.1.1 [Opciones comunes de construcción], page 181). Por tanto, la orden:

```
guix build bash --no-grafts
```

devuelve el nombre de archivo del almacén de la versión original de Bash, mientras que:

```
guix build bash
```

devuelve el nombre de archivo del almacén de la versión "corregida", reemplazo de Bash. Esto le permite distinguir entre las dos variantes de Bash.

Para verificar a qué Bash hace referencia su perfil al completo, puede ejecutar (see Section 5.6 [Invocación de guix gc], page 53):

```
guix gc -R $(readlink -f ~/.guix-profile) | grep bash
```

... y compare los nombres de archivo del almacén que obtendrá con los ejemplos previos. Del mismo modo, para una generación completa del sistema Guix:

```
guix gc -R $(guix system build my-config.scm) | grep bash
```

Por último, para comprobar qué versión de Bash están usando los procesos en ejecución, puede usar la orden `lsof`:

```
lsof | grep /gnu/store/.*bash
```

# 20 Lanzamiento inicial

Bootstrapping in our context refers to how the distribution gets built "from nothing". Remember that the build environment of a derivation contains nothing but its declared inputs (see Chapter 1 [Introducción], page 1). So there's an obvious chicken-and-egg problem: how does the first package get built? How does the first compiler get compiled?

It is tempting to think of this question as one that only die-hard hackers may care about. However, while the answer to that question is technical in nature, its implications are wide-ranging. How the distribution is bootstrapped defines the extent to which we, as individuals and as a collective of users and hackers, can trust the software we run. It is a central concern from the standpoint of *security* and from a *user freedom* viewpoint.

El sistema GNU está compuesto principalmente de código C, con libc en su base. El sistema de construcción GNU en sí asume la disponibilidad del shell Bourne y las herramientas de línea de órdenes proporcionadas por GNU Coreutils, Awk, Findutils, 'sed' y 'grep'. Además, los programas de construcción—programas que ejecutan `./configure`, `make`, etc.—están escritos en Scheme Guile (see Section 8.10 [Derivaciones], page 159). Consecuentemente, para ser capaz de construir cualquier cosa, desde cero, Guix depende en binarios preconstruidos de Guile, GCC, Binutils, libc y otros paquetes mencionados anteriormente—los *binarios del lanzamiento inicial*.

Estos binarios del lanzamiento inicial se "dan por supuestos", aunque se pueden volver a crear en caso de ser necesario (see Section 20.2 [Preparación para usar los binarios del lanzamiento inicial], page 728).

## 20.1 The Full-Source Bootstrap

Guix—al igual que otras distribuciones de GNU/Linux—se lanza inicialmente desde un conjunto de binarios de manera tradicional: un shell Bourne, herramientas de línea de órdenes que proporcionan GNU Coreutils, Awk, Findutils, 'sed' y 'grep' y Guile, GCC, Binutils y la biblioteca de C de GNU (see Chapter 20 [Lanzamiento inicial], page 725). Habitualmente dichos binarios se "dan por hecho".

El dar por hecho estos binarios significa que consideramos que son una "semilla" correcta y fiable para la construcción del sistema completo. En esta asunción yace un problema: el tamaño combinado de dichos binarios necesarios para el lanzamiento inicial es de alrededor de 250MB (see Section "Bootstrappable Builds" in *GNU Mes*). Auditar o incluso la inspeccionar de dichos binarios es prácticamente imposible.

For `i686-linux` and `x86_64-linux`, Guix now features a *full-source bootstrap*. This bootstrap is rooted in `hex0-seed` from the Stage0 (`https://savannah.gnu.org/projects/stage0`) package. The hex0 program is minimalist assembler: it reads space-separated hexadecimal digits (nibbles) from a file, possibly including comments, and emits on standard output the bytes corresponding to those hexadecimal numbers. The source code of this initial hex0 program is a file called `hex0_x86.hex0` (`https://github.com/oriansj/bootstrap-seeds/blob/master/POSIX/x86/hex0_x86.hex0`) and is written in the `hex0` language.

Hex0 is self-hosting, which means that it can build itself:

```
./hex0-seed hex0_x86.hex0 hex0
```

Hex0 it is the ASCII-equivalent of the binary program and can be produced by doing something much like:

```
sed 's/[;#].*$//g' hex0_x86.hex0 | xxd -r -p > hex0
chmod +x hex0
```

It is because of this ASCII-binary equivalence that we can bless this initial 357-byte binary as source, and hence 'full-source bootstrap".

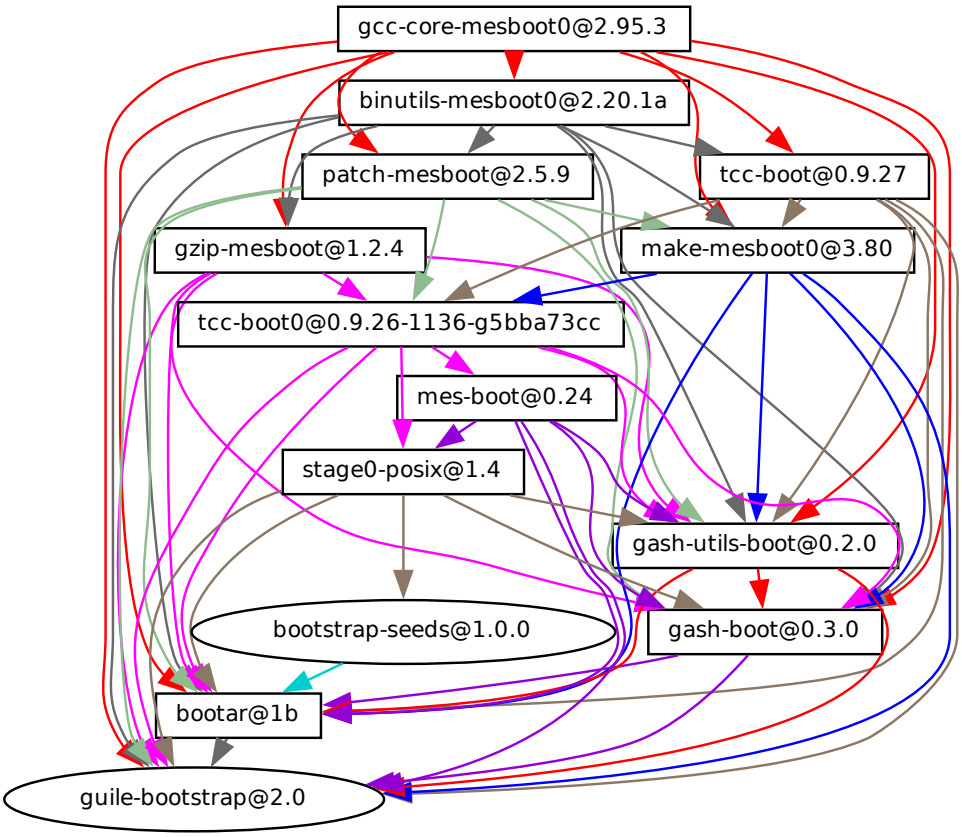
The bootstrap then continues: `hex0` builds `hex1` and then on to `M0`, `hex2`, `M1`, `mescc-tools` and finally `M2-Planet`. Then, using `mescc-tools`, `M2-Planet` we build Mes (see *GNU Mes*, a Scheme interpreter and C compiler in Scheme). From here on starts the more traditional `C`-based bootstrap of the GNU System.

Another step that Guix has taken is to replace the shell and all its utilities with implementations in Guile Scheme, the *Scheme-only bootstrap*. Gash (see Section "Gash" in *The Gash manual*) is a POSIX-compatible shell that replaces Bash, and it comes with Gash Utils which has minimalist replacements for Awk, the GNU Core Utilities, Grep, Gzip, Sed, and Tar.

Building the GNU System from source is currently only possible by adding some historical GNU packages as intermediate steps[1]. As Gash and Gash Utils mature, and GNU packages become more bootstrappable again (e.g., new releases of GNU Sed will also ship as gzipped tarballs again, as alternative to the hard to bootstrap `xz`-compression), this set of added packages can hopefully be reduced again.

A continuación se encuentra el grafo de dependencias generado para `gcc-core-mesboot0`, el compilador del lanzamiento inicial usado para el lanzamiento inicial tradicional del resto del sistema Guix.

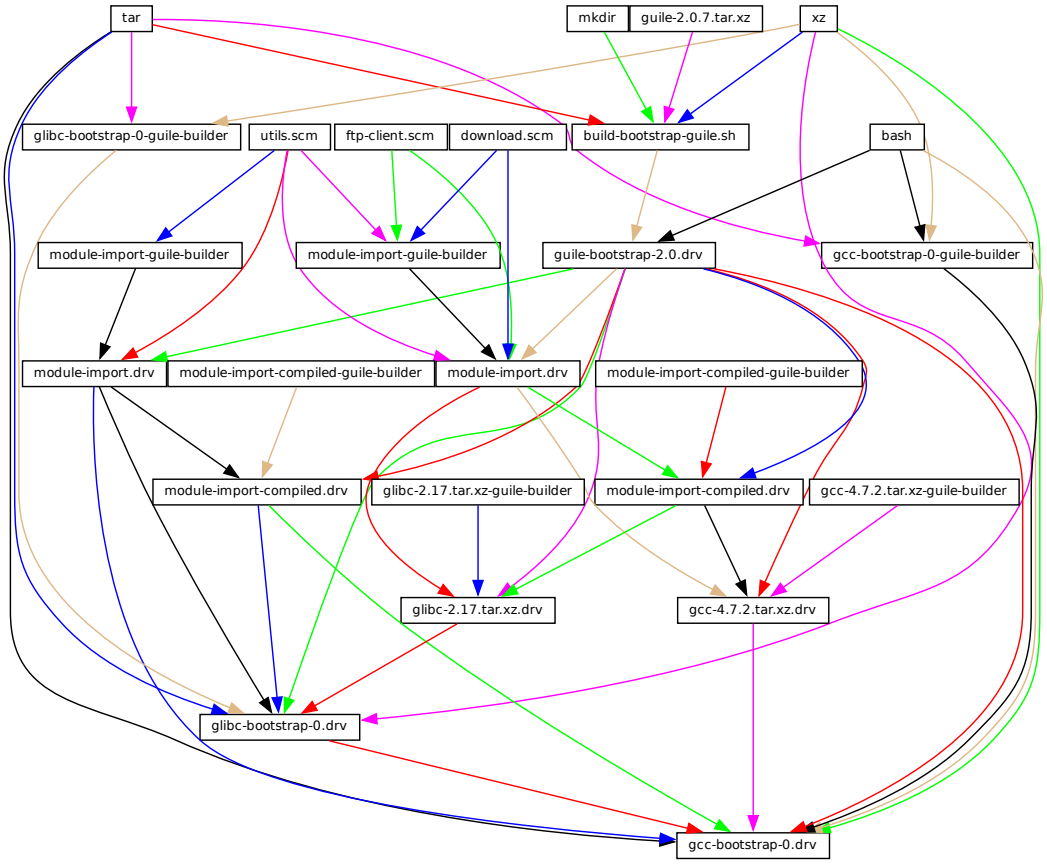
[1] Packages such as `gcc-2.95.3`, `binutils-2.14`, `glibc-2.2.5`, `gzip-1.2.4`, `tar-1.22`, and some others. For details, see `gnu/packages/commencement.scm`.




Work is ongoing to bring these bootstraps to the `arm-linux` and `aarch64-linux` architectures and to the Hurd.

If you are interested, join us on '`#bootstrappable`' on the Libera.Chat IRC network or discuss on `bug-mes@gnu.org` or `gash-devel@nongnu.org`.

## 20.2 Preparación para usar los binarios del lanzamiento inicial




La figura previa muestra el auténtico inicio del grafo de dependencias de la distribución, correspondiente a las definiciones de paquete del módulo `(gnu packages bootstrap)`. Un gráfico similar puede generarse con `guix graph` (see Section 9.10 [Invocación de guix graph], page 221), más o menos así:

```
guix graph -t derivation \
  -e '(@@ (gnu packages bootstrap) %bootstrap-gcc)' \
  | dot -Tps > gcc.ps
```

o, para la semilla binaria aún más reducida del lanzamiento inicial

```
guix graph -t derivation \
  -e '(@@ (gnu packages bootstrap) %bootstrap-mes)' \
  | dot -Tps > mes.ps
```

En este nivel de detalle, las cosas son ligeramente complejas. Primero, Guile en sí consiste en un ejecutable ELF, junto a muchas fuentes y archivos compilados Scheme que se cargan dinámicamente durante la ejecución. Esto se almacena en el archivador tar

`guile-2.0.7.tar.xz` mostrado en este grafo. Este archivador es parte de la distribución de "fuentes" de Guix, y se inserta en el almacén con `add-to-store` (see Section 8.9 [El almacén], page 157).

¿Pero cómo escribimos una derivación que extraiga este archivador y lo añada al almacén? Para resolver este problema, la derivación `guile-bootstrap-2.0.drv`—la primera en construirse—usa `bash` como su constructor, que ejecuta `build-bootstrap-guile.sh`, que a su vez llama a `tar` para extraer el archivador. Por tanto, `bash`, `tar`, `xz` y `mkdir` son binarios enlazados estáticamente, también parte de la distribución de fuentes de Guix, cuyo único propósito es permitir la extracción del archivador de Guile.

Una vez que`guile-bootstrap-2.0.drv` se ha construido, tenemos un Guile funcional que se puede usar para ejecutar los programas de construcción siguientes. Su primera tarea es descargar los archivadores qu contienen los otros binarios preconstruidos—esto es lo que las derivaciones `.tar.xz.drv` hacen. Módulos Guix como `ftp-client.scm` se usan para este propósito. Las derivaciones `module-import.drv` importan esos módulos en un directorio del almacén, manteniendo la distribución de carpetas. Las derivaciones `module-import-compiled.drv` compilan esos módulos, y los escriben en un directorio con la distribución de carpetas correcta. Esto corresponde al parámetro `#:modules` de `build-expression->derivation` (see Section 8.10 [Derivaciones], page 159).

Finalmente, los archivadores tar son extraídos por las derivaciones `gcc-bootstrap-0.drv`, `glibc-bootstrap-0.drv`, or `bootstrap-mes-0.drv` y `bootstrap-mescc-tools-0.drv`, hasta el punto en el que disponemos de una cadena de herramientas C funcional.

## Construcción de las herramientas de construcción

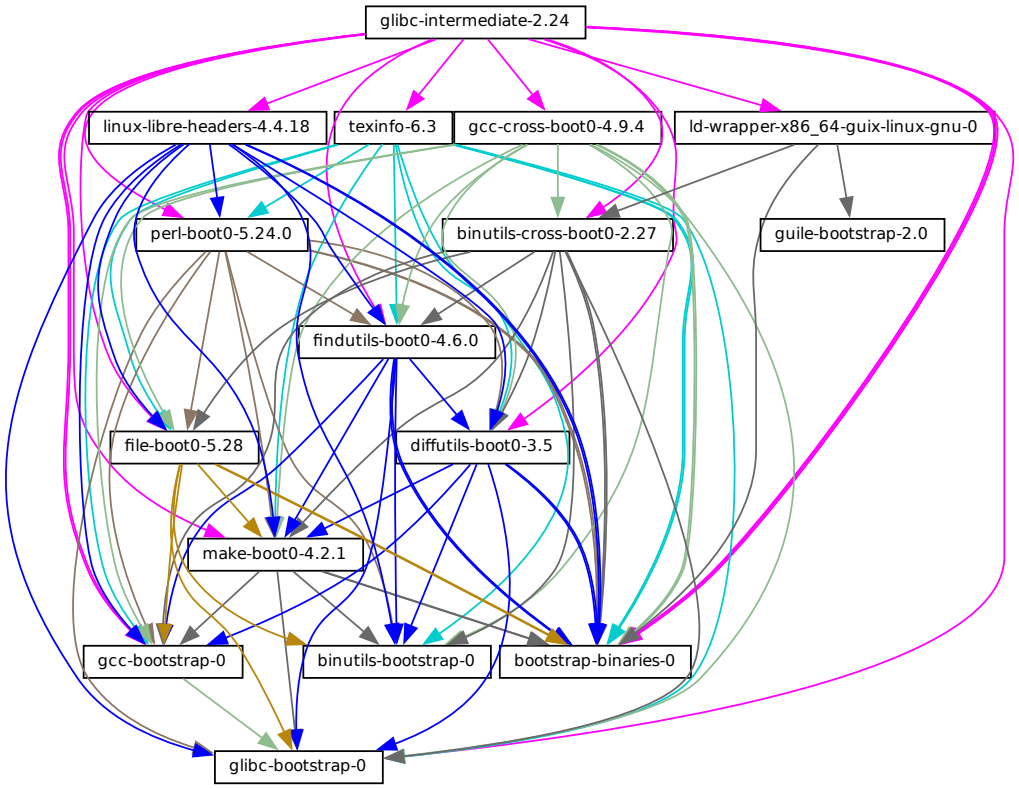
El lanzamiento inicial está completo cuando tenemos una cadena de herramientas completa que no depende en las herramientas preconstruidas del lanzamiento inicial descritas previamente. Este requisito de no-dependencia se verifica comprobando si los archivos de la cadena de herramientas final contienen referencias a directorios de `/gnu/store` de las entradas del lanzamiento. El proceso que lleva a esta cadena de herramientas "final" es descrito por las definiciones de paquetes encontradas en el módulo `(gnu packages commencement)`.

La orden `guix graph` nos permite "distanciarnos" en comparación con el grafo previo, mirando al nivel de objetos de paquetes en vez de derivaciones individuales—recuerde que un paquete puede traducirse en varias derivaciones, típicamente una derivación para descargar sus fuentes, una para construir los módulos Guile que necesita y uno para realmente construir el paquete de las fuentes. La orden:

```
guix graph -t bag \
  -e '(@@ (gnu packages commencement)
          glibc-final-with-bootstrap-bash)' | xdot -
```

muestra el grafo de dependencias que lleva a la biblioteca C "final"[2], mostrado a continuación.

[2] Puede haberse dado cuenta de la etiqueta `glibc-intermediate`, sugiriendo que no es *completamente* final, pero como es una buena aproximación, la consideraremos final.




La primera herramienta que se construye con los binarios del lanzamiento inicial es GNU Make—marcado como `make-boot0` en el grafo—, que es un pre-requisito para todos los paquetes siguientes. Una vez hecho se construyen Findutils y Diffutils.

Después viene la primera fase de Binutils y GCC, construidas como herramientas pseudo-cruzadas—es decir, con `--target` igual a `--host`. Se usan para construir libc. Gracias a este truco de compilación cruzada, se garantiza que esta libc no tendrá ninguna referencia a la cadena de herramientas inicial.

Posteriormente se construyen las herramientas Binutils y GCC (no mostradas previamente) finales. GCC usa `ld` de la construcción final de Binutils y enlazan los programas contra la libc recién construía. Esta cadena de herramientas se usa para construir otros paquetes usados por Guix y el sistema de construcción GNU: Guile, Bash, Coreutils, etc.

¡Y voilà! En este punto tenemos un conjunto completo de herramientas de construcción esperadas por el sistema de construcción GNU. Están en la variable `%final-inputs` del módulo `(gnu packages commencement)`, y se usan implícitamente por cualquier paquete que use `gnu-build-system` (see Section 8.5 [Sistemas de construcción], page 123).

## Construir los binarios de lanzamiento

Debido a que la cadena de herramientas final no depende de los binarios de lanzamiento, estos rara vez necesitan ser actualizados. No obstante, es útil tener una forma automatizada de producirlos en caso de que se dé una actualización, y esto es lo que proporciona el módulo `(gnu packages make-bootstrap)`.

La siguiente orden construye los archivadores que contienen los binarios de lanzamiento (Binutils, GCC, glibc para el lanzamiento inicial tradicional y linux-libre-headers, bootstrap-mescc-tools y bootstrap-mes para el lanzamiento inicial basado en la semilla binaria reducida, y Guile y un archivador que contiene una mezcla de Coreutils y otras herramientas básicas de línea de órdenes):

```
guix build bootstrap-tarballs
```

Los archivadores "tar" generados son aquellos a cuya referencia debe encontrarse en el módulo `(gnu packages bootstrap)` mencionado al inicio de esta sección.

¿Todavía aquí? Entonces quizá se habrá empezado a preguntar: ¿cuándo llegamos a un punto fijo? ¡Esa es una pregunta interesante! La respuesta es desconocida, pero si pudiese investigar más a fondo (y tiene unos recursos computacionales y de almacenamiento significativos para hacerlo) háganoslo saber.

## Reducción del conjunto de binarios de lanzamiento

Nuestros binarios de lanzamiento actualmente incluyen GCC, GNU Libc, Guile, etc. ¡Eso es mucho código binario! ¿Por qué es eso un problema? Es un problema porque esos grandes fragmentos de código binario no son auditables en la práctica, lo que hace difícil establecer qué código fuente los produjo. Cada binario no-auditable también nos deja vulnerables a puertas traseras en los compiladores, como describió Ken Thompson en su publicación de 1984 *Reflections on Trusting Trust*.

Esto se mitiga por el hecho de que nuestros binarios de lanzamiento fueron generados por una revisión anterior de Guix. No obstante, esto no posee el nivel de transparencia que obtenemos en el resto del grado de dependencias de los paquetes, donde Guix siempre nos da una asociación de fuente-a-binario. Por lo tanto, nuestro objetivo es reducir el conjunto de binarios de lanzamiento al mínimo posible.

El sitio web Bootstrappable.org (`https://bootstrappable.org`) enumera proyectos en activo realizándolo. Uno de ellos está a punto de sustituir el GCC de lanzamiento con una secuencia de ensambladores, interpretes y compiladores de complejidad incremental, que pueden ser construidos desde las fuentes empezando con un código ensamblador simple y auditable.

Our first major achievement is the replacement of GCC, the GNU C Library and Binutils by MesCC-Tools (a simple hex linker and macro assembler) and Mes (see *GNU Mes*, a Scheme interpreter and C compiler in Scheme). Neither MesCC-Tools nor Mes can be fully bootstrapped yet and thus we inject them as binary seeds. We call this the Reduced Binary Seed bootstrap, as it has halved the size of our bootstrap binaries! Also, it has eliminated the C compiler binary; i686-linux and x86_64-linux Guix packages are now bootstrapped without any binary C compiler.

Se está trabajando en hacer que MesCC-Tools y Mes puedan lanzarse inicialmente de manera completa, y también se buscan otros binarios para el lanzamiento inicial. ¡Su ayuda es bienvenida!

# 21 Transportar a una nueva plataforma

Como se explicó previamente, la distribución GNU es autocontenida, lo cual se consigue dependiendo de unos "binarios del lanzamiento inicial" preconstruidos (see Chapter 20 [Lanzamiento inicial], page 725). Estos binarios son específicos para un núcleo del sistema operativo, arquitectura de la CPU e interfaz binaria de aplicaciones (ABI). Por tanto, para transportar la distribución a una nueva plataforma que no está soportada todavía, se deben construir estos binarios del lanzamiento inicial, y actualizar el módulo `(gnu packages bootstrap)` para usarlos en dicha plataforma.

Por suerte, Guix puede *compilar de forma cruzada* esos binarios del lanzamiento inicial. Cuando todo va bien, y asumiendo que la cadena de herramientas GNU soporta para la plataforma deseada, esto puede ser tan simple como ejecutar una orden así:

```
guix build --target=armv5tel-linux-gnueabi bootstrap-tarballs
```

For this to work, it is first required to register a new platform as defined in the `(guix platform)` module. A platform is making the connection between a GNU triplet (see Section "Specifying Target Triplets" in *Autoconf*), the equivalent *system* in Nix notation, the name of the *glibc-dynamic-linker*, and the corresponding Linux architecture name if applicable (see Chapter 15 [Platforms], page 708).

Once the bootstrap tarball are built, the `(gnu packages bootstrap)` module needs to be updated to refer to these binaries on the target platform. That is, the hashes and URLs of the bootstrap tarballs for the new platform must be added alongside those of the currently supported platforms. The bootstrap Guile tarball is treated specially: it is expected to be available locally, and `gnu/local.mk` has rules to download it for the supported architectures; a rule for the new platform must be added as well.

En la práctica puede haber algunas complicaciones. Primero, puede ser que la tripleta extendida GNU que especifica un ABI (como el sufijo `eabi` previamente) no es reconocida por todas las herramientas GNU. Típicamente, glibc reconoce algunas de ellas, mientras que GCC usa una opción de configuración extra `--with-abi` (vea `gcc.scm` para ejemplos de como manejar este caso). En segundo lugar, algunos de los paquetes necesarios pueden fallar en su construcción para dicha plataforma. Por último, los binarios generados pueden estar defectuosos por alguna razón.

# 22 Contribuir

Este proyecto es un esfuerzo colaborativo, y ¡necesitamos su ayuda para que crezca! Por favor, contacte con nosotros en `guix-devel@gnu.org` y en `#guix` en la red IRC Libera Chat. Estamos abiertos a ideas, informes de errores, parches y cualquier cosa que pueda ser de ayuda para el proyecto. Especialmente se agradece ayuda con la creación de paquetes (see Section 22.8 [Pautas de empaquetamiento], page 748).

Queremos proporcionar un entorno cálido, amistoso y libre de acoso, para que cualquiera pueda contribuir al máximo de sus capacidades. Para este fin nuestro proyecto usa un "Acuerdo de Contribución", que fue adaptado de `https://contributor-coventant.org`. Se puede encontrar una versión local en el archivo `CODE-OF-CONDUCT` del árbol de fuentes.

Las contribuidoras no están obligadas a usar su nombre legal en los parches ni en la comunicación on-line; pueden usar cualquier nombre o seudónimo de su elección.

## 22.1 Requisitos

You can easily hack on Guix itself using Guix and Git, which we use for version control (see Section 22.2 [Construcción desde Git], page 735).

But when packaging Guix for foreign distros or when bootstrapping on systems without Guix, and if you decide to not just trust and install our readily made binary (see Section 2.1 [Instalación binaria], page 4), you can download a release version of our reproducible source tarball and read on.

Esta sección enumera los requisitos para construir Guix desde las fuentes. El procedimiento de construcción de Guix es el mismo que el de otro software GNU, y no está cubierto aquí. Por favor, eche un vistazo a los archivos `README` y `INSTALL` en el árbol de fuentes de Guix para obtener detalles adicionales.

GNU Guix está disponible para descarga desde su sitio web en `http://www.gnu.org/software/guix/`.

GNU Guix depende de los siguientes paquetes:

- GNU Guile (`https://gnu.org/software/guile/`), version 3.0.x, version 3.0.3 or later;
- Guile-Gcrypt (`https://notabug.org/cwebber/guile-gcrypt`), versión 0.1.0 o posterior;
- Guile-GnuTLS (`https://gitlab.com/gnutls/guile/`) (see Section "Guile Preparations" in *GnuTLS-Guile*)[1];
- Guile-SQLite3 (`https://notabug.org/guile-sqlite3/guile-sqlite3`), versión 0.1.0 o posterior;
- Guile-zlib (`https://notabug.org/guile-zlib/guile-zlib`), version 0.1.0 or later;
- Guile-lzlib (`https://notabug.org/guile-lzlib/guile-lzlib`);
- Guile-Avahi (`https://www.nongnu.org/guile-avahi/`);
- Guile-Git (`https://gitlab.com/guile-git/guile-git`), version 0.5.0 or later;
- Git (`https://git-scm.com`) (yes, both!);

[1] The Guile bindings to GnuTLS (`https://gnutls.org/`) were distributed as part of GnuTLS until version 3.7.8 included.

- Guile-JSON (`https://savannah.nongnu.org/projects/guile-json/`) 4.3.0 o posterior;
- GNU Make (`https://www.gnu.org/software/make/`).

Las siguientes dependencias son opcionales:

- La delegación de construcciones (see Section 2.2.2 [Configuración de delegación del daemon], page 7) y `guix copy` (see Section 9.13 [Invocación de guix copy], page 233) dependen de Guile-SSH (`https://github.com/artyom-poptsov/guile-ssh`), versión 0.13.0 o posterior.
- Guile-zstd (`https://notabug.org/guile-zstd/guile-zstd`), for zstd compression and decompression in `guix publish` and for substitutes (see Section 9.11 [Invocación de guix publish], page 226).
- Guile-Semver (`https://ngyro.com/software/guile-semver.html`) for the `crate` importer (see Section 9.5 [Invocación de guix import], page 198).
- Guile-Lib (`https://www.nongnu.org/guile-lib/doc/ref/htmlprag/`) for the `go` importer (see Section 9.5 [Invocación de guix import], page 198) and for some of the "updaters" (see Section 9.6 [Invocación de guix refresh], page 207).
- Cuando libbz2 (`http://www.bzip.org`) está disponible, `guix daemon` puede usarla para comprimir los registros de construcción.

Si no se ha proporcionado `--disable-daemon` a `configure`, los siguientes paquetes también son necesarios:

- GNU libgcrypt (`https://gnupg.org/`);
- SQLite 3 (`https://sqlite.org`);
- g++ de GCC (`https://gcc.gnu.org`), con implementación del estándar C++11

## 22.2 Construcción desde Git

Si quiere picar en el mismo Guix se recomienda usar la última versión del repositorio Git:

```
git clone https://git.savannah.gnu.org/git/guix.git
```

¿Cómo se puede asegurar de que ha obtenido una copia auténtica del repositorio? Para ello ejecute `guix git authenticate`, proporcionando la revisión y la huella de OpenPGP de la *presentación del canal* (see Section 7.5 [Invocación de guix git authenticate], page 99):

```
git fetch origin keyring:keyring
guix git authenticate 9edb3f66fd807b096b48283debdcddccfea34bad \
  "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA"
```

Esta orden termina con un código de salida cero cuando al finalizar correctamente; o imprime un mensaje de error y sale con un código de salida distinto a cero en otro caso.

Como puede ver, nos encontramos ante el problema del huevo y la gallina: es necesario haber instalado Guix. Durante la instalación habitual del sistema Guix (see Chapter 3 [Instalación del sistema], page 21) o Guix sobre otra distribución (see Section 2.1 [Instalación binaria], page 4) debería verificar la firma de OpenPGP del medio de instalación. Este paso es el primer eslabón de la cadena de confianza.

El modo más fácil de preparar un entorno de desarrollo para Guix es, por supuesto, ¡usando Guix! Las siguientes órdenes inician un nuevo intérprete donde todas las dependencias y las variables de entorno apropiadas están listas para picar código en Guix:

```
guix shell -D guix -CPW
```

or even, from within a Git worktree for Guix:

```
guix shell -CPW
```

If `-C` (short for `--container`) is not supported on your system, try `--pure` instead of `-CPW`. See Section 7.1 [Invoking guix shell], page 79, for more information on that command.

Si no puede usar Guix para construir Guix desde una copia de trabajo, son necesarios los paquetes siguientes además de los mencionados en las instrucciones de instalación (see Section 22.1 [Requisitos], page 734).

- GNU Autoconf (`https://gnu.org/software/autoconf/`);
- GNU Automake (`https://gnu.org/software/automake/`);
- GNU Gettext (`https://gnu.org/software/gettext/`);
- GNU Texinfo (`https://gnu.org/software/texinfo/`);
- Graphviz (`https://www.graphviz.org/`);
- GNU Help2man (opcional) (`https://www.gnu.org/software/help2man/`).

En Guix se pueden añadir dependencias adicionales ejecutando en su lugar `guix shell`:

```
guix shell -D guix help2man git strace --pure
```

From there you can generate the build system infrastructure using Autoconf and Automake:

```
./bootstrap
```

Si se produce un error como el siguiente:

```
configure.ac:46: error: possibly undefined macro: PKG_CHECK_MODULES
```

probablemente significa que Autoconf no pudo encontrar el archivo pkg.m4, que proporciona pkg-config. Asegúrese de que `pkg.m4` está disponible. Lo mismo aplica para el conjunto de macros `guile.m4` que proporciona Guile. Por ejemplo, si ha instalado Automake en `/usr/local`, no va a buscar archivos `.m4` en `/usr/share`. En ese caso tiene que ejecutar la siguiente orden:

```
export ACLOCAL_PATH=/usr/share/aclocal
```

See Section "Macro Search Path" in *The GNU Automake Manual* para más información.

Entonces, ejecute:

```
./configure
```

... where `/var` is the normal `localstatedir` value (see Section 8.9 [El almacén], page 157, for information about this) and `/etc` is the normal `sysconfdir` value. Note that you will probably not run `make install` at the end (you don't have to) but it's still important to pass the right `localstatedir` and `sysconfdir` values, which get recorded in the `(guix config)` Guile module.

Finalmente, puede construir Guix y, si se siente inclinado a ello, ejecutar los tests (see Section 22.3 [Ejecución de la batería de pruebas], page 737):

```
make
```

```
make check
```

If anything fails, take a look at installation instructions (see Chapter 2 [Instalación], page 4) or send a message to the mailing list.

De aquí en adelante, puede identificar todos las revisiones incluidas en su copia ejecutando:

```
guix git authenticate \
  9edb3f66fd807b096b48283debdcddccfea34bad \
  "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA"
```

The first run takes a couple of minutes, but subsequent runs are faster. On subsequent runs, you can run the command without any arguments since the *introduction* (the commit ID and OpenPGP fingerprints above) will have been recorded[2]:

```
guix git authenticate
```

When your configuration for your local Git repository doesn't match the default one, you can provide the reference for the `keyring` branch *via* the `-k` option. The following example assumes that you have a Git remote called '`myremote`' pointing to the official repository:

```
guix git authenticate \
  -k myremote/keyring \
  9edb3f66fd807b096b48283debdcddccfea34bad \
  "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA"
```

See Section 7.5 [Invocación de guix git authenticate], page 99, for more information on this command.

> **Nota:** By default, hooks are installed such that `guix git authenticate` is invoked anytime you run `git pull` or `git push`.

Después de actualizar el repositorio, `make` podría fallar con un error similar al del ejemplo siguiente:

```
error: failed to load 'gnu/packages/linux.scm':
ice-9/eval.scm:293:34: In procedure abi-check: #<record-type <origin>>: record ABI mis
```

Esto significa que uno de los tipos de registro que Guix define (en este ejemplo, el registro `origin`) ha cambiado, y todo guix necesita ser recompilado para tener en cuenta ese cambio. Para ello, ejecute `make clean-go` seguido de `make`.

Should `make` fail with an Automake error message after updating, you need to repeat the steps outlined in this section, commencing with `./bootstrap`.

## 22.3 Ejecución de la batería de pruebas

Después de una ejecución exitosa de `configure` y `make`, es una buena idea ejecutar la batería de pruebas. Puede ayudar a encontrar problemas con la configuración o el entorno, o errores en el mismo Guix—e informar de fallos en las pruebas es realmente una buena forma de ayudar a mejorar el software. Para ejecutar la batería de pruebas, teclee:

```
make check
```

Los casos de prueba pueden ejecutarse en paralelo: puede usar la opción `-j` de GNU make para acelerar las cosas. La primera ejecución puede tomar algunos minutos en una máquina

[2] This requires a recent version of Guix, from May 2024 or more recent.

reciente; las siguientes ejecuciones serán más rápidas puesto que el almacén creado para las pruebas ya tendrá varias cosas en la caché.

También es posible ejecutar un subconjunto de las pruebas definiendo la variable de makefile `TESTS` como en el ejemplo:

```
make check TESTS="tests/store.scm tests/cpio.scm"
```

Por defecto, los resultados de las pruebas se muestran a nivel de archivo. Para ver los detalles de cada caso de prueba individual, es posible definir la variable de makefile `SCM_LOG_DRIVER_FLAGS` como en el ejemplo:

```
make check TESTS="tests/base64.scm" SCM_LOG_DRIVER_FLAGS="--brief=no"
```

The underlying SRFI 64 custom Automake test driver used for the 'check' test suite (located at `build-aux/test-driver.scm`) also allows selecting which test cases to run at a finer level, via its `--select` and `--exclude` options. Here's an example, to run all the test cases from the `tests/packages.scm` test file whose names start with "transaction-upgrade-entry":

```
export SCM_LOG_DRIVER_FLAGS="--select=^transaction-upgrade-entry"
make check TESTS="tests/packages.scm"
```

Those wishing to inspect the results of failed tests directly from the command line can add the `--errors-only=yes` option to the `SCM_LOG_DRIVER_FLAGS` makefile variable and set the `VERBOSE` Automake makefile variable, as in:

```
make check SCM_LOG_DRIVER_FLAGS="--brief=no --errors-only=yes" VERBOSE=1
```

The `--show-duration=yes` option can be used to print the duration of the individual test cases, when used in combination with `--brief=no`:

```
make check SCM_LOG_DRIVER_FLAGS="--brief=no --show-duration=yes"
```

See Section "Parallel Test Harness" in *GNU Automake* for more information about the Automake Parallel Test Harness.

En caso de fallo, le rogamos que envíe un correo a `bug-guix@gnu.org` y adjunte el archivo `test-suite.log`. Por favor, especifique la versión de Guix usada así como los números de versión de las dependencias (see Section 22.1 [Requisitos], page 734) en su mensaje.

Guix también viene como una batería de pruebas del sistema completo que prueban instancias completas del sistema Guix. Se puede ejecutar únicamente en sistemas donde Guix ya está instalado, usando:

```
make check-system
```

o, de nuevo, definiendo `TESTS` para seleccionar un subconjunto de las pruebas a ejecutar:

```
make check-system TESTS="basic mcron"
```

Estas pruebas de sistema están definidas en los módulos `(gnu tests ...)`. Funcionan ejecutando el sistema operativo con una instrumentación ligera en una máquina virtual (VM). Pueden ser computacionalmente intensivas o bastante baratas, dependiendo de si hay sustituciones disponibles para sus dependencias (see Section 5.3 [Sustituciones], page 46). Algunas requieren mucho espacio de almacenamiento para alojar las imágenes de la máquina virtual.

De nuevo, en caso de fallos en las pruebas, le rogamos que envíe a `bug-guix@gnu.org` todos los detalles.

## 22.4 Ejecución de Guix antes de estar instalado

Para mantener un entorno de trabajo estable, encontrará útil probar los cambios hechos en su copia de trabajo local sin instalarlos realmente. De esa manera, puede distinguir entre su sombrero de "usuaria final" y el traje de "harapos".

To that end, all the command-line tools can be used even if you have not run `make install`. To do that, you first need to have an environment with all the dependencies available (see Section 22.2 [Construcción desde Git], page 735), and then simply prefix each command with `./pre-inst-env` (the `pre-inst-env` script lives in the top build tree of Guix; see Section 22.2 [Construcción desde Git], page 735, to generate it). As an example, here is how you would build the `hello` package as defined in your working tree (this assumes `guix-daemon` is already running on your system; it's OK if it's a different version):

```
$ ./pre-inst-env guix build hello
```

De manera similar, un ejemplo de una sesión de Guile con los módulos Guix disponibles:

```
$ ./pre-inst-env guile -c '(use-modules (guix utils)) (pk (%current-system))'

;;; ("x86_64-linux")
```

. . . and for a REPL (see Section 8.14 [Using Guix Interactively], page 178):

```
$ ./pre-inst-env guile
scheme@(guile-user)> ,use(guix)
scheme@(guile-user)> ,use(gnu)
scheme@(guile-user)> (define serpientes
                       (fold-packages
                         (lambda (paquete lst)
                           (if (string-prefix? "python"
                                               (package-name paquete))
                               (cons paquete lst)
                               lst))
                         '()))
scheme@(guile-user)> (length serpientes)
$1 = 361
```

Si modifica el código del daemon o código auxiliar, o si `guix-daemon` no se está ejecutando todavía en su sistema, puede ejecutarlo desde el árbol de construción[3]:

```
$ sudo -E ./pre-inst-env guix-daemon --build-users-group=guixbuild
```

El guión `pre-inst-env` proporciona valor a todas las variables de entorno necesarias para permitirlo, incluyendo `PATH` y `GUILE_LOAD_PATH`.

Fíjese que la orden `./pre-inst-env guix pull` *no* actualiza el árbol de fuentes local; simplemente actualiza el enlace `~/.config/guix/latest` (see Section 5.7 [Invocación de guix pull], page 57). Ejecute `git pull` si quiere actualizar su árbol de fuentes local.

A veces, especialmente si ha actualizado recientemente su repositorio, la ejecución de `./pre-inst-env` imprimirá un mensaje similar al siguiente ejemplo:

```
;;; nota: archivo fuente /home/user/projects/guix/guix/progress.scm
```

[3] La opción `-E` a `sudo` asegura que `GUILE_LOAD_PATH` contiene la información correcta para que `guix-daemon` y las herramientas que usa puedan encontrar los módulos Guile que necesitan.

```
;;; más reciente que el compilado /home/user/projects/guix/guix/progress.go
```

Esto es sólo una nota y se puede ignorar con seguridad. Puede deshacerse del mensaje ejecutando `make -j4`. Hasta que lo haga, Guile se ejecutará ligeramente más lento porque interpretará el código en lugar de utilizar archivos de objetos Guile preparados (`.go`).

Puede ejecutar `make` automáticamente mientras trabaja utilizando `watchexec` del paquete `watchexec`. Por ejemplo, para construir de nuevo cada vez que actualice un archivo de paquete, ejecute '`watchexec -w gnu/packages -- make -j4`'.

## 22.5 La configuración perfecta

La configuración perfecta para hackear en Guix es básicamente la configuración perfecta para hacerlo en Guile (see Section "Using Guile in Emacs" in *Guile Reference Manual*). Primero, necesita más que un editor, necesita Emacs (`https://www.gnu.org/software/emacs`), con su potencia aumentada gracias al maravilloso Geiser (`https://nongnu.org/geiser`). Para conseguir esta configuración ejecute:

```
guix install emacs guile emacs-geiser emacs-geiser-guile
```

Geiser allows for interactive and incremental development from within Emacs: code compilation and evaluation from within buffers, access to on-line documentation (docstrings), context-sensitive completion, `M-.` to jump to an object definition, a REPL to try out your code, and more (see Section "Introduction" in *Geiser User Manual*). If you allow Emacs to load the `.dir-locals.el` file at the root of the project checkout, it will cause Geiser to automatically add the local Guix sources to the Guile load path.

Para realmente editar el código, Emacs tiene un modo para Scheme muy limpio. Pero además de eso, no debe perderse Paredit (`http://www.emacswiki.org/emacs/ParEdit`). Provee de facilidades para operar directamente en el árbol sintáctico como elevar una expresión-S o recubrirla, embeber o expulsar la siguiente expresión-S, etc.

We also provide templates for common git commit messages and package definitions in the `etc/snippets` directory. These templates can be used to expand short trigger strings to interactive text snippets. If you use YASnippet (`https://joaotavora.github.io/yasnippet/`), you may want to add the `etc/snippets/yas` snippets directory to the *yas-snippet-dirs* variable. If you use Tempel (`https://github.com/minad/tempel/`), you may want to add the `etc/snippets/tempel/*` path to the *tempel-path* variable in Emacs.

```
;; Suponiendo que el checkout de Guix está en ~/src/guix.
;; Configuración de yasnippet
(with-eval-after-load 'yasnippet
   (add-to-list 'yas-snippet-dirs "~/src/guix/etc/snippets/yas"))
;; Tempel configuration
(con-eval-after-load 'tempel
    ;; Asegúrese de que tempel-path es una lista -- también puede ser una cadena.
    (unless (listp 'tempel-path)
       (setq tempel-path (list tempel-path))
    (add-to-list 'tempel-path "~/src/guix/etc/snippets/tempel/*"))
```

Los fragmentos de mensajes de la revisión dependen de Magit (`https://magit.vc/`) para mostrar los archivos preparados. En la edición del mensaje de la revisión teclee `add` seguido de *`TAB`* (el tabulador) para insertar la plantilla del mensaje de la revisión de adición

de un paquete; teclee `update` seguido de *TAB* para insertar una plantilla de actualización de un paquete; teclee `https` seguido de *TAB* para insertar una plantilla para cambiar la URI de la página de un paquete a HTTPS.

El fragmento principal para `scheme-mode` es activado al teclear `package...` seguido de *TAB*. Este fragmento también inserta el lanzador `origin...` que puede ser expandido de nuevo. El fragmento `origin` puede a su vez insertar otros identificadores de lanzado terminando en `...`, que pueden ser expandidos de nuevo.

También proporcionamos herramientas para la inserción y actualización automática del copyright en `etc/copyright.el`. Puede proporcionar su nombre completo, correo electrónico y cargar el archivo.

```
(setq user-full-name "Alicia Díaz")
(setq user-mail-address "alicia@correo.org")
;; Se asume que la copia trabajo de guix está en ~/src/guix.
(load-file "~/src/guix/etc/copyright.el")
```

Para insertar el aviso de copyright en la línea actual invoque `M-x guix-copyright`.

Para actualizar el aviso de copyright debe especificar una expresión regular de nombres en la variable `copyright-names-regexp`.

```
(setq copyright-names-regexp
      (format "%s <%s>" user-full-name user-mail-address))
```

Puede comprobar si su copyright está actualizado evaluando `M-x copyright-update`. Si desea hacerlo de manera automática tras guardar un archivo añada `(add-hook 'after-save-hook 'copyright-update)` en Emacs.

### 22.5.1 Viewing Bugs within Emacs

Emacs has a nice minor mode called `bug-reference`, which, when combined with '`emacs-debbugs`' (the Emacs package), can be used to open links such as '`<https://bugs.gnu.org/58697>`' or '`<https://issues.guix.gnu.org/58697>`' as bug report buffers. From there you can easily consult the email thread via the Gnus interface, reply or modify the bug status, all without leaving the comfort of Emacs! Below is a sample configuration to add to your `~/.emacs` configuration file:

```
;;; Bug references.
(require 'bug-reference)
(add-hook 'prog-mode-hook #'bug-reference-prog-mode)
(add-hook 'gnus-mode-hook #'bug-reference-mode)
(add-hook 'erc-mode-hook #'bug-reference-mode)
(add-hook 'gnus-summary-mode-hook #'bug-reference-mode)
(add-hook 'gnus-article-mode-hook #'bug-reference-mode)

;;; This extends the default expression (the top-most, first expression
;;; provided to 'or') to also match URLs such as
;;; <https://issues.guix.gnu.org/58697> or <https://bugs.gnu.org/58697>.
;;; It is also extended to detect "Fixes: #NNNNN" git trailers.
(setq bug-reference-bug-regexp
      (rx (group (or (seq word-boundary
                          (or (seq (char "Bb") "ug"
```

```
                                    (zero-or-one " ")
                                    (zero-or-one "#"))
                               (seq (char "Pp") "atch"
                                    (zero-or-one " ")
                                    "#")
                               (seq (char "Ff") "ixes"
                                    (zero-or-one ":")
                                    (zero-or-one " ") "#")
                               (seq "RFE"
                                    (zero-or-one " ") "#")
                               (seq "PR "
                                    (one-or-more (char "a-z+-")) "/"))
                          (group (one-or-more (char "0-9"))
                                 (zero-or-one
                                  (seq "#" (one-or-more
                                            (char "0-9"))))))
                     (seq (? "<") "https://bugs.gnu.org/"
                          (group-n 2 (one-or-more (char "0-9")))
                          (? ">"))
                     (seq (? "<") "https://issues.guix.gnu.org/"
                          (? "issue/")
                          (group-n 2 (one-or-more (char "0-9")))
                          (? ">"))))))
(setq bug-reference-url-format "https://issues.guix.gnu.org/%s")

(require 'debbugs)
(require 'debbugs-browse)
(add-hook 'bug-reference-mode-hook #'debbugs-browse-mode)
(add-hook 'bug-reference-prog-mode-hook #'debbugs-browse-mode)

;; The following allows Emacs Debbugs user to open the issue directly within
;; Emacs.
(setq debbugs-browse-url-regexp
      (rx line-start
          "http" (zero-or-one "s") "://"
          (or "debbugs" "issues.guix" "bugs")
          ".gnu.org" (one-or-more "/")
          (group (zero-or-one "cgi/bugreport.cgi?bug="))
          (group-n 3 (one-or-more digit))
          line-end))

;; Change the default when run as 'M-x debbugs-gnu'.
(setq debbugs-gnu-default-packages '("guix" "guix-patches"))

;; Show feature requests.
(setq debbugs-gnu-default-severities
 '("serious" "important" "normal" "minor" "wishlist"))
```

For more information, refer to Section “Bug Reference” in *The GNU Emacs Manual* and Section “Minor Mode” in *The Debbugs User Guide*.

## 22.6 Alternative Setups

Alternative setups than Emacs may let you work on Guix with a similar development experience and they might work better with the tools you currently use or help you make the transition to Emacs.

The options listed below only provide the alternatives to the Emacs based setup, which is the most widely used in the Guix community. If you want to really understand how is the perfect setup for Guix development supposed to work, we encourage you to read the section before this regardless the editor you choose to use.

### 22.6.1 Guile Studio

Guile Studio is a pre-configured Emacs with mostly everything you need to start hacking in Guile. If you are not familiar with Emacs it makes the transition easier for you.

```
guix install guile-studio
```

Guile Studio comes with Geiser preinstalled and prepared for action.

### 22.6.2 Vim and NeoVim

Vim (and NeoVim) are also packaged in Guix, just in case you decided to go for the evil path.

```
guix install vim
```

If you want to enjoy a similar development experience to that in the perfect setup, you should install several plugins to configure the editor. Vim (and NeoVim) have the equivalent to Paredit, `paredit.vim` (`https://www.vim.org/scripts/script.php?script_id=3998`), that will help you with the structural editing of Scheme files (the support for very large files is not great, though).

```
guix install vim-paredit
```

We also recommend that you run `:set autoindent` so that your code is automatically indented as you type.

For the interaction with Git, `fugitive.vim` (`https://www.vim.org/scripts/script.php?script_id=2975`) is the most commonly used plugin:

```
guix install vim-fugitive
```

And of course if you want to interact with Guix directly from inside of vim, using the built-in terminal emulator, we have our very own `guix.vim` package!

```
guix install vim-guix-vim
```

In NeoVim you can even make a similar setup to Geiser using Conjure (`https://conjure.fun/`) that lets you connect to a running Guile process and inject your code there live (sadly it’s not packaged in Guix yet).

## 22.7 Source Tree Structure

If you're willing to contribute to Guix beyond packages, or if you'd like to learn how it all fits together, this section provides a guided tour in the code base that you may find useful.

Overall, the Guix source tree contains almost exclusively Guile *modules*, each of which can be seen as an independent library (see Section "Modules" in *GNU Guile Reference Manual*).

The following table gives an overview of the main directories and what they contain. Remember that in Guile, each module name is derived from its file name—e.g., the module in file `guix/packages.scm` is called `(guix packages)`.

`guix` This is the location of core Guix mechanisms. To illustrate what is meant by "core", here are a few examples, starting from low-level tools and going towards higher-level tools:

`(guix store)`
: Connecting to and interacting with the build daemon (see Section 8.9 [El almacén], page 157).

`(guix derivations)`
: Creating derivations (see Section 8.10 [Derivaciones], page 159).

`(guix gexps)`
: Writing G-expressions (see Section 8.12 [Expresiones-G], page 167).

`(guix packages)`
: Defining packages and origins (see Section 8.2.1 [Referencia de package], page 105).

`(guix download)`
`(guix git-download)`
: The `url-fetch` and `git-fetch` origin download methods (see Section 8.2.2 [Referencia de origin], page 110).

`(guix swh)`
: Fetching source code from the Software Heritage archive (`https://archive.softwareheritage.org`).

`(guix search-paths)`
: Implementing search paths (see Section 8.8 [Search Paths], page 154).

`(guix build-system)`
: The build system interface (see Section 8.5 [Sistemas de construcción], page 123).

`(guix profiles)`
: Implementing profiles.

`guix/build-system`
: This directory contains specific build system implementations (see Section 8.5 [Sistemas de construcción], page 123), such as:

`(guix build-system gnu)`
: the GNU build system;

`(guix build-system cmake)`
: the CMake build system;

`(guix build-system pyproject)`
: The Python "pyproject" build system.

`guix/build`
: This contains code generally used on the "build side" (see Section 8.12 [Expresiones-G], page 167). This includes code used to build packages or other operating system components, as well as utilities:

`(guix build utils)`
: Utilities for package definitions and more (see Section 8.7 [Utilidades de construcción], page 147).

`(guix build gnu-build-system)`
`(guix build cmake-build-system)`
`(guix build pyproject-build-system)`
: Implementation of build systems, and in particular definition of their build phases (see Section 8.6 [Fases de construcción], page 143).

`(guix build syscalls)`
: Interface to the C library and to Linux system calls.

`guix/scripts`
: This contains modules corresponding to `guix` sub-commands. For example, the `(guix scripts shell)` module exports the `guix-shell` procedure, which directly corresponds to the `guix shell` command (see Section 7.1 [Invoking guix shell], page 79).

`guix/import`
: This contains supporting code for the importers and updaters (see Section 9.5 [Invocación de guix import], page 198, and see Section 9.6 [Invocación de guix refresh], page 207). For example, `(guix import pypi)` defines the interface to PyPI, which is used by the `guix import pypi` command.

The directories we have seen so far all live under `guix/`. The other important place is the `gnu/` directory, which contains primarily package definitions as well as libraries and tools for Guix System (see Chapter 11 [Configuración del sistema], page 242) and Guix Home (see Chapter 13 [Home Configuration], page 671), all of which build upon functionality provided by `(guix ...)` modules[4].

`gnu/packages`
: This is by far the most crowded directory of the source tree: it contains *package modules* that export package definitions (see Section 8.1 [Módulos de paquetes], page 101). A few examples:

`(gnu packages base)`
: Module providing "base" packages: `glibc`, `coreutils`, `grep`, etc.

[4] For this reason, `(guix ...)` modules must generally not depend on `(gnu ...)` modules, with notable exceptions: `(guix build-system ...)` modules may look up packages at run time—e.g., `(guix build-system cmake)` needs to access the `cmake` variable at run time—, `(guix scripts ...)` often rely on `(gnu ...)` modules, and the same goes for some of the `(guix import ...)` modules.

`(gnu packages guile)`
: Guile and core Guile packages.

`(gnu packages linux)`
: The Linux-libre kernel and related packages.

`(gnu packages python)`
: Python and core Python packages.

`(gnu packages python-xyz)`
: Miscellaneous Python packages (we were not very creative).

In any case, you can jump to a package definition using `guix edit` (see Section 9.2 [Invocación de guix edit], page 196) and view its location with `guix show` (see Section 5.2 [Invocación de guix package], page 36).

`gnu/packages/patches`
: This directory contains patches applied against packages and obtained using the `search-patches` procedure.

`gnu/services`
: This contains service definitions, primarily for Guix System (see Section 11.10 [Servicios], page 275) but some of them are adapted and reused for Guix Home as we will see below. Examples:

  `(gnu services)`
  : The service framework itself, which defines the service and service type data types (see Section 11.19.1 [Composición de servicios], page 649).

  `(gnu services base)`
  : "Base" services (see Section 11.10.1 [Servicios base], page 276).

  `(gnu services desktop)`
  : "Desktop" services (see Section 11.10.9 [Servicios de escritorio], page 363).

  `(gnu services shepherd)`
  : Support for Shepherd services (see Section 11.19.4 [Servicios de Shepherd], page 657).

  You can jump to a service definition using `guix system edit` and view its location with `guix system search` (see Section 11.16 [Invocación de guix system], page 634).

`gnu/system`
: These are core Guix System modules, such as:

  `(gnu system)`
  : Defines `operating-system` (see Section 11.3 [Referencia de operating-system], page 253).

  `(gnu system file-systems)`
  : Defines `file-system` (see Section 11.4 [Sistemas de archivos], page 257).

`(gnu system mapped-devices)`
: Defines `mapped-device` (see Section 11.5 [Dispositivos traducidos], page 263).

`gnu/build`
: These are modules that are either used on the "build side" when building operating systems or packages, or at run time by operating systems.

`(gnu build accounts)`
: Creating `/etc/passwd`, `/etc/shadow`, etc. (see Section 11.7 [Cuentas de usuaria], page 268).

`(gnu build activation)`
: Activating an operating system at boot time or reconfiguration time.

`(gnu build file-systems)`
: Searching, checking, and mounting file systems.

`(gnu build linux-boot)`
`(gnu build hurd-boot)`
: Booting GNU/Linux and GNU/Hurd operating systems.

`(gnu build linux-initrd)`
: Creating a Linux initial RAM disk (see Section 11.14 [Disco en RAM inicial], page 624).

`gnu/home` This contains all things Guix Home (see Chapter 13 [Home Configuration], page 671); examples:

`(gnu home services)`
: Core services such as `home-files-service-type`.

`(gnu home services ssh)`
: SSH-related services (see Section 13.3.6 [Secure Shell], page 686).

`gnu/installer`
: This contains the text-mode graphical system installer (see Section 3.5 [Instalación gráfica guiada], page 23).

`gnu/machine`
: These are the *machine abstractions* used by `guix deploy` (see Section 11.17 [Invocación de guix deploy], page 643).

`gnu/tests`
: This contains system tests—tests that spawn virtual machines to check that system services work as expected (see Section 22.3 [Ejecución de la batería de pruebas], page 737).

Last, there's also a few directories that contain files that are *not* Guile modules:

`nix` This is the C++ implementation of `guix-daemon`, inherited from Nix (see Section 2.3 [Invocación de guix-daemon], page 12).

`tests` These are unit tests, each file corresponding more or less to one module, in particular `(guix ...)` modules (see Section 22.3 [Ejecución de la batería de pruebas], page 737).

`doc` This is the documentation in the form of Texinfo files: this manual and the Cookbook. See Section "Writing a Texinfo File" in *GNU Texinfo*, for information on Texinfo markup language.

`po` This is the location of translations of Guix itself, of package synopses and descriptions, of the manual, and of the cookbook. Note that `.po` files that live here are pulled directly from Weblate (see Section 22.16 [Traduciendo Guix], page 775).

`etc` Miscellaneous files: shell completions, support for systemd and other init systems, Git hooks, etc.

With all this, a fair chunk of your operating system is at your fingertips! Beyond `grep` and `git grep`, see Section 22.5 [La configuración perfecta], page 740, on how to navigate code from your editor, and see Section 8.14 [Using Guix Interactively], page 178, for information on how to use Scheme modules interactively. Enjoy!

## 22.8 Pautas de empaquetamiento

La distribución GNU es reciente y puede no disponer de alguno de sus paquetes favoritos. Esta sección describe cómo puede ayudar a hacer crecer la distribución.

Los paquetes de software libre habitualmente se distribuyen en forma de *archivadores de código fuente*—típicamente archivos `tar.gz` que contienen todos los archivos fuente. Añadir un paquete a la distribución significa esencialmente dos cosas: añadir una *receta* que describe cómo construir el paquete, la que incluye una lista de otros paquetes necesarios para la construcción, y añadir *metadatos del paquete* junto a dicha receta, como la descripción y la información de licencias.

En Guix toda esta información está contenida en *definiciones de paquete*. Las definiciones de paquete proporcionan una vista de alto nivel del paquete. Son escritas usando la sintaxis del lenguaje de programación Scheme; de hecho, definimos una variable por cada paquete enlazada a su definición y exportamos esa variable desde un módulo (see Section 8.1 [Módulos de paquetes], page 101). No obstante, un conocimiento profundo de Scheme *no* es un pre-requisito para la creación de paquetes. Para más información obre las definiciones de paquetes, see Section 8.2 [Definición de paquetes], page 102.

Una vez que una definición de paquete está en su lugar, almacenada en un archivo del árbol de fuentes de Guix, puede probarse usando la orden `guix build` (see Section 9.1 [Invocación de guix build], page 181). Por ejemplo, asumiendo que el nuevo paquete se llama `gnuevo`, puede ejecutar esta orden desde el árbol de construcción de Guix (see Section 22.4 [Ejecución de Guix antes de estar instalado], page 739):

```
./pre-inst-env guix build gnuevo --keep-failed
```

El uso de `--keep-failed` facilita la depuración de errores de construcción ya que proporciona acceso al árbol de la construcción fallida. Otra opción útil de línea de órdenes para la depuración es `--log-file`, para acceder al log de construcción.

Si el paquete resulta desconocido para la orden `guix`, puede ser que el archivo fuente contenga un error de sintaxis, o no tenga una cláusula `define-public` para exportar la

variable del paquete. Para encontrar el problema puede cargar el módulo desde Guile para obtener más información sobre el error real:

```
./pre-inst-env guile -c '(use-modules (gnu packages gnuevo))'
```

Once your package builds correctly, please send us a patch (see Section 22.10 [Envío de parches], page 760). Well, if you need help, we will be happy to help you too. Once the patch is committed in the Guix repository, the new package automatically gets built on the supported platforms by our continuous integration system (`https://bordeaux.guix.gnu.org`).

Users can obtain the new package definition simply by running `guix pull` (see Section 5.7 [Invocación de guix pull], page 57). When `bordeaux.guix.gnu.org` is done building the package, installing the package automatically downloads binaries from there (see Section 5.3 [Sustituciones], page 46). The only place where human intervention is needed is to review and apply the patch.

### 22.8.1 Libertad del software

El sistema operativo GNU se ha desarrollado para que las usuarias puedan ejercitar su libertad de computación. GNU es *software libre*, lo que significa que las usuarias tienen las cuatro libertades esenciales (`https://www.gnu.org/philosophy/free-sw.html`): para ejecutar el programa, para estudiar y modificar el programa en la forma de código fuente, para redistribuir copias exactas y para distribuir versiones modificadas. Los paquetes encontrados en la distribución GNU contienen únicamente software que proporciona estas cuatro libertades.

Además, la distribución GNU sigue las directrices de distribución de software libre (`https://www.gnu.org/distros/free-system-distribution-guidelines.html`). Entre otras cosas, estas directrices rechazan firmware no-libre, recomendaciones de software no-libre, y tienen en cuenta formas de tratar con marcas registradas y patentes.

Algunos paquetes originales, que serían de otra manera software libre, contienen un subconjunto pequeño y opcional que viola estas directrices, por ejemplo debido a que ese subconjunto sea en sí código no-libre. Cuando esto sucede, las partes indeseadas son eliminadas con parches o fragmentos de código en la forma `origin` del paquete (see Section 8.2 [Definición de paquetes], page 102). De este modo, `guix build --source` devuelve las fuentes "liberadas" en vez de la versión original de las fuentes.

### 22.8.2 Nombrado de paquetes

Un paquete tiene actualmente dos nombre asociados con él. Primero el nombre de la *Variable Scheme*, seguido de `define-public`. Por este nombre, el paquete se puede conocer por este nombre en el código Scheme, por ejemplo como entrada a otro paquete. El segundo es la cadena en el campo `nombre` de la definición de paquete. Este nombre es el utilizado por los comandos de administración de paquetes como `guix package` y `guix build`.

Ambos normalmente son iguales y corresponden a la conversión a minúsculas del nombre de proyecto elegido por sus creadoras, con los guiones bajos sustituidos por guiones. Por ejemplo, GNUnet está disponible como `gnunet`, y SDL_net como `sdl-net`.

Una excepción notable a esta regla es cuando el nombre del proyecto es un sólo carácter o si existe un proyecto más antiguo mantenido con el mismo nombre —sin importar si ha sido empaquetado para Guix. Utilice el sentido común para hacer que estos nombre no sean

ambiguos y tengan significado. Por ejemplo, el paquete de Guix para la shell llamado "s" hacia arriba es `s-shell` y *not* `s`. No dude en pedir inspiración a sus compañeros hackers.

No añadimos prefijos `lib` para paquetes de bibliotecas, a menos que sean parte del nombre oficial del proyecto. Pero vea Section 22.8.8 [Módulos Python], page 754, y Section 22.8.9 [Módulos Perl], page 755, para reglas especiales que conciernen a los módulos de los lenguajes Python y Perl.

Los nombres de paquetes de tipografías se manejan de forma diferente, see Section 22.8.13 [Tipos de letra], page 758.

### 22.8.3 Versiones numéricas

Normalmente empaquetamos únicamente la última versión de un proyecto dado de software libre. Pero a veces, por ejemplo para versiones de bibliotecas incompatibles, se necesitan dos (o más) versiones del mismo paquete. Estas necesitan nombres diferentes para las variables Scheme. Usamos el nombre como se define en Section 22.8.2 [Nombrado de paquetes], page 749, para la versión más reciente; las versiones previas usan el mismo nombre, añadiendo un `-` y el prefijo menor del número de versión que permite distinguir las dos versiones.

El nombre dentro de la definición de paquete es el mismo para todas las versiones de un paquete y no contiene ningún número de versión.

Por ejemplo, las versiones 2.24.20 y 3.9.12 de GTK+ pueden empaquetarse como sigue:

```
(define-public gtk+
  (package
    (name "gtk+")
    (version "3.9.12")
    ...))
(define-public gtk+-2
  (package
    (name "gtk+")
    (version "2.24.20")
    ...))
```

Si también deseásemos GTK+3.8.2, se empaquetaría como

```
(define-public gtk+-3.8
  (package
    (name "gtk+")
    (version "3.8.2")
    ...))
```

De manera ocasional, empaquetamos instantáneas del sistema de control de versiones (VCS) de las desarrolladoras originales en vez de publicaciones formales. Esto debería permanecer como algo excepcional, ya que son las desarrolladoras originales quienes deben clarificar cual es la entrega estable. No obstante, a veces es necesario. Por tanto, ¿qué deberíamos poner en el campo `version`?

Claramente, tenemos que hacer visible el identificador de la revisión en el VCS en la cadena de versión, pero también debemos asegurarnos que la cadena de versión incrementa monotónicamente de manera que `guix package --upgrade` pueda determinar qué versión

es más moderna. Ya que los identificadores de revisión, notablemente en Git, no incrementan monotónicamente, añadimos un número de revisión que se incrementa cada vez que actualizamos a una nueva instantánea. La versión que resulta debería ser así:

```
2.0.11-3.cabba9e
  ^    ^    ^
  |    |    `-- ID de revisión original
  |    |
  |    `--- revisión del paquete Guix
  |
última versión de publicación
```

It is a good idea to strip commit identifiers in the `version` field to, say, 7 digits. It avoids an aesthetic annoyance (assuming aesthetics have a role to play here) as well as problems related to OS limits such as the maximum shebang length (127 bytes for the Linux kernel). There are helper functions for doing this for packages using `git-fetch` or `hg-fetch` (see below). It is best to use the full commit identifiers in `origin`s, though, to avoid ambiguities. A typical package definition may look like this:

```
(define mi-paquete
  (let ((commit "c3f29bc928d5900971f65965feaae59e1272a3f7")
        (revision "1"))          ;Revisión Guix del paquete
    (package
      (version (git-version "0.9" revision commit))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "git://example.org/mi-paquete.git")
                      (commit commit)))
                (sha256 (base32 "1mbikn..."))
                (file-name (git-file-name name version))))
      ;; ...
      )))
```

`git-version` *VERSION REVISION COMMIT* [Procedure]

Return the version string for packages using `git-fetch`.

```
(git-version "0.2.3" "0" "93818c936ee7e2f1ba1b315578bde363a7d43d05")
⇒ "0.2.3-0.93818c9"
```

`hg-version` *VERSION REVISION CHANGESET* [Procedure]

Devuelve la cadena de versión de los paquetes utilizando `hg-fetch`. Funciona de la misma manera que `git-version`.

### 22.8.4 Sinopsis y descripciones

Como hemos visto previamente, cada paquete en GNU Guix incluye una sinopsis y una descripción (see Section 8.2 [Definición de paquetes], page 102). Las sinopsis y descripciones son importantes: son en lo que `guix package --search` busca, y una pieza crucial de información para ayudar a las usuarias a determinar si un paquete dado cubre sus necesidades. Consecuentemente, las empaquetadoras deben prestar atención a qué se incluye en ellas.

Las sinopsis deben empezar con mayúscula y no deben terminar con punto. No deben empezar con un artículo que habitualmente no aporta nada; por ejemplo, se prefiere "Herramienta para chiribizar" sobre "Una herramienta que chiribiza archivos". La sinopsis debe decir qué es el paquete—por ejemplo, "Utilidades básicas GNU (archivos, texto, intérprete de consola)"—o para qué se usa—por ejemplo, la sinopsis de GNU grep es "Imprime líneas aceptadas por un patrón".

Tenga en cuenta que las sinopsis deben tener un claro significado para una audiencia muy amplia. Por ejemplo, "Manipula la alineación en el formato SAM" puede tener sentido para una investigadora de bioinformática con experiencia, pero puede ser de poca ayuda o incluso llevar a confusión a una audiencia no-especializada. Es una buena idea proporcionar una sinopsis que da una idea del dominio de aplicación del paquete. En ese ejemplo, esto podría ser algo como "Manipula la alineación de secuencias de nucleótidos", lo que con suerte proporcionará a la usuaria una mejor idea sobre si esto es lo que está buscando.

Las descripciones deben tener entre cinco y diez líneas. Use frases completas, y evite usar acrónimos sin introducirlos previamente. Por favor evite frases comerciales como "líder mundial", "de potencia industrial" y "siguiente generación", y evite superlativos como "el más avanzado"—no son útiles para las usuarias que buscan un paquete e incluso pueden sonar sospechosas. En vez de eso, intente ceñirse a los hechos, mencionando casos de uso y características.

Las descripciones pueden incluir marcado Texinfo, lo que es útil para introducir ornamentos como `@code` o `@dfn`, listas de puntos o enlaces (see Section "Overview" in *GNU Texinfo*). Por consiguiente, debe ser cuidadosa cuando use algunos caracteres, por ejemplo '`@`' y llaves, que son los caracteres especiales básicos en Texinfo (see Section "Special Characters" in *GNU Texinfo*). Las interfaces de usuaria como `guix show` se encargan de su correcta visualización.

Synopses and descriptions are translated by volunteers at Weblate (`https://translate.fedoraproject.org/projects/guix/packages`) so that as many users as possible can read them in their native language. User interfaces search them and display them in the language specified by the current locale.

Para permitir a `xgettext` extraerlas como cadenas traducibles, las sinopsis y descripciones *deben ser cadenas literales*. Esto significa que no puede usar `string-append` o `format` para construir estas cadenas:

```
(package
  ;; ...
  (synopsis "Esto es traducible")
  (description (string-append "Esto " "*no*" " es traducible.")))
```

La traducción requiere mucho trabajo, por lo que, como empaquetadora, le rogamos que ponga incluso más atención a sus sinopsis y descripciones ya que cada cambio puede suponer trabajo adicional para las traductoras. Para ayudarlas, es posible hacer recomendaciones o instrucciones insertando comentarios especiales como este (see Section "xgettext Invocation" in *GNU Gettext*):

```
;; TRANSLATORS: "X11 resize-and-rotate" should not be translated.
(description "ARandR is designed to provide a simple visual front end
for the X11 resize-and-rotate (RandR) extension. ...")
```

### 22.8.5 `snippets` frente a fases

La frontera entre el uso de un fragmento de código para la modificación de un origen (`snippet`) frente a una fase de construcción puede ser ténue. Los fragmentos de código para el origen se usan habitualmente para eliminar archivos no deseados, como bibliotecas incluidas, fuentes no libres, o simplemente para aplicar sustituciones simples. Las fuentes que derivan de un origen deben producir unas fuentes capaces de compilar en cualquier sistema que permita el paquete original (es decir, actuar como la fuente correspondiente). En particular, los `snippet` del campo `origin` no deben incluir elementos del almacén en las fuentes; esos parches deben llevarse a cabo en las fases de construcción. Véase el registro `origin` para obtener más información (see Section 8.2.2 [Referencia de origin], page 110).

### 22.8.6 Cyclic Module Dependencies

While there cannot be circular dependencies between packages, Guile's lax module loading mechanism allows circular dependencies between Guile modules, which doesn't cause problems as long as the following conditions are followed for two modules part of a dependency cycle:

1. Macros are not shared between the co-dependent modules
2. Top-level variables are only referenced in delayed (*thunked*) package fields: `arguments`, `native-inputs`, `inputs`, `propagated-inputs` or `replacement`
3. Procedures referencing top-level variables from another module are not called at the top level of a module themselves.

Straying away from the above rules may work while there are no dependency cycles between modules, but given such cycles are confusing and difficult to troubleshoot, it is best to follow the rules to avoid introducing problems down the line.

Here is a common trap to avoid:

```
(define-public avr-binutils
  (package
    (inherit (cross-binutils "avr"))
    (name "avr-binutils")))
```

In the above example, the `avr-binutils` package was defined in the module `(gnu packages avr)`, and the `cross-binutils` procedure in `(gnu packages cross-base)`. Because the `inherit` field is not delayed (thunked), it is evaluated at the top level at load time, which is problematic in the presence of module dependency cycles. This could be resolved by turning the package into a procedure instead, like:

```
(define (make-avr-binutils)
  (package
    (inherit (cross-binutils "avr"))
    (name "avr-binutils")))
```

Care would need to be taken to ensure the above procedure is only ever used in a package delayed fields or within another procedure also not called at the top level.

### 22.8.7 Paquetes Emacs

Los paquetes Emacs deberían usar preferentemente el sistema de construcción de Emacs (see [emacs-build-system], page 141), por uniformidad y por los beneficios que proporcionan

sus fases de construcción, tales como la autogeneración del fichero de autocargas y la compilación de bytes de las fuentes. Debido a que no hay una forma estandarizada de ejecutar un conjunto de pruebas para los paquetes Emacs, las pruebas están deshabilitadas por defecto. Cuando un conjunto de pruebas está disponible, debe ser habilitado estableciendo el argumento `#:tests?` a `#true`. Por defecto, el comando para ejecutar la prueba es `make check`, pero se puede especificar cualquier comando mediante el argumento `#:test-command`. El argumento `#:test-command` espera una lista que contenga un comando y sus argumentos, para ser invocado durante la fase `check`.

Las dependencias de Elisp de los paquetes de Emacs se proporcionan normalmente como `propagated-inputs` cuando se requieren en tiempo de ejecución. En cuanto a otros paquetes, las dependencias de construcción o de prueba deben especificarse como `native-inputs`.

Los paquetes de Emacs a veces dependen de directorios de recursos que deben instalarse junto con los archivos de Elisp. El argumento `#:include` puede utilizarse para este fin, especificando una lista de expresiones regulares que deben coincidir. La mejor práctica cuando se utiliza el argumento `#:include` es ampliar en lugar de anular su valor por defecto (accesible a través de la variable `%default-include`). Como ejemplo, un paquete de extensión de yasnippet suele incluir un directorio `snippets`, que podría copiarse en el directorio de instalación utilizando:

```
#:include (cons "^snippets/" %default-include)
```

When encountering problems, it is wise to check for the presence of the `Package-Requires` extension header in the package main source file, and whether any dependencies and their versions listed therein are satisfied.

### 22.8.8 Módulos Python

Actualmente empaquetamos Python 2 y Python 3, bajo los nombres de variable Scheme `python-2` y `python` como se explica en Section 22.8.3 [Versiones numéricas], page 750. Para evitar confusiones y conflictos de nombres con otros lenguajes de programación, parece deseable que el nombre de paquete para un módulo Python contenga la palabra `python`.

Some modules are compatible with only one version of Python, others with both. If the package Foo is compiled with Python 3, we name it `python-foo`. If it is compiled with Python 2, we name it `python2-foo`. Python 2 packages are being removed from the distribution; please do no not submit any new Python 2 packages.

Si un proyecto ya contiene la palabra `python`, la eliminamos; por ejemplo, el módulo python-dateutil se empaqueta con los nombres `python-dateutil` y `python2-dateutil`. Si el nombre del proyecto empieza con `py` (por ejemplo `pytz`), este se mantiene y el prefijo es el especificado anteriormente..

> **Nota:** Currently there are two different build systems for Python packages in Guix: *python-build-system* and *pyproject-build-system*. For the longest time, Python packages were built from an informally specified `setup.py` file. That worked amazingly well, considering Python's success, but was difficult to build tooling around. As a result, a host of alternative build systems emerged and the community eventually settled on a formal standard (`https://peps.python.org/pep-0517/`) for specifying build requirements. *pyproject-build-system* is Guix's implementation of this standard. It is considered "experimental" in that it does not yet support all the various PEP-517 *build backends*, but you

are encouraged to try it for new Python packages and report any problems. It will eventually be deprecated and merged into *python-build-system*.

#### 22.8.8.1 Especificación de dependencias

Dependency information for Python packages is usually available in the package source tree, with varying degrees of accuracy: in the `pyproject.toml` file, the `setup.py` file, in `requirements.txt`, or in `tox.ini` (the latter mostly for test dependencies).

Su misión, cuando escriba una receta para un paquete Python, es asociar estas dependencias con el tipo apropiado de "entrada" (see Section 8.2.1 [Referencia de package], page 105). Aunque el importador de `pypi` normalmente hace un buen trabajo (see Section 9.5 [Invocación de guix import], page 198), puede querer comprobar la siguiente lista para determinar qué dependencia va dónde.

- We currently package Python with `setuptools` and `pip` installed per default. This is about to change, and users are encouraged to use `python-toolchain` if they want a build environment for Python.

  `guix lint` will warn if `setuptools` or `pip` are added as native-inputs because they are generally not necessary.
- Las dependencias Python requeridas en tiempo de ejecución van en `propagated-inputs`. Típicamente están definidas con la palabra clave `install_requires` en `setup.py`, o en el archivo `requirements.txt`.
- Python packages required only at build time—e.g., those listed under `build-system.requires` in `pyproject.toml` or with the `setup_requires` keyword in `setup.py`—or dependencies only for testing—e.g., those in `tests_require` or `tox.ini`—go into `native-inputs`. The rationale is that (1) they do not need to be propagated because they are not needed at run time, and (2) in a cross-compilation context, it's the "native" input that we'd want.

  Ejemplos son las bibliotecas de pruebas `pytest`, `mock` y `nose`. Por supuesto, si alguno de estos paquetes también se necesita en tiempo de ejecución, necesita ir en `propagated-inputs`.
- Todo lo que no caiga en las categorías anteriores va a `inputs`, por ejemplo programas o bibliotecas C requeridas para construir los paquetes Python que contienen extensiones C.
- Si un paquete Python tiene dependencias opcionales (`extras_require`), queda en su mano decidir si las añade o no, en base a la relación utilidad/sobrecarga (see Section 22.10 [Envío de parches], page 760).

### 22.8.9 Módulos Perl

Los programas ejecutables Perl se nombran como cualquier otro paquete, mediante el uso del nombre oficial en minúsculas. Para paquetes Perl que contienen una única clase, usamos el nombre en minúsculas de la clase, substituyendo todas las ocurrencias de `::` por guiones y agregando el prefijo `perl-`. Por tanto la clase `XML::Parser` se convierte en `perl-xml-parser`. Los módulos que contienen varias clases mantienen su nombre oficial en minúsculas y también se agrega `perl-` al inicio. Dichos módulos tienden a tener la palabra `perl` en alguna parte de su nombre, la cual se elimina en favor del prefijo. Por ejemplo, `libwww-perl` se convierte en `perl-libwww`.

### 22.8.10 Paquetes Java

Los programas Java ejecutables se nombran como cualquier otro paquete, mediante el uso del nombre oficial en minúsculas.

Para evitar confusión y colisiones de nombres con otros lenguajes de programación, es deseable que el nombre del paquete para un paquete Java contenga el prefijo `java-`. Si el proyecto ya tiene la palabra `java`, eliminamos esta; por ejemplo, el paquete `ngsjaga` se empaqueta bajo el nombre `java-ngs`.

Para los paquetes Java que contienen una clase única o una jerarquía pequeña, usamos el nombre de clase en minúsculas, substituyendo todas las ocurrencias de `.` por guiones y agregando el prefijo `java-`. Por tanto la clase `apache.commons.cli` se convierte en el paquete `java-apache-commons-cli`.

### 22.8.11 Crates de Rust

Los programas Rust ejecutables se nombran como cualquier otro paquete, mediante el uso del nombre oficial en minúsculas.

Para evitar colisiones en el espacio de nombres añadimos `rust-` como prefijo al resto de paquetes de Rust. El nombre debe cambiarse a letras minúsculas cuando sea apropiado y los guiones deben mantenerse.

In the rust ecosystem it is common for multiple incompatible versions of a package to be used at any given time, so all package definitions should have a versioned suffix. The versioned suffix is the left-most non-zero digit (and any leading zeros, of course). This follows the "caret" version scheme intended by Cargo. Examples `rust-clap-2`, `rust-rand-0.6`.

Debido a la dificultad a la hora de reusar paquetes de rust como entradas pre-compiladas de otros paquetes, el sistema de construcción de Cargo (see Section 8.5 [Sistemas de construcción], page 123) presenta las palabras clave `#:cargo-inputs` y `cargo-development-inputs` como parámetros del sistema de construcción. Puede servir de ayuda pensar en estos parámetros de manera similar a `propagated-inputs` y `native-inputs`. Las dependencias de rust de `dependencies` y `build-dependencies` deben proporcionarse a través de `#:cargo-inputs`, y `dev-dependencies` deben proporcionarse a través de `#:cargo-development-inputs`. Si un paquete de Rust se enlaza con otras bibliotecas deben proporcionarse como habitualmente en `inputs` y otros campos relacionados.

Se debe tener cuidado a la hora de asegurar que se usan las versiones correctas de las dependencias; para ello intentamos no evitar la ejecución de pruebas o la construcción completa con `#:skip-build?` cuando sea posible. Por supuesto, no siempre es posible, ya que el paquete puede desarrollarse para un sistema operativo distinto, depender de características del compilador de Rust que se construye a diario (Nightly), o la batería de pruebas puede haberse atrofiado desde su lanzamiento.

### 22.8.12 Paquetes Elm

Elm applications can be named like other software: their names need not mention Elm.

Packages in the Elm sense (see `elm-build-system` under Section 8.5 [Sistemas de construcción], page 123) are required use names of the format *author*`/`*project*, where both the *author* and the *project* may contain hyphens internally, and the *author* sometimes contains uppercase letters.

To form the Guix package name from the upstream name, we follow a convention similar to Python packages (see Section 22.8.8 [Módulos Python], page 754), adding an `elm-` prefix unless the name would already begin with `elm-`.

In many cases we can reconstruct an Elm package's upstream name heuristically, but, since conversion to a Guix-style name involves a loss of information, this is not always possible. Care should be taken to add the `'upstream-name` property when necessary so that '`guix import elm`' will work correctly (see Section 9.5 [Invocación de guix import], page 198). The most notable scenarios when explicitly specifying the upstream name is necessary are:

1. When the *author* is `elm` and the *project* contains one or more hyphens, as with `elm/virtual-dom`; and
2. When the *author* contains hyphens or uppercase letters, as with `Elm-Canvas/raster-shapes`—unless the *author* is `elm-explorations`, which is handled as a special case, so packages like `elm-explorations/markdown` do *not* need to use the `'upstream-name` property.

The module `(guix build-system elm)` provides the following utilities for working with names and related conventions:

`elm-package-origin` *elm-name version hash* *Returns a Git origin using the repository naming and tagging* [Procedure]
regime required for a published Elm package with the upstream name *elm-name* at version *version* with sha256 checksum *hash*.

Por ejemplo:

```
(package
  (name "elm-html")
  (version "1.0.0")
  (source
   (elm-package-origin
    "elm/html"
    version
    (base32 "15k1679ja57vvlpinpv06znmrxy09lbhzfkzdc89i01qa8c4gb4a")))
  ...)
```

`elm->package-name` *elm-name* [Procedure]
Returns the Guix-style package name for an Elm package with upstream name *elm-name*.

Note that there is more than one possible *elm-name* for which `elm->package-name` will produce a given result.

`guix-package->elm-name` *package* [Procedure]
Given an Elm *package*, returns the possibly-inferred upstream name, or `#f` the upstream name is not specified via the `'upstream-name` property and can not be inferred by `infer-elm-package-name`.

`infer-elm-package-name` *guix-name* [Procedure]
Given the *guix-name* of an Elm package, returns the inferred upstream name, or `#f` if the upstream name can't be inferred. If the result is not `#f`, supplying it to `elm->package-name` would produce *guix-name*.

### 22.8.13 Tipos de letra

Para tipografías que no se instalan generalmente por una usuaria para propósitos tipográficos, o que se distribuyen como parte de un paquete de software más grande, seguimos las reglas generales de empaquetamiento de software; por ejemplo, esto aplica a las tipografías distribuidas como parte del sistema X.Org o las tipografías que son parte de TeX Live.

Para facilitar a las usuarias la búsqueda de tipografías, los nombres para otros paquetes que contienen únicamente tipografías se construyen como sigue, independientemente del nombre de paquete oficial.

El nombre de un paquete que contiene únicamente una familia tipográfica comienza con `font-`; seguido por el nombre de la tipografía y un guión si la tipografía es conocida, y el nombre de la familia tipográfica, donde los espacios se sustituyen por guiones (y como es habitual, todas las letras mayúsculas se transforman a minúsculas). Por ejemplo, la familia de tipografías Gentium de SIL se empaqueta bajo el nombre de `font-sil-gentium`.

Para un paquete que contenga varias familias tipográficas, el nombre de la colección se usa en vez del nombre de la familia tipográfica. Por ejemplo, las tipografías Liberation consisten en tres familias: Liberation Sans, Liberation Serif y Liberation Mono. Estas se podrían empaquetar por separado bajo los nombres `font-liberation-sans`, etcétera; pero como se distribuyen de forma conjunta bajo un nombre común, preferimos empaquetarlas conjuntamente como `font-liberation`.

En el caso de que varios formatos de la misma familia o colección tipográfica se empaqueten de forma separada, una forma corta del formato, precedida por un guión, se añade al nombre del paquete. Usamos `-ttf` para tipografías TrueType, `-otf` para tipografías OpenType y `-type1` para tipografías Tipo 1 PostScript.

## 22.9 Estilo de codificación

En general nuestro código sigue los Estándares de codificación GNU (see *GNU Coding Standards*). No obstante, no dicen mucho de Scheme, así que aquí están algunas reglas adicionales.

### 22.9.1 Paradigma de programación

El código scheme en Guix está escrito en un estilo puramente funcional. Una excepción es el código que incluye entrada/salida, y procedimientos que implementan conceptos de bajo nivel, como el procedimiento `memoize`.

### 22.9.2 Módulos

Guile modules that are meant to be used on the builder side must live in the `(guix build ...)` name space. They must not refer to other Guix or GNU modules. However, it is OK for a "host-side" module to use a build-side module. As an example, the `(guix search-paths)` module should not be imported and used by a package since it isn't meant to be used

as a "build-side" module. It would also couple the module with the package's dependency graph, which is undesirable.

Los módulos que tratan con el sistema GNU más amplio deben estar en el espacio de nombres `(gnu ...)` en vez de en `(guix ...)`.

### 22.9.3 Tipos de datos y reconocimiento de patrones

La tendencia en el Lisp clásico es usar listas para representar todo, y recorrerlas "a mano" usando `car`, `cdr`, `cadr` y compañía. Hay varios problemas con este estilo, notablemente el hecho de que es difícil de leer, propenso a errores y una carga para informes adecuados de errores de tipado.

Guix code should define appropriate data types (for instance, using `define-record-type*`) rather than abuse lists. In addition, it should use pattern matching, via Guile's `(ice-9 match)` module, especially when matching lists (see Section "Pattern Matching" in *GNU Guile Reference Manual*); pattern matching for records is better done using `match-record` from `(guix records)`, which, unlike `match`, verifies field names at macro-expansion time.

When defining a new record type, keep the *record type descriptor* (RTD) private (see Section "Records" in *GNU Guile Reference Manual*, for more on records and RTDs). As an example, the `(guix packages)` module defines `<package>` as the RTD for package records but it does not export it; instead, it exports a type predicate, a constructor, and field accessors. Exporting RTDs would make it harder to change the application binary interface (because code in other modules might be matching fields by position) and would make it trivial for users to forge records of that type, bypassing any checks we may have in the official constructor (such as "field sanitizers").

### 22.9.4 Formato del código

Cuando escribimos código Scheme, seguimos la sabiduría común entre las programadoras Scheme. En general, seguimos las Reglas de estilo Lisp de Riastradh (`https://mumble.net/~campbell/scheme/style.txt`). Este documento resulta que también describe las convenciones más usadas en el código Guile. Está lleno de ideas y bien escrito, así que recomendamos encarecidamente su lectura.

Algunas formas especiales introducidas en Guix, como el macro `substitute*` tienen reglas de indentación especiales. Estas están definidas en el archivo `.dir-locals.el`, el cual Emacs usa automáticamente. Fíjese que además Emacs-Guix proporciona el modo `guix-devel-mode` que indenta y resalta adecuadamente el código de Guix (see Section "Development" in *The Emacs-Guix Reference Manual*).

Si no usa Emacs, por favor asegúrese de que su editor conoce esas reglas. Para indentar automáticamente una definición de paquete también puede ejecutar:

```
./pre-inst-env guix style package
```

See Section 9.7 [Invoking guix style], page 214, for more information.

Requerimos que todos los procedimientos del nivel superior tengan una cadena de documentación. Este requisito puede relajarse para procedimientos simples privados en el espacio de nombres `(guix build ...)` no obstante.

Los procedimientos no deben tener más de cuatro parámetros posicionales. Use parámetros con palabras clave para procedimientos que toman más de cuatro parámetros.

## 22.10 Envío de parches

Development is done using the Git distributed version control system. Thus, access to the repository is not strictly necessary. We welcome contributions in the form of patches as produced by `git format-patch` sent to the `guix-patches@gnu.org` mailing list (see Section "Submitting patches to a project" in *Git User Manual*). Contributors are encouraged to take a moment to set some Git repository options (see Section 22.10.1 [Configuring Git], page 762) first, which can improve the readability of patches. Seasoned Guix developers may also want to look at the section on commit access (see Section 22.12 [Acceso al repositorio], page 769).

This mailing list is backed by a Debbugs instance, which allows us to keep track of submissions (see Section 22.11 [Tracking Bugs and Changes], page 765). Each message sent to that mailing list gets a new tracking number assigned; people can then follow up on the submission by sending email to *`ISSUE_NUMBER`*`@debbugs.gnu.org`, where *ISSUE_NUMBER* is the tracking number (see Section 22.10.2 [Envío de una serie de parches], page 763).

Le rogamos que escriba los mensajes de revisiones en formato ChangeLog (see Section "Change Logs" in *GNU Coding Standards*); puede comprobar la historia de revisiones en busca de ejemplos.

You can help make the review process more efficient, and increase the chance that your patch will be reviewed quickly, by describing the context of your patch and the impact you expect it to have. For example, if your patch is fixing something that is broken, describe the problem and how your patch fixes it. Tell us how you have tested your patch. Will users of the code changed by your patch have to adjust their workflow at all? If so, tell us how. In general, try to imagine what questions a reviewer will ask, and answer those questions in advance.

Antes de enviar un parche que añade o modifica una definición de un paquete, por favor recorra esta lista de comprobaciones:

1. Si las autoras del paquete software proporcionan una firma criptográfica para el archivo de la versión, haga un esfuerzo para verificar la autenticidad del archivo. Para un archivo de firma GPG separado esto puede hacerse con la orden `gpg --verify`.
2. Dedique algún tiempo a proporcionar una sinopsis y descripción adecuadas para el paquete. See Section 22.8.4 [Sinopsis y descripciones], page 751, para algunas directrices.
3. Ejecute `guix lint` *`paquete`*, donde *paquete* es el nombre del paquete nuevo o modificado, y corrija cualquier error del que informe (see Section 9.8 [Invocación de guix lint], page 216).
4. Run `guix style` *`package`* to format the new package definition according to the project's conventions (see Section 9.7 [Invoking guix style], page 214).
5. Asegúrese de que el paquete compile en su plataforma, usando `guix build` *`package`*.
6. We recommend you also try building the package on other supported platforms. As you may not have access to actual hardware platforms, we recommend using the `qemu-binfmt-service-type` to emulate them. In order to enable it, add the `virtualization` service module and the following service to the list of services in your `operating-system` configuration:

```
(service qemu-binfmt-service-type
 (qemu-binfmt-configuration
```

```
(platforms (lookup-qemu-platforms "arm" "aarch64"))))
```

Una vez hecho esto, reconfigure su sistema.

You can then build packages for different platforms by specifying the `--system` option. For example, to build the "hello" package for the armhf or aarch64 architectures, you would run the following commands, respectively:

```
guix build --system=armhf-linux --rounds=2 hello
guix build --system=aarch64-linux --rounds=2 hello
```

7. Asegúrese de que el paquete no usa copias empaquetadas de software ya disponible como paquetes separados.

   A veces, paquetes incluyen copias embebidas del código fuente de sus dependencias para conveniencia de las usuarias. No obstante, como distribución, queremos asegurar que dichos paquetes efectivamente usan la copia que ya tenemos en la distribución si hay ya una. Esto mejora el uso de recursos (la dependencia es construida y almacenada una sola vez), y permite a la distribución hacer cambios transversales como aplicar actualizaciones de seguridad para un software dado en un único lugar y que afecte a todo el sistema—algo que esas copias embebidas impiden.

8. Take a look at the profile reported by `guix size` (see Section 9.9 [Invocación de guix size], page 219). This will allow you to notice references to other packages unwillingly retained. It may also help determine whether to split the package (see Section 5.4 [Paquetes con múltiples salidas], page 51), and which optional dependencies should be used. In particular, avoid adding `texlive` as a dependency: because of its extreme size, use `texlive-updmap.cfg` procedure instead.

9. Check that dependent packages (if applicable) are not affected by the change; `guix refresh --list-dependent` *package* will help you do that (see Section 9.6 [Invocación de guix refresh], page 207).

10. Compruebe si el proceso de construcción de un paquete es determinista. Esto significa típicamente comprobar si una construcción independiente del paquete ofrece exactamente el mismo resultado que usted obtuvo, bit a bit.

    Una forma simple de hacerlo es construyendo el mismo paquete varias veces seguidas en su máquina (see Section 9.1 [Invocación de guix build], page 181):

    ```
    guix build --rounds=2 mi-paquete
    ```

    Esto es suficiente una clase común de problemas de no-determinismo, como las marcas de tiempo o salida generada aleatoriamente en el resultado de la construcción.

    Another option is to use `guix challenge` (see Section 9.12 [Invocación de guix challenge], page 230). You may run it once the package has been committed and built by `bordeaux.guix.gnu.org` to check whether it obtains the same result as you did. Better yet: Find another machine that can build it and run `guix publish`. Since the remote build machine is likely different from yours, this can catch non-determinism issues related to the hardware—e.g., use of different instruction set extensions—or to the operating system kernel—e.g., reliance on `uname` or `/proc` files.

11. Cuando escriba documentación, por favor use construcciones neutrales de género para

referirse a la gente[5], como singular "they", "their", "them" (`https://en.wikipedia.org/wiki/Singular_they`) y demás.

12. Compruebe que su parche contiene únicamente un conjunto relacionado de cambios. Agrupando cambios sin relación dificulta y ralentiza la revisión.

    Ejemplos de cambios sin relación incluyen la adición de varios paquetes, o una actualización de un paquete junto a correcciones a ese paquete.

13. Please follow our code formatting rules, possibly running `guix style` script to do that automatically for you (see Section 22.9.4 [Formato del código], page 759).

14. Cuando sea posible, use espejos en la URL de las fuentes (see Section 9.3 [Invocación de guix download], page 196). Use URL fiables, no generadas. Por ejemplo, los archivos de GitHub no son necesariamente idénticos de una generación a la siguiente, así que en este caso es normalmente mejor clonar el repositorio. No use el campo `name` en la URL: no es muy útil y si el nombre cambia, la URL probablemente estará mal.

15. Comprueba si Guix se puede construir correctamente (see Section 22.2 [Construcción desde Git], page 735) y trata los avisos, especialmente aquellos acerca del uso de símbolos sin definición.

16. Asegúrese de que sus cambios no rompen Guix y simule `guix pull` con:

    ```
    guix pull --url=/ruta/a/su/copia --profile=/tmp/guix.master
    ```

When posting a patch to the mailing list, use '`[PATCH] ...`' as a subject, if your patch is to be applied on a branch other than `master`, say `core-updates`, specify it in the subject like '`[PATCH core-updates] ...`'.

You may use your email client or the `git send-email` command (see Section 22.10.2 [Envío de una serie de parches], page 763). We prefer to get patches in plain text messages, either inline or as MIME attachments. You are advised to pay attention if your email client changes anything like line breaks or indentation which could potentially break the patches.

Expect some delay when you submit your very first patch to `guix-patches@gnu.org`. You have to wait until you get an acknowledgement with the assigned tracking number. Future acknowledgements should not be delayed.

When a bug is resolved, please close the thread by sending an email to *`ISSUE_NUMBER`*`-done@debbugs.gnu.org`.

### 22.10.1 Configuring Git

If you have not done so already, you may wish to set a name and email that will be associated with your commits (see Section "Telling Git your name" in *Git User Manual*). If you wish to use a different name or email just for commits in this repository, you can use `git config --local`, or edit `.git/config` in the repository instead of `~/.gitconfig`.

[5] NdT: En esta traducción se ha optado por usar el femenino para referirse a *personas*, ya que es el género gramatical de dicha palabra. Aunque las construcciones impersonales pueden adoptarse en la mayoría de casos, también pueden llegar a ser muy artificiales en otros usos del castellano; en ocasiones son directamente imposibles. Algunas construcciones que proponen la neutralidad de género dificultan la lectura automática (-x), o bien dificultan la corrección automática (-e), o bien aumentan significativamente la redundancia y reducen del mismo modo la velocidad en la lectura (-as/os, -as y -os). No obstante, la adopción del genero neutro heredado del latín, el que en castellano se ha unido con el masculino, como construcción neutral de género se considera inaceptable, ya que sería equivalente al "it" en inglés, nada más lejos de la intención de las autoras originales del texto.

Other important Git configuration will automatically be configured when building the project (see Section 22.2 [Construcción desde Git], page 735). A `.git/hooks/commit-msg` hook will be installed that embeds '`Change-Id`' Git *trailers* in your commit messages for traceability purposes. It is important to preserve these when editing your commit messages, particularly if a first version of your proposed changes was already submitted for review. If you have a `commit-msg` hook of your own you would like to use with Guix, you can place it under the `.git/hooks/commit-msg.d/` directory.

### 22.10.2 Envío de una serie de parches

#### Single Patches

The `git send-email` command is the best way to send both single patches and patch series (see [Multiple Patches], page 764) to the Guix mailing list. Sending patches as email attachments may make them difficult to review in some mail clients, and `git diff` does not store commit metadata.

> **Nota:** The `git send-email` command is provided by the `send-email` output of the `git` package, i.e. `git:send-email`.

The following command will create a patch email from the latest commit, open it in your *EDITOR* or *VISUAL* for editing, and send it to the Guix mailing list to be reviewed and merged. Assuming you have already configured Git according to See Section 22.10.1 [Configuring Git], page 762, you can simply use:

```
$ git send-email --annotate -1
```

> **Tip:** To add a prefix to the subject of your patch, you may use the `--subject-prefix` option. The Guix project uses this to specify that the patch is intended for a branch or repository other than the `master` branch of `https://git.savannah.gnu.org/cgit/guix.git`.
>
> ```
> git send-email --annotate --subject-prefix='PATCH core-updates' -1
> ```

The patch email contains a three-dash separator line after the commit message. You may "annotate" the patch with explanatory text by adding it under this line. If you do not wish to annotate the email, you may drop the `--annotate` option.

If you need to send a revised patch, don't resend it like this or send a "fix" patch to be applied on top of the last one; instead, use `git commit --amend` or `git rebase` (`https://git-rebase.io`) to modify the commit, and use the *ISSUE_NUMBER*`@debbugs.gnu.org` address and the `-v` flag with `git send-email`.

```
$ git commit --amend
$ git send-email --annotate -vREVISION \
      --to=ISSUE_NUMBER@debbugs.gnu.org -1
```

> **Nota:** Due to an apparent bug in `git send-email`, `-v` *REVISION* (with the space) will not work; you *must* use `-v`*REVISION*.

You can find out *ISSUE_NUMBER* either by searching on the mumi interface at `https://issues.guix.gnu.org` for the name of your patch or reading the acknowledgement email sent automatically by Debbugs in reply to incoming bugs and patches, which contains the bug number.

### Notifying Teams

If your git checkout has been correctly configured (see Section 22.10.1 [Configuring Git], page 762), the `git send-email` command will automatically notify the appropriate team members, based on the scope of your changes. This relies on the `etc/teams.scm` script, which can also be invoked manually if you do not use the preferred `git send-email` command to submit patches. To list the available actions of the script, you can invoke it via the `etc/teams.scm help` command. For more information regarding teams, see Section 22.10.3 [Teams], page 764.

> **Nota:** On foreign distros, you might have to use `./pre-inst-env git send-email` for `etc/teams.scm` to work.

### Multiple Patches

While `git send-email` alone will suffice for a single patch, an unfortunate flaw in Debbugs means you need to be more careful when sending multiple patches: if you send them all to the `guix-patches@gnu.org` address, a new issue will be created for each patch!

When sending a series of patches, it's best to send a Git "cover letter" first, to give reviewers an overview of the patch series. We can create a directory called `outgoing` containing both our patch series and a cover letter called `0000-cover-letter.patch` with `git format-patch`.

```
$ git format-patch -NUMBER_COMMITS -o outgoing \
      --cover-letter --base=auto
```

We can now send *just* the cover letter to the `guix-patches@gnu.org` address, which will create an issue that we can send the rest of the patches to.

```
$ git send-email outgoing/0000-cover-letter.patch --annotate
$ rm outgoing/0000-cover-letter.patch # we don't want to resend it!
```

Ensure you edit the email to add an appropriate subject line and blurb before sending it. Note the automatically generated shortlog and diffstat below the blurb.

Once the Debbugs mailer has replied to your cover letter email, you can send the actual patches to the newly-created issue address.

```
$ git send-email outgoing/*.patch --to=ISSUE_NUMBER@debbugs.gnu.org
$ rm -rf outgoing # we don't need these anymore
```

Thankfully, this `git format-patch` dance is not necessary to send an amended patch series, since an issue already exists for the patchset.

```
$ git send-email -NUMBER_COMMITS -vREVISION \
      --to=ISSUE_NUMBER@debbugs.gnu.org
```

If need be, you may use `--cover-letter --annotate` to send another cover letter, e.g. for explaining what's changed since the last revision, and these changes are necessary.

## 22.10.3 Teams

There are several teams mentoring different parts of the Guix source code. To list all those teams, you can run from a Guix checkout:

```
$ ./etc/teams.scm list-teams
id: mentors
```

```
name: Mentors
description: A group of mentors who chaperone contributions by newcomers.
members:
+ Christopher Baines <mail@cbaines.net>
+ Ricardo Wurmus <rekado@elephly.net>
+ Mathieu Othacehe <othacehe@gnu.org>
+ jgart <jgart@dismail.de>
+ Ludovic Courtès <ludo@gnu.org>
...
```

You can run the following command to have the `Mentors` team put in CC of a patch series:

```
$ git send-email --to=ISSUE_NUMBER@debbugs.gnu.org \
  --header-cmd='etc/teams.scm cc-mentors-header-cmd' *.patch
```

The appropriate team or teams can also be inferred from the modified files. For instance, if you want to send the two latest commits of the current Git repository to review, you can run:

```
$ guix shell -D guix
[env]$ git send-email --to=ISSUE_NUMBER@debbugs.gnu.org -2
```

## 22.11 Tracking Bugs and Changes

This section describes how the Guix project tracks its bug reports, patch submissions and topic branches.

### 22.11.1 The Issue Tracker

El seguimiento de los informes de errores y los envíos de parches se realiza con una instancia de Debbugs en `https://bugs.gnu.org`. Los informes de errores se abren para el "paquete" `guix` (en la jerga de Debbugs), enviando un correo a `bug-guix@gnu.org`, mientras que para los envíos de parches se usa el paquete `guix-patches` enviando un correo a `guix-patches@gnu.org` (see Section 22.10 [Envío de parches], page 760).

### 22.11.2 Managing Patches and Branches

Changes should be posted to `guix-patches@gnu.org`. This mailing list fills the patch-tracking database (see Section 22.11.1 [The Issue Tracker], page 765). It also allows patches to be picked up and tested by the quality assurance tooling; the result of that testing eventually shows up on the dashboard at '`https://qa.guix.gnu.org/issue/`*ISSUE_NUMBER*', where *ISSUE_NUMBER* is the number assigned by the issue tracker. Leave time for a review, without committing anything.

As an exception, some changes considered "trivial" or "obvious" may be pushed directly to the `master` branch. This includes changes to fix typos and reverting commits that caused immediate problems. This is subject to being adjusted, allowing individuals to commit directly on non-controversial changes on parts they're familiar with.

Changes which affect more than 300 dependent packages (see Section 9.6 [Invocación de guix refresh], page 207) should first be pushed to a topic branch other than `master`; the set of changes should be consistent—e.g., "GNOME update", "NumPy update", etc. This allows for testing: the branch will automatically show up at

‘`https://qa.guix.gnu.org/branch/branch`’, with an indication of its build status on various platforms.

To help coordinate the merging of branches, you must create a new guix-patches issue each time you create a branch (see Section 22.11.1 [The Issue Tracker], page 765). The title of the issue requesting to merge a branch should have the following format:

```
Request for merging "name" branch
```

The QA infrastructure (`https://qa.guix.gnu.org/`) recognizes such issues and lists the merge requests on its main page. The following points apply to managing these branches:

1. The commits on the branch should be a combination of the patches relevant to the branch. Patches not related to the topic of the branch should go elsewhere.
2. Any changes that can be made on the master branch, should be made on the master branch. If a commit can be split to apply part of the changes on master, this is good to do.
3. It should be possible to re-create the branch by starting from master and applying the relevant patches.
4. Avoid merging master in to the branch. Prefer rebasing or re-creating the branch on top of an updated master revision.
5. Minimise the changes on master that are missing on the branch prior to merging the branch in to master. This means that the state of the branch better reflects the state of master should the branch be merged.
6. If you don’t have commit access, create the “Request for merging” issue and request that someone creates the branch. Include a list of issues/patches to include on the branch.

Normally branches will be merged in a “first come, first merged” manner, tracked through the guix-patches issues. If you agree on a different order with those involved, you can track this by updating which issues block[6] which other issues. Therefore, to know which branch is at the front of the queue, look for the oldest issue, or the issue that isn’t *blocked* by any other branch merges. An ordered list of branches with the open issues is available at `https://qa.guix.gnu.org`.

Once a branch is at the front of the queue, wait until sufficient time has passed for the build farms to have processed the changes, and for the necessary testing to have happened. For example, you can check ‘`https://qa.guix.gnu.org/branch/branch`’ to see information on some builds and substitute availability.

Once the branch has been merged, the issue should be closed and the branch deleted.

## 22.11.3 Debbugs User Interfaces

### 22.11.3.1 Web interface

Hay disponible una interfaz web (¡en realidad *dos* interfaces web!) para la navegación por las incidencias:

[6] You can mark an issue as blocked by another by emailing `control@debbugs.gnu.org` with the following line in the body of the email: `block XXXXX by YYYYY`. Where `XXXXX` is the number for the blocked issue, and `YYYYY` is the number for the issue blocking it.

- `https://issues.guix.gnu.org` provides a pleasant interface[7] to browse bug reports and patches, and to participate in discussions;
- `https://bugs.gnu.org/guix` muestra informes de errores;
- `https://bugs.gnu.org/guix-patches` muestra parches enviados.

Para ver los hilos relacionados con la incidencia número *n*, visite '`https://issues.guix.gnu.org/n`' o '`https://bugs.gnu.org/n`'.

### 22.11.3.2 Command-line interface

Mumi also comes with a command-line interface that can be used to search existing issues, open new issues and send patches. You do not need to use Emacs to use the mumi command-line client. You interact with it only on the command-line.

To use the mumi command-line interface, navigate to a local clone of the Guix git repository, and drop into a shell with mumi, git and git:send-email installed.

```
$ cd guix
~/guix$ guix shell mumi git git:send-email
```

To search for issues, say all open issues about "zig", run

```
~/guix [env]$ mumi search zig is:open

#60889 Add zig-build-system
opened on 17 Jan 17:37 Z by Ekaitz Zarraga
#61036 [PATCH 0/3] Update zig to 0.10.1
opened on 24 Jan 09:42 Z by Efraim Flashner
#39136 [PATCH] gnu: services: Add endlessh.
opened on 14 Jan 2020 21:21 by Nicol? Balzarotti
#60424 [PATCH] gnu: Add python-online-judge-tools
opened on 30 Dec 2022 07:03 by gemmaro
#45601 [PATCH 0/6] vlang 0.2 update
opened on  1 Jan 2021 19:23 by Ryan Prior
```

Pick an issue and make it the "current" issue.

```
~/guix [env]$ mumi current 61036

#61036 [PATCH 0/3] Update zig to 0.10.1
opened on 24 Jan 09:42 Z by Efraim Flashner
```

Once an issue is the current issue, you can easily create and send patches to it using

```
~/guix [env]$ git format-patch origin/master
~/guix [env]$ mumi send-email foo.patch bar.patch
```

Note that you do not have to pass in '`--to`' or '`--cc`' arguments to `git format-patch`. `mumi send-email` will put them in correctly when sending the patches.

To open a new issue, run

```
~/guix [env]$ mumi new
```

[7] The web interface at `https://issues.guix.gnu.org` is powered by Mumi, a nice piece of software written in Guile, and you can help! See `https://git.savannah.gnu.org/cgit/guix/mumi.git`.

and send patches

```
~/guix [env]$ mumi send-email foo.patch bar.patch
```

`mumi send-email` is really a wrapper around `git send-email` that automates away all the nitty-gritty of sending patches. It uses the current issue state to automatically figure out the correct '`To`' address to send to, other participants to '`Cc`', headers to add, etc.

Also note that, unlike `git send-email`, `mumi send-email` works perfectly well with single and multiple patches alike. It automates away the debbugs dance of sending the first patch, waiting for a response from debbugs and sending the remaining patches. It does so by sending the first patch, polling the server for a response, and then sending the remaining patches. This polling can unfortunately take a few minutes. So, please be patient.

#### 22.11.3.3 Emacs interface

Si usa Emacs, puede encontrar más conveniente la interacción con las incidencias mediante `debbugs.el`, que puede instalar con:

```
guix install emacs-debbugs
```

Por ejemplo, para enumerar todos las incidencias abiertas en `guix-patches` pulse:

*`C-u M-x`* `debbugs-gnu` *`RET RET`* `guix-patches` *`RET`* `n y`

For a more convenient (shorter) way to access both the bugs and patches submissions, you may want to configure the `debbugs-gnu-default-packages` and `debbugs-gnu-default-severities` Emacs variables (see Section 22.5.1 [Viewing Bugs within Emacs], page 741).

To search for bugs, '*`M-x`* `debbugs-gnu-guix-search`' can be used.

See *Debbugs User Guide*, para más información sobre esta útil herramienta.

### 22.11.4 Debbugs Usertags

Debbugs provides a feature called *usertags* that allows any user to tag any bug with an arbitrary label. Bugs can be searched by usertag, so this is a handy way to organize bugs[8]. If you use Emacs Debbugs, the entry-point to consult existing usertags is the '`C-u M-x debbugs-gnu-usertags`' procedure. To set a usertag, press '`C`' while consulting a bug within the *Guix-Patches* buffer opened with '`C-u M-x debbugs-gnu-bugs`' buffer, then select `usertag` and follow the instructions.

For example, to view all the bug reports (or patches, in the case of `guix-patches`) tagged with the usertag `powerpc64le-linux` for the user `guix`, open a URL like the following in a web browser: `https://debbugs.gnu.org/cgi-bin/pkgreport.cgi?tag=powerpc64le-linux;users=guix`.

For more information on how to use usertags, please refer to the documentation for Debbugs or the documentation for whatever tool you use to interact with Debbugs.

In Guix, we are experimenting with usertags to keep track of architecture-specific issues, as well as reviewed ones. To facilitate collaboration, all our usertags are associated with the single user `guix`. The following usertags currently exist for that user:

`powerpc64le-linux`
: The purpose of this usertag is to make it easy to find the issues that matter most for the `powerpc64le-linux` system type. Please assign this usertag to

[8] The list of usertags is public information, and anyone can modify any user's list of usertags, so keep that in mind if you choose to use this feature.

bugs or patches that affect `powerpc64le-linux` but not other system types. In addition, you may use it to identify issues that for some reason are particularly important for the `powerpc64le-linux` system type, even if the issue affects other system types, too.

`reproducibilidad`
: For issues related to reproducibility. For example, it would be appropriate to assign this usertag to a bug report for a package that fails to build reproducibly.

`reviewed-looks-good`
: You have reviewed the series and it looks good to you (LGTM).

If you're a committer and you want to add a usertag, just start using it with the `guix` user. If the usertag proves useful to you, consider updating this section of the manual so that others will know what your usertag means.

### 22.11.5 Cuirass Build Notifications

Cuirass includes RSS (Really Simple Syndication) feeds as one of its features (see Section "Notifications" in `cuirass`). Since Berlin (`https://ci.guix.gnu.org/`) runs an instance of Cuirass, this feature can be used to keep track of recently broken or fixed packages caused by changes pushed to the Guix git repository. Any RSS client can be used. A good one, included with Emacs, is See Section "Gnus" in `gnus`. To register the feed, copy its URL, then from the main Gnus buffer, '`*Group*`', do the following:

```
G R https://ci.guix.gnu.org/events/rss/?specification=master RET
Guix CI - master RET Build events for specification master. RET
```

Then, back at the '`*Group*`' buffer, press `s` to save the newly added RSS group. As for any other Gnus group, you can update its content by pressing the `g` key. You should now receive notifications that read like:

```
. [ ?: Cuirass ] Build tree-sitter-meson.aarch64-linux on master is fixed.
. [ ?: Cuirass ] Build rust-pbkdf2.aarch64-linux on master is fixed.
. [ ?: Cuirass ] Build rust-pbkdf2.x86_64-linux on master is fixed.
```

where each RSS entry contains a link to the Cuirass build details page of the associated build.

## 22.12 Acceso al repositorio

Everyone can contribute to Guix without having commit access (see Section 22.10 [Envío de parches], page 760). However, for frequent contributors, having write access to the repository can be convenient. As a rule of thumb, a contributor should have accumulated fifty (50) reviewed commits to be considered as a committer and have sustained their activity in the project for at least 6 months. This ensures enough interactions with the contributor, which is essential for mentoring and assessing whether they are ready to become a committer. Commit access should not be thought of as a "badge of honor" but rather as a responsibility a contributor is willing to take to help the project. It is expected from all contributors, and even more so from committers, to help build consensus and make decisions based on consensus. By using consensus, we are committed to finding solutions that everyone can live with. It implies that no decision is made against significant concerns and these concerns are actively resolved with proposals that work for everyone. A contributor

(which may or may not have commit access) wishing to block a proposal bears a special responsibility for finding alternatives, proposing ideas/code or explain the rationale for the status quo to resolve the deadlock. To learn what consensus decision making means and understand its finer details, you are encouraged to read `https://www.seedsforchange.org.uk/consensus`.

The following sections explain how to get commit access, how to be ready to push commits, and the policies and community expectations for commits pushed upstream.

### 22.12.1 Applying for Commit Access

When you deem it necessary, consider applying for commit access by following these steps:

1. Encuentre tres personas que contribuyan al proyecto que puedan respaldarle. Puede ver la lista de personas que contribuyen en `https://savannah.gnu.org/project/memberlist.php?group=guix`. Cada una de ellas deberá enviar un correo confirmando el respaldo a `guix-maintainers@gnu.org` (un alias privado para el colectivo de personas que mantienen el proyecto), firmado con su clave OpenPGP.

   Se espera que dichas personas hayan tenido algunas interacciones con usted en sus contribuciones y sean capaces de juzgar si es suficientemente familiar con las prácticas del proyecto. *No* es un juicio sobre el valor de su trabajo, por lo que un rechazo debe ser interpretado más bien como un "habrá que probar de nuevo más adelante".

2. Envíe un correo a `guix-maintainers@gnu.org` expresando su intención, enumerando a las tres contribuidoras que respaldan su petición, firmado con su clave OpenPGP que usará para firmar las revisiones, y proporcionando su huella dactilar (véase a continuación). Véase `https://emailselfdefense.fsf.org/es/` para una introducción a la criptografía de clave pública con GnuPG.

   Configure GnuPG de modo que no use el algorítmo de hash SHA1 nunca para las firmas digitales, el cual se sabe que no es seguro desde 2019, añadiendo, por ejemplo, la siguiente línea en `~/.gnupg/gpg.conf` (see Section "GPG Esoteric Options" in *The GNU Privacy Guard Manual*):

   ```
   digest-algo sha512
   ```

3. Las personas que mantienen el proyecto decidirán en última instancia si conceder o no el acceso de escritura, habitualmente siguiendo las recomendaciones de las personas de referencia proporcionadas.

4. Una vez haya conseguido acceso, en caso de hacerlo, por favor envíe un mensaje a `guix-devel@gnu.org` para notificarlo, de nuevo firmado con la clave OpenPGP que vaya a usar para firmar las revisiones (hágalo antes de subir su primera revisión). De esta manera todo el mundo puede enterarse y asegurarse de que controla su clave OpenPGP.

   > **Importante:** Antes de que suba alguna revisión por primera vez, quienes mantienen Guix deben:
   >
   > 1. añadir su clave OpenPGP a la rama `keyring`;
   > 2. añadir su firma OpenPGP al archivo `.guix-authorizations` de la(s) rama(s) a las que vaya a subir código.

5. Asegúrese de leer el resto de esta sección y... ¡a disfrutar!

> **Nota:** Quienes mantienen el proyecto están encantadas de proporcionar acceso al repositorio a personas que han contribuido durante algún tiempo y tienen buen registro—¡no sea tímida y no subestime su trabajo!
>
> No obstante, tenga en cuenta que el proyecto está trabajando hacia la automatización de la revisión de parches y el sistema de mezclas, lo que, como consecuencia, puede hacer necesario que menos gente tenga acceso de escritura al repositorio principal. ¡Seguiremos informando!

Todas las revisiones que se suban al repositorio central de Savannah deben estar firmadas por una clave OpenPGP, y la clave pública debe subirse a su cuenta de usuaria en Savannah y a servidores públicos de claves, como `keys.openpgp.org`. Para configurar que Git firme automáticamente las revisiones ejecute:

```
git config commit.gpgsign true

# Substitute the fingerprint of your public PGP key.
git config user.signingkey CABBA6EA1DC0FF33
```

To check that commits are signed with correct key, use:

```
make authenticate
```

To avoid accidentally pushing unsigned or signed with the wrong key commits to Savannah, make sure to configure Git according to See Section 22.10.1 [Configuring Git], page 762.

### 22.12.2 Commit Policy

Si obtiene acceso, por favor asegúrese de seguir la política descrita a continuación (el debate sobre dicha política puede llevarse a cabo en `guix-devel@gnu.org`).

Ensure you're aware of how the changes should be handled (see Section 22.11.2 [Managing Patches and Branches], page 765) prior to being pushed to the repository, especially for the `master` branch.

If you're committing and pushing your own changes, try and wait at least one week (two weeks for more significant changes) after you send them for review. After this, if no one else is available to review them and if you're confident about the changes, it's OK to commit.

Cuando suba un commit en nombre de alguien, por favor añada una línea de `Signed-off-by` al final del mensaje de la revisión—por ejemplo con `git am --signoff`. Esto mejora el seguimiento sobre quién hizo qué.

Cuando añada entradas de noticias del canal (see Chapter 6 [Canales], page 69), compruebe que tienen el formato correcto con la siguiente órden antes de subir los cambios al repositorio:

```
make check-channel-news
```

### 22.12.3 Addressing Issues

Peer review (see Section 22.10 [Envío de parches], page 760) and tools such as `guix lint` (see Section 9.8 [Invocación de guix lint], page 216) and the test suite (see Section 22.3 [Ejecución de la batería de pruebas], page 737) should catch issues before they are pushed. Yet, commits that "break" functionality might occasionally go through. When that happens, there are two priorities: mitigating the impact, and understanding what happened to reduce the

chance of similar incidents in the future. The responsibility for both these things primarily lies with those involved, but like everything this is a group effort.

Some issues can directly affect all users—for instance because they make `guix pull` fail or break core functionality, because they break major packages (at build time or run time), or because they introduce known security vulnerabilities.

The people involved in authoring, reviewing, and pushing such commit(s) should be at the forefront to mitigate their impact in a timely fashion: by pushing a followup commit to fix it (if possible), or by reverting it to leave time to come up with a proper fix, and by communicating with other developers about the problem.

If these persons are unavailable to address the issue in time, other committers are entitled to revert the commit(s), explaining in the commit log and on the mailing list what the problem was, with the goal of leaving time to the original committer, reviewer(s), and author(s) to propose a way forward.

Once the problem has been dealt with, it is the responsibility of those involved to make sure the situation is understood. If you are working to understand what happened, focus on gathering information and avoid assigning any blame. Do ask those involved to describe what happened, do not ask them to explain the situation—this would implicitly blame them, which is unhelpful. Accountability comes from a consensus about the problem, learning from it and improving processes so that it's less likely to reoccur.

### 22.12.4 Commit Revocation

In order to reduce the possibility of mistakes, committers will have their Savannah account removed from the Guix Savannah project and their key removed from `.guix-authorizations` after 12 months of inactivity; they can ask to regain commit access by emailing the maintainers, without going through the vouching process.

Maintainers[9] may also revoke an individual's commit rights, as a last resort, if cooperation with the rest of the community has caused too much friction—even within the bounds of the project's code of conduct (see Chapter 22 [Contribuir], page 734). They would only do so after public or private discussion with the individual and a clear notice. Examples of behavior that hinders cooperation and could lead to such a decision include:

- repeated violation of the commit policy stated above;
- repeated failure to take peer criticism into account;
- breaching trust through a series of grave incidents.

When maintainers resort to such a decision, they notify developers on `guix-devel@gnu.org`; inquiries may be sent to `guix-maintainers@gnu.org`. Depending on the situation, the individual may still be welcome to contribute.

### 22.12.5 Helping Out

Una última cosa: el proyecto sigue adelante porque las contribuidoras no solo suben sus cambios, sino que también ofrecen su tiempo *revisando* y subiendo cambios de otras personas. Como contribuidora, también se agradece que use su experiencia y derechos de escritura en el repositorio para ayudar a otras personas que quieren contribuir.

[9] See `https://guix.gnu.org/en/about` for the current list of maintainers. You can email them privately at `guix-maintainers@gnu.org`.

## 22.13 Reviewing the Work of Others

Perhaps the biggest action you can do to help GNU Guix grow as a project is to review the work contributed by others. You do not need to be a committer to do so; applying, reading the source, building, linting and running other people's series and sharing your comments about your experience will give some confidence to committers. Basically, you must ensure the check list found in the Section 22.10 [Envío de parches], page 760, section has been correctly followed. A reviewed patch series should give the best chances for the proposed change to be merged faster, so if a change you would like to see merged hasn't yet been reviewed, this is the most appropriate thing to do!

Review comments should be unambiguous; be as clear and explicit as you can about what you think should be changed, ensuring the author can take action on it. Please try to keep the following guidelines in mind during review:

1. *Be clear and explicit about changes you are suggesting*, ensuring the author can take action on it. In particular, it is a good idea to explicitly ask for new revisions when you want it.
2. *Remain focused: do not change the scope of the work being reviewed.* For example, if the contribution touches code that follows a pattern deemed unwieldy, it would be unfair to ask the submitter to fix all occurrences of that pattern in the code; to put it simply, if a problem unrelated to the patch at hand was already there, do not ask the submitter to fix it.
3. *Ensure progress.* As they respond to review, submitters may submit new revisions of their changes; avoid requesting changes that you did not request in the previous round of comments. Overall, the submitter should get a clear sense of progress; the number of items open for discussion should clearly decrease over time.
4. *Aim for finalization.* Reviewing code is time-consuming. Your goal as a reviewer is to put the process on a clear path towards integration, possibly with agreed-upon changes, or rejection, with a clear and mutually-understood reasoning. Avoid leaving the review process in a lingering state with no clear way out.
5. *Review is a discussion.* The submitter's and reviewer's views on how to achieve a particular change may not always be aligned. To lead the discussion, remain focused, ensure progress and aim for finalization, spending time proportional to the stakes[10]. As a reviewer, try hard to explain the rationale for suggestions you make, and to understand and take into account the submitter's motivation for doing things in a certain way.

When you deem the proposed change adequate and ready for inclusion within Guix, the following well understood/codified '`Reviewed-by: Your Name<your-email@example.com>`'[11] line should be used to sign off as a reviewer, meaning you have reviewed the change and that it looks good to you:

[10] The tendency to discuss minute details at length is often referred to as "bikeshedding", where much time is spent discussing each one's preference for the color of the shed at the expense of progress made on the project to keep bikes dry.

[11] The '`Reviewed-by`' Git trailer is used by other projects such as Linux, and is understood by third-party tools such as the '`b4 am`' sub-command, which is able to retrieve the complete submission email thread from a public-inbox instance and add the Git trailers found in replies to the commit patches.

- If the *whole* series (containing multiple commits) looks good to you, reply with '`Reviewed-by: Your Name<your-email@example.com>`' to the cover page if it has one, or to the last patch of the series otherwise, adding another '`(for the whole series)`' comment on the line below to explicit this fact.
- If you instead want to mark a *single commit* as reviewed (but not the whole series), simply reply with '`Reviewed-by: Your Name<your-email@example.com>`' to that commit message.

If you are not a committer, you can help others find a *series* you have reviewed more easily by adding a `reviewed-looks-good` usertag for the `guix` user (see Section 22.11.4 [Debbugs Usertags], page 768).

## 22.14 Actualizar el paquete Guix

A veces es deseable actualizar el propio paquete `guix` (el paquete definido en `(gnu packages package-management)`, por ejemplo para poner a disposición del tipo de servicio `guix-service-type` nuevas características disponibles en el daemon. Para simplificar esta tarea se puede usar la siguiente orden:

```
make update-guix-package
```

El objetivo de make `update-guix-package` usa la última *revisión* (commit en inglés) de `HEAD` en su copia local de Guix, calcula el hash correspondiente a las fuentes de Guix en dicho commit y actualiza los campos `commit`, `revision` y el hash de la defínción del paquete `guix`.

Para validar que la actualización del hash del paquete `guix` es correcta y que se puede construir de manera satisfactoria se puede ejecutar la siguiente orden en el directorio de su copia de trabajo local de Guix:

```
./pre-inst-env guix build guix
```

Para prevenir de actualizaciones accidentales del paquete `guix` a una revisión a la que otras personas no puedan hacer referencia se comprueba que dicha revisión se haya publicado ya en el repositorio git de Guix alojado en Savannah.

This check can be disabled, *at your own peril*, by setting the `GUIX_ALLOW_ME_TO_USE_PRIVATE_COMMIT` environment variable. When this variable is set, the updated package source is also added to the store. This is used as part of the release process of Guix.

## 22.15 Writing Documentation

Guix is documented using the Texinfo system. If you are not yet familiar with it, we accept contributions for documentation in most formats. That includes plain text, Markdown, Org, etc.

Documentation contributions can be sent to `guix-patches@gnu.org`. Prepend '`[DOCUMENTATION]`' to the subject.

When you need to make more than a simple addition to the documentation, we prefer that you send a proper patch as opposed to sending an email as described above. See Section 22.10 [Envío de parches], page 760, for more information on how to send your patches.

To modify the documentation, you need to edit `doc/guix.texi` and `doc/contributing.texi` (which contains this documentation section), or `doc/guix-cookbook.texi` for the cookbook. If you compiled the Guix repository before, you will have many more `.texi` files that are translations of these documents. Do not modify them, the translation is managed through Weblate (`https://translate.fedoraproject.org/projects/guix`). See Section 22.16 [Traduciendo Guix], page 775, for more information.

To render documentation, you must first make sure that you ran `./configure` in your source tree (see Section 22.4 [Ejecución de Guix antes de estar instalado], page 739). After that you can run one of the following commands:

- '`make doc/guix.info`' to compile the Info manual. You can check it with `info doc/guix.info`.
- '`make doc/guix.html`' to compile the HTML version. You can point your browser to the relevant file in the `doc/guix.html` directory.
- '`make doc/guix-cookbook.info`' for the cookbook Info manual.
- '`make doc/guix-cookbook.html`' for the cookbook HTML version.

## 22.16 Traduciendo Guix

Writing code and packages is not the only way to provide a meaningful contribution to Guix. Translating to a language you speak is another example of a valuable contribution you can make. This section is designed to describe the translation process. It gives you advice on how you can get involved, what can be translated, what mistakes you should avoid and what we can do to help you!

Guix is a big project that has multiple components that can be translated. We coordinate the translation effort on a Weblate instance (`https://translate.fedoraproject.org/projects/guix/`) hosted by our friends at Fedora. You will need an account to submit translations.

Some of the software packaged in Guix also contain translations. We do not host a translation platform for them. If you want to translate a package provided by Guix, you should contact their developers or find the information on their website. As an example, you can find the homepage of the `hello` package by typing `guix show hello`. On the "homepage" line, you will see `https://www.gnu.org/software/hello/` as the homepage.

Many GNU and non-GNU packages can be translated on the Translation Project (`https://translationproject.org`). Some projects with multiple components have their own platform. For instance, GNOME has its own platform, Damned Lies (`https://l10n.gnome.org/`).

Guix has five components hosted on Weblate.

- `guix` contains all the strings from the Guix software (the guided system installer, the package manager, etc), excluding packages.
- `packages` contains the synopsis (single-sentence description of a package) and description (longer description) of packages in Guix.
- `website` contains the official Guix website, except for blog posts and multimedia content.
- `documentation-manual` corresponds to this manual.
- `documentation-cookbook` is the component for the cookbook.

## General Directions

Once you get an account, you should be able to select a component from the guix project (`https://translate.fedoraproject.org/projects/guix/`), and select a language. If your language does not appear in the list, go to the bottom and click on the "Start new translation" button. Select the language you want to translate to from the list, to start your new translation.

Like lots of other free software packages, Guix uses GNU Gettext (`https://www.gnu.org/software/gettext`) for its translations, with which translatable strings are extracted from the source code to so-called PO files.

Even though PO files are text files, changes should not be made with a text editor but with PO editing software. Weblate integrates PO editing functionality. Alternatively, translators can use any of various free-software tools for filling in translations, of which Poedit (`https://poedit.net/`) is one example, and (after logging in) upload (`https://docs.weblate.org/en/latest/user/files.html`) the changed file. There is also a special PO editing mode (`https://www.emacswiki.org/emacs/PoMode`) for users of GNU Emacs. Over time translators find out what software they are happy with and what features they need.

On Weblate, you will find various links to the editor, that will show various subsets (or all) of the strings. Have a look around and at the documentation (`https://docs.weblate.org/en/latest/`) to familiarize yourself with the platform.

## Translation Components

In this section, we provide more detailed guidance on the translation process, as well as details on what you should or should not do. When in doubt, please contact us, we will be happy to help!

guix

Guix is written in the Guile programming language, and some strings contain special formatting that is interpreted by Guile. These special formatting should be highlighted by Weblate. They start with `~` followed by one or more characters.

When printing the string, Guile replaces the special formatting symbols with actual values. For instance, the string '`ambiguous package specification `~a'`' would be substituted to contain said package specification instead of `~a`. To properly translate this string, you must keep the formatting code in your translation, although you can place it where it makes sense in your language. For instance, the French translation says '`spécification du paquet « ~a » ambiguë`' because the adjective needs to be placed in the end of the sentence.

If there are multiple formatting symbols, make sure to respect the order. Guile does not know in which order you intended the string to be read, so it will substitute the symbols in the same order as the English sentence.

As an example, you cannot translate '`package '~a' has been superseded by '~a'`' by '`'~a' superseeds package '~a'`', because the meaning would be reversed. If *foo* is superseded by *bar*, the translation would read '`'foo' superseeds package 'bar'`'. To work around this problem, it is possible to use more advanced formatting to select a given piece of data, instead of

following the default English order. See Section "Formatted Output" in *GNU Guile Reference Manual*, for more information on formatting in Guile.

paquetes

Package descriptions occasionally contain Texinfo markup (see Section 22.8.4 [Sinopsis y descripciones], page 751). Texinfo markup looks like '`@code{rm -rf}`', '`@emph{important}`', etc. When translating, please leave markup as is.

The characters after "@" form the name of the markup, and the text between "{" and "}" is its content. In general, you should not translate the content of markup like `@code`, as it contains literal code that do not change with language. You can translate the content of formatting markup such as `@emph`, `@i`, `@itemize`, `@item`. However, do not translate the name of the markup, or it will not be recognized. Do not translate the word after `@end`, it is the name of the markup that is closed at this position (e.g. `@itemize ... @end itemize`).

documentation-manual and documentation-cookbook

The first step to ensure a successful translation of the manual is to find and translate the following strings *first*:

- `version.texi`: Translate this string as `version-xx.texi`, where `xx` is your language code (the one shown in the URL on weblate).
- `contributing.texi`: Translate this string as `contributing.xx.texi`, where `xx` is the same language code.
- `Top`: Do not translate this string, it is important for Texinfo. If you translate it, the document will be empty (missing a Top node). Please look for it, and register `Top` as its translation.

Translating these strings first ensure we can include your translation in the guix repository without breaking the make process or the `guix pull` machinery.

The manual and the cookbook both use Texinfo. As for `packages`, please keep Texinfo markup as is. There are more possible markup types in the manual than in the package descriptions. In general, do not translate the content of `@code`, `@file`, `@var`, `@value`, etc. You should translate the content of formatting markup such as `@emph`, `@i`, etc.

The manual contains sections that can be referred to by name by `@ref`, `@xref` and `@pxref`. We have a mechanism in place so you do not have to translate their content. If you keep the English title, we will automatically replace it with your translation of that title. This ensures that Texinfo will always be able to find the node. If you decide to change the translation of the title, the references will automatically be updated and you will not have to update them all yourself.

When translating references from the cookbook to the manual, you need to replace the name of the manual and the name of the section. For instance, to translate `@pxref{Defining Packages,,, guix, GNU Guix Reference Manual}`, you would replace `Defining Packages` with the title of that section in the translated manual *only* if that title is translated. If the title is not translated in your language yet, do not translate it here, or the link will be broken. Replace `guix` with `guix.xx` where `xx` is your language code. `GNU Guix`

`Reference Manual` is the text of the link. You can translate it however you wish.

website

The website pages are written using SXML, an s-expression version of HTML, the basic language of the web. We have a process to extract translatable strings from the source, and replace complex s-expressions with a more familiar XML markup, where each markup is numbered. Translators can arbitrarily change the ordering, as in the following example.

```
#. TRANSLATORS: Defining Packages is a section name
#. in the English (en) manual.
#: apps/base/templates/about.scm:64
msgid "Packages are <1>defined<1.1>en</1.1><1.2>Defining-Packages.html</1.2>
msgstr "Pakete werden als reine <2>Guile</2>-Module <1>definiert<1.1>de</1.1
```

Note that you need to include the same markups. You cannot skip any.

In case you make a mistake, the component might fail to build properly with your language, or even make guix pull fail. To prevent that, we have a process in place to check the content of the files before pushing to our repository. We will not be able to update the translation for your language in Guix, so we will notify you (through weblate and/or by email) so you get a chance to fix the issue.

## Outside of Weblate

Currently, some parts of Guix cannot be translated on Weblate, help wanted!

- `guix pull` news can be translated in `news.scm`, but is not available from Weblate. If you want to provide a translation, you can prepare a patch as described above, or simply send us your translation with the name of the news entry you translated and your language. See Section 6.12 [Escribir de noticias del canal], page 77, for more information about channel news.
- Guix blog posts cannot currently be translated.
- The installer script (for foreign distributions) is entirely in English.
- Some of the libraries Guix uses cannot be translated or are translated outside of the Guix project. Guile itself is not internationalized.
- Other manuals linked from this manual or the cookbook might not be translated.

## Conditions for Inclusion

There are no conditions for adding new translations of the `guix` and `guix-packages` components, other than they need at least one translated string. New languages will be added to Guix as soon as possible. The files may be removed if they fall out of sync and have no more translated strings.

Given that the web site is dedicated to new users, we want its translation to be as complete as possible before we include it in the language menu. For a new language to be included, it needs to reach at least 80% completion. When a language is included, it may be removed in the future if it stays out of sync and falls below 60% completion.

The manual and cookbook are automatically added in the default compilation target. Every time we synchronize translations, developers need to recompile all the translated

manuals and cookbooks. This is useless for what is essentially the English manual or cookbook. Therefore, we will only include a new language when it reaches 10% completion in the component. When a language is included, it may be removed in the future if it stays out of sync and falls below 5% completion.

### Translation Infrastructure

Weblate is backed by a git repository from which it discovers new strings to translate and pushes new and updated translations. Normally, it would be enough to give it commit access to our repositories. However, we decided to use a separate repository for two reasons. First, we would have to give Weblate commit access and authorize its signing key, but we do not trust it in the same way we trust guix developers, especially since we do not manage the instance ourselves. Second, if translators mess something up, it can break the generation of the website and/or guix pull for all our users, independently of their language.

For these reasons, we use a dedicated repository to host translations, and we synchronize it with our guix and artworks repositories after checking no issue was introduced in the translation.

Developers can download the latest PO files from weblate in the Guix repository by running the `make download-po` command. It will automatically download the latest files from weblate, reformat them to a canonical form, and check they do not contain issues. The manual needs to be built again to check for additional issues that might crash Texinfo.

Before pushing new translation files, developers should add them to the make machinery so the translations are actually available. The process differs for the various components.

- New po files for the `guix` and `packages` components must be registered by adding the new language to `po/guix/LINGUAS` or `po/packages/LINGUAS`.
- New po files for the `documentation-manual` component must be registered by adding the file name to `DOC_PO_FILES` in `po/doc/local.mk`, the generated `%D%/guix.xx.texi` manual to `info_TEXINFOS` in `doc/local.mk` and the generated `%D%/guix.xx.texi` and `%D%/contributing.xx.texi` to `TRANSLATED_INFO` also in `doc/local.mk`.
- New po files for the `documentation-cookbook` component must be registered by adding the file name to `DOC_COOKBOOK_PO_FILES` in `po/doc/local.mk`, the generated `%D%/guix-cookbook.xx.texi` manual to `info_TEXINFOS` in `doc/local.mk` and the generated `%D%/guix-cookbook.xx.texi` to `TRANSLATED_INFO` also in `doc/local.mk`.
- New po files for the `website` component must be added to the `guix-artwork` repository, in `website/po/`. `website/po/LINGUAS` and `website/po/ietf-tags.scm` must be updated accordingly (see `website/i18n-howto.txt` for more information on the process).

# 23 Reconocimientos

Guix está basado en el gestor de paquetes Nix (`https://nixos.org/nix/`), que fue diseñado e implementado por Eelco Dolstra, con contribuciones de otra gente (véase el archivo `nix/AUTHORS` en Guix). Nix fue pionero en la gestión de paquetes funcional, y promovió características sin precedentes, como las actualizaciones y reversiones de paquetes transaccionales, perfiles por usuaria y un proceso de compilación referencialmente transparente. Sin este trabajo, Guix no existiría.

Las distribuciones de software basadas en Nix, Nixpkgs y NixOS, también han sido una inspiración para Guix.

GNU Guix en sí es un trabajo colectivo con contribuciones de un número de gente. Mire el archivo `AUTHORS` en Guix para más información sobre esa gente maja. El archivo `THANKS` enumera personas que han ayudado informando de errores, se han encargado de infraestructura, han proporcionando arte y temas, han realizado sugerencias, y más—¡gracias!

# Appendix A Licencia de documentación libre GNU

Version 1.3, 3 November 2008



0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# Índice de conceptos

## C



## D

## E

## F



## G

## H



## I



## J



## K



## L

## M



## N



## O



## P

# Q



# R

## S

## T



## U



## V



## W



## X

## Z

# Índice programático

## H



## I



## K



## L



## M



## N



## O



## P

## Q



## R



## S



## T



## U



## V



## W



## X