Referenzhandbuch zu GNU Guix

Den funktionalen Paketmanager GNU Guix benutzen

Die Entwickler von GNU Guix

Edition 1.4.0
8 June 2023

Inhaltsverzeichnis

1 Einführung

GNU Guix[1] ist ein Werkzeug zur Verwaltung von Softwarepaketen für das GNU-System und eine Distribution (eine „Verteilung“) desselbigen GNU-Systems. Guix macht es *nicht* mit besonderen Berechtigungen ausgestatteten, „unprivilegierten“ Nutzern leicht, Softwarepakete zu installieren, zu aktualisieren oder zu entfernen, zu einem vorherigen Satz von Paketen zurückzuwechseln, Pakete aus ihrem Quellcode heraus zu erstellen und hilft allgemein bei der Erzeugung und Wartung von Software-Umgebungen.

Sie können GNU Guix auf ein bestehendes GNU/Linux-System aufsetzen, wo es die bereits verfügbaren Werkzeuge ergänzt, ohne zu stören (siehe Kapitel 2 [Installation], Seite 5), oder Sie können es als eine eigenständige Betriebssystem-Distribution namens *Guix System* verwenden[2]. Siehe Abschnitt 1.2 [GNU-Distribution], Seite 2.

1.1 Auf Guix-Art Software verwalten

Guix bietet eine befehlszeilenbasierte Paketverwaltungsschnittstelle (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45), Werkzeuge als Hilfestellung bei der Software-Entwicklung (siehe Kapitel 8 [Entwicklung], Seite 86), Befehlszeilenwerkzeuge für fortgeschrittenere Nutzung (siehe Kapitel 10 [Zubehör], Seite 189) sowie Schnittstellen zur Programmierung in Scheme (siehe Kapitel 9 [Programmierschnittstelle], Seite 108).

Der *Erstellungs-Daemon* ist für das Erstellen von Paketen im Auftrag von Nutzern verantwortlich (siehe Abschnitt 2.4 [Den Daemon einrichten], Seite 11) und für das Herunterladen vorerstellter Binärdateien aus autorisierten Quellen (siehe Abschnitt 6.3 [Substitute], Seite 56).

Guix enthält Paketdefinitionen für viele Pakete, manche aus GNU und andere nicht aus GNU, die alle die Freiheit des Computernutzers respektieren (`https://www.gnu.org/philosophy/free-sw.html`). Es ist *erweiterbar*: Nutzer können ihre eigenen Paketdefinitionen schreiben (siehe Abschnitt 9.2 [Pakete definieren], Seite 109) und sie als unabhängige Paketmodule verfügbar machen (siehe Abschnitt 9.1 [Paketmodule], Seite 108). Es ist auch *anpassbar*: Nutzer können spezialisierte Paketdefinitionen aus bestehenden *ableiten*, auch von der Befehlszeile (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192).

Intern implementiert Guix die Disziplin der *funktionalen Paketverwaltung*, zu der Nix schon die Pionierarbeit geleistet hat (siehe Kapitel 23 [Danksagungen], Seite 744). In Guix wird der Prozess, ein Paket zu erstellen und zu installieren, als eine *Funktion* im mathematischen Sinn aufgefasst. Diese Funktion hat Eingaben, wie zum Beispiel Erstellungs-Skripts, einen Compiler und Bibliotheken, und liefert ein installiertes Paket. Als eine reine Funktion hängt sein Ergebnis allein von seinen Eingaben ab – zum Beispiel kann er nicht auf Software oder Skripts Bezug nehmen, die nicht ausdrücklich als Eingaben übergeben wurden. Eine Erstellungsfunktion führt immer zum selben Ergebnis, wenn ihr die gleiche Menge an Eingaben übergeben wurde. Sie kann die Umgebung des laufenden Systems auf keine Weise

[1] „Guix“ wird wie „geeks“ ausgesprochen, also als „iks“ in der Notation des Internationalen Phonetischen Alphabets (IPA).

[2] Der Name *Guix System* wird auf englische Weise ausgesprochen. Früher hatten wir „Guix System“ als „Guix System Distribution“ bezeichnet und mit „GuixSD“ abgekürzt. Wir denken mittlerweile aber, dass es sinnvoller ist, alles unter der Fahne von Guix zu gruppieren, weil schließlich „Guix System“ auch über den Befehl `guix system` verfügbar ist, selbst wenn Sie Guix auf einer fremden Distribution benutzen!

beeinflussen, zum Beispiel kann sie keine Dateien außerhalb ihrer Erstellungs- und Installationsverzeichnisse verändern. Um dies zu erreichen, laufen Erstellungsprozesse in isolierten Umgebungen (sogenannte *Container*), wo nur ausdrückliche Eingaben sichtbar sind.

Das Ergebnis von Paketerstellungsfunktionen wird im Dateisystem *zwischengespeichert* in einem besonderen Verzeichnis, was als *der Store* bezeichnet wird (siehe Abschnitt 9.9 [Der Store], Seite 165). Jedes Paket wird in sein eigenes Verzeichnis im Store installiert – standardmäßig ist er unter `/gnu/store` zu finden. Der Verzeichnisname enthält einen Hash aller Eingaben, anhand derer das Paket erzeugt wurde, somit hat das Ändern einer Eingabe einen völlig anderen Verzeichnisnamen zur Folge.

Dieses Vorgehen ist die Grundlage für die Guix auszeichnenden Funktionalitäten: Unterstützung transaktionsbasierter Paketaktualisierungen und -rücksetzungen, Installation von Paketen für jeden Nutzer sowie Garbage Collection für Pakete (siehe Abschnitt 6.1 [Funktionalitäten], Seite 44).

1.2 GNU-Distribution

Mit Guix kommt eine Distribution des GNU-Systems, die nur aus freier Software[3] besteht. Die Distribution kann für sich allein installiert werden (siehe Kapitel 3 [Systeminstallation], Seite 27), aber Guix kann auch auf einem bestehenden GNU/Linux-System installiert werden. Wenn wir die Anwendungsfälle unterscheiden möchten, bezeichnen wir die alleinstehende Distribution als „Guix System“ (mit englischer Aussprache).

Die Distribution stellt den Kern der GNU-Pakete, also insbesondere GNU libc, GCC und Binutils, sowie zahlreiche zum GNU-Projekt gehörende und nicht dazu gehörende Anwendungen zur Verfügung. Die vollständige Liste verfügbarer Pakete können Sie online (`https://www.gnu.org/software/guix/packages`) einsehen, oder indem Sie `guix package` ausführen (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45):

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

Unser Ziel ist, eine zu 100% freie Software-Distribution von Linux-basierten und von anderen GNU-Varianten anzubieten, mit dem Fokus darauf, das GNU-Projekt und die enge Zusammenarbeit seiner Bestandteile zu befördern, sowie die Programme und Werkzeuge hervorzuheben, die die Nutzer dabei unterstützen, von dieser Freiheit Gebrauch zu machen.

Pakete sind zurzeit auf folgenden Plattformen verfügbar:

`x86_64-linux`
: Intel/AMD-`x86_64`-Architektur, Linux-Libre als Kernel.

`i686-linux`
: Intel-32-Bit-Architektur (IA-32), Linux-Libre als Kernel.

`armhf-linux`
: ARMv7-A-Architektur mit „hard float“, Thumb-2 und NEON, für die EABI „hard-float application binary interface“, mit Linux-Libre als Kernel.

`aarch64-linux`
: 64-Bit-ARMv8-A-Prozessoren, little-endian, mit Linux-Libre als Kernel.

[3] Die Bezeichnung „frei“ steht hier für die Freiheiten, die Nutzern der Software geboten werden (`https://www.gnu.org/philosophy/free-sw.html`).

`i586-gnu` GNU/Hurd (`https://hurd.gnu.org`) auf der Intel-32-Bit-Architektur (IA32).

Diese Konfiguration ist experimentell und befindet sich noch in der Entwicklung. Wenn Sie sie ausprobieren möchten, ist es am einfachsten, eine Instanz des Diensttyps `hurd-vm-service-type` auf Ihrer GNU/Linux-Maschine einzurichten (siehe [transparent-emulation-qemu], Seite 543). Siehe auch Kapitel 22 [Mitwirken], Seite 708, für Informationen, wie Sie helfen können!

`mips64el-linux (eingeschränkte Unterstützung)`
64-Bit-MIPS-Prozessoren, little-endian, speziell die Loongson-Reihe, n32-ABI, mit Linux-Libre als Kernel. Diese Konfiguration wird nicht länger in vollem Umfang unterstützt; insbesondere gibt es *keine* laufenden Bemühungen, die Funktionsfähigkeit dieser Architektur sicherzustellen. Wenn sich jemand findet, der diese Architektur wiederbeleben will, dann ist der Code dafür noch verfügbar.

`powerpc-linux (eingeschränkte Unterstützung)`
32-Bit-PowerPC-Prozessoren, big-endian, speziell der PowerPC G4 mit AltiVec, mit Linux-Libre als Kernel. Diese Konfiguration wird *nicht* in vollem Umfang unterstützt und es gibt *keine* laufenden Bemühungen, die Funktionsfähigkeit dieser Architektur sicherzustellen.

`powerpc64le-linux`
64-Bit-Prozessoren mit Power-Befehlssatz, little-endian, mit Linux-Libre als Kernel. Dazu gehören POWER9-Systeme wie die RYF-zertifizierte Talos-II-Hauptplatine (`https://www.fsf.org/news/talos-ii-mainboard-and-talos-ii-lite-mainboard-now-fsf-certified-to-respect-your-`
Bei der Plattform handelt es sich um eine „Technologievorschau“; obwohl sie unterstützt wird, gibt es noch keine Substitute von der Erstellungsfarm (siehe Abschnitt 6.3 [Substitute], Seite 56) und bei manchen Paketen könnte die Erstellung fehlschlagen (siehe Abschnitt 22.7 [Überblick über gemeldete Fehler und Patches], Seite 731). Dennoch arbeitet die Guix-Gemeinde aktiv daran, diese Unterstützung auszubauen, und jetzt ist eine gute Gelegenheit, sie auszuprobieren und mitzumachen!

`riscv64-linux`
64-Bit-Prozessoren mit RISC-V-Befehlssatz, little-endian, speziell der RV64GC, mit Linux-Libre als Kernel. Bei der Plattform handelt es sich um eine „Technologievorschau“; obwohl sie unterstützt wird, gibt es noch keine Substitute von der Erstellungsfarm (siehe Abschnitt 6.3 [Substitute], Seite 56) und bei manchen Paketen könnte die Erstellung fehlschlagen (siehe Abschnitt 22.7 [Überblick über gemeldete Fehler und Patches], Seite 731). Dennoch arbeitet die Guix-Gemeinde aktiv daran, diese Unterstützung auszubauen, und jetzt ist eine gute Gelegenheit, sie auszuprobieren und mitzumachen!

Mit Guix System *deklarieren* Sie alle Aspekte der Betriebssystemkonfiguration und Guix kümmert sich darum, die Konfiguration auf transaktionsbasierte, reproduzierbare und zustandslose Weise zu instanziieren (siehe Kapitel 12 [Systemkonfiguration], Seite 250). Guix System benutzt den Kernel Linux-libre, das Shepherd-Initialisierungssystem (siehe Abschnitt “Introduction” in *The GNU Shepherd Manual*), die wohlbekannten GNU-Werkzeuge

mit der zugehörigen Werkzeugkette sowie die grafische Umgebung und Systemdienste Ihrer Wahl.

Guix System ist auf allen oben genannten Plattformen außer `mips64el-linux`, `powerpc-linux`, `powerpc64le-linux` und `riscv64-linux` verfügbar.

Informationen, wie auf andere Architekturen oder Kernels portiert werden kann, finden Sie im Abschnitt Kapitel 21 [Portierung], Seite 707.

Diese Distribution aufzubauen basiert auf Kooperation, und Sie sind herzlich eingeladen, dabei mitzumachen! Im Abschnitt Kapitel 22 [Mitwirken], Seite 708, stehen weitere Informationen, wie Sie uns helfen können.

2 Installation

> **Anmerkung:** Wir empfehlen, dieses Shell-basierte Installationsskript (`https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh`) zu benutzen, um Guix auf ein bestehendes GNU/Linux-System zu installieren – im Folgenden als *Fremddistribution* bezeichnet.[1] Das Skript automatisiert das Herunterladen, das Installieren und die anfängliche Konfiguration von Guix. Es sollte als der Administratornutzer „root“ ausgeführt werden.

Wenn es auf einer Fremddistribution installiert wird, ergänzt GNU Guix die verfügbaren Werkzeuge, ohne dass sie sich gegenseitig stören. Guix' Daten befinden sich ausschließlich in zwei Verzeichnissen, üblicherweise `/gnu/store` und `/var/guix`; andere Dateien auf Ihrem System wie `/etc` bleiben unberührt.

Sobald es installiert ist, kann Guix durch Ausführen von `guix pull` aktualisiert werden (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65).

Sollten Sie es vorziehen, die Installationsschritte manuell durchzuführen, oder falls Sie Anpassungen daran vornehmen möchten, könnten sich die folgenden Unterabschnitte als nützlich erweisen. Diese beschreiben die Software-Voraussetzungen von Guix und wie man es manuell installiert, so dass man es benutzen kann.

2.1 Aus Binärdatei installieren

Dieser Abschnitt beschreibt, wie sich Guix auf einem beliebigen System aus einem alle Komponenten umfassenden Tarball installieren lässt, der Binärdateien für Guix und all seine Abhängigkeiten liefert. Dies geht in der Regel schneller, als Guix aus seinen Quelldateien zu installieren, was in den nächsten Abschnitten beschrieben wird. Vorausgesetzt wird hier lediglich, dass GNU tar und Xz verfügbar sind.

> **Anmerkung:** Wir empfehlen, dass Sie dieses Installations-Skript für die Shell (`https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh`) verwenden, welches Guix automatisch herunterlädt, installiert und eine erste Konfiguration von Guix mit sich bringt. Es sollte als der Administratornutzer (als „root“) ausgeführt werden.
>
> ```
> cd /tmp
> wget https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh
> chmod +x guix-install.sh
> ./guix-install.sh
> ```
>
> Wenn Sie Debian oder ein Debian-Derivat wie Ubuntu verwenden, können Sie stattdessen das Guix-Paket installieren (obwohl es eine ältere Version als 1.4.0 mitbringen könnte, können Sie es anschließend über den Befehl '`guix pull`' aktualisieren):
>
> ```
> sudo apt install guix
> ```
>
> Das Gleiche gilt auf openSUSE:
>
> ```
> sudo zypper install guix
> ```

[1] Dieser Abschnitt bezieht sich auf die Installation des Paketverwaltungswerkzeugs, das auf ein bestehendes GNU/Linux-System aufsetzend installiert werden kann. Wenn Sie stattdessen das vollständige GNU-Betriebssystem installieren möchten, lesen Sie Kapitel 3 [Systeminstallation], Seite 27.

Wenn Sie das erledigt haben, werfen Sie einen Blick auf Abschnitt 2.6 [Anwendungen einrichten], Seite 23, für weitere Einstellungen, die Sie vielleicht vornehmen möchten, und lesen Sie die ersten Schritte im Kapitel 5 [Einstieg in Guix], Seite 41, um loszulegen!

Die Installation läuft so ab:

1. Laden Sie den binären Tarball von ‘`https://ftp.gnu.org/gnu/guix/guix-binary-1.4.0.x86_64-linux` herunte. Falls Sie Guix auf einer Maschine mit `i686`-Architektur (32 Bit) einrichten, auf der bereits der Linux-Kernel läuft, ersetzen Sie `x86_64-linux` durch `i686-linux` oder entsprechend für andere Maschinen (siehe Abschnitt 1.2 [GNU-Distribution], Seite 2).

   Achten Sie darauf, auch die zugehörige `.sig`-Datei herunterzuladen und verifizieren Sie damit die Authentizität des Tarballs, ungefähr so:

   ```
   $ wget https://ftp.gnu.org/gnu/guix/guix-binary-1.4.0.x86_64-linux.tar.xz.sig
   $ gpg --verify guix-binary-1.4.0.x86_64-linux.tar.xz.sig
   ```

   Falls dieser Befehl fehlschlägt, weil Sie nicht über den nötigen öffentlichen Schlüssel verfügen, können Sie ihn mit diesem Befehl importieren:

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

   und den Befehl `gpg --verify` erneut ausführen.

   Beachten Sie, dass eine Warnung wie „Dieser Schlüssel trägt keine vertrauenswürdige Signatur!“ normal ist.

2. Nun müssen Sie zum Administratornutzer `root` wechseln. Abhängig von Ihrer Distribution müssen Sie dazu etwa `su -` oder `sudo -i` ausführen. Danach führen Sie als `root`-Nutzer aus:

   ```
   # cd /tmp
   # tar --warning=no-timestamp -xf \
        /pfad/zur/guix-binary-1.4.0.x86_64-linux.tar.xz
   # mv var/guix /var/ && mv gnu /
   ```

   Dadurch wird `/gnu/store` (siehe Abschnitt 9.9 [Der Store], Seite 165) und `/var/guix` erzeugt. Letzteres enthält ein fertiges Guix-Profil für den Administratornutzer `root` (wie im nächsten Schritt beschrieben).

   Entpacken Sie den Tarball *nicht* auf einem schon funktionierenden Guix-System, denn es würde seine eigenen essenziellen Dateien überschreiben.

   Die Befehlszeilenoption `--warning=no-timestamp` stellt sicher, dass GNU tar nicht vor „unplausibel alten Zeitstempeln“ warnt (solche Warnungen traten bei GNU tar 1.26 und älter auf, neue Versionen machen keine Probleme). Sie treten auf, weil alle Dateien im Archiv als Änderungszeitpunkt 1 eingetragen bekommen haben (das bezeichnet den 1. Januar 1970). Das ist Absicht, damit der Inhalt des Archivs nicht davon abhängt, wann es erstellt wurde, und es somit reproduzierbar wird.

3. Machen Sie das Profil als `~root/.config/guix/current` verfügbar, wo `guix pull` es aktualisieren kann (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65):

   ```
   # mkdir -p ~root/.config/guix
   # ln -sf /var/guix/profiles/per-user/root/current-guix \
            ~root/.config/guix/current
   ```

„Sourcen“ Sie `etc/profile`, um `PATH` und andere relevante Umgebungsvariable zu ergänzen:

```
# GUIX_PROFILE="`echo ~root`/.config/guix/current" ; \
  source $GUIX_PROFILE/etc/profile
```

4. Erzeugen Sie Nutzergruppe und Nutzerkonten für die Erstellungs-Benutzer wie folgt (siehe Abschnitt 2.4.1 [Einrichten der Erstellungsumgebung], Seite 11).
5. Führen Sie den Daemon aus, und lassen Sie ihn automatisch bei jedem Hochfahren starten.

   Wenn Ihre Wirts-Distribution systemd als „init“-System verwendet, können Sie das mit folgenden Befehlen veranlassen:

   ```
   # cp ~root/.config/guix/current/lib/systemd/system/gnu-store.mount \
        ~root/.config/guix/current/lib/systemd/system/guix-daemon.service \
        /etc/systemd/system/
   # systemctl enable --now gnu-store.mount guix-daemon
   ```

   Außerdem, wenn Sie möchten, dass regelmäßig `guix gc` durchgeführt wird:

   ```
   # cp ~root/.config/guix/current/lib/systemd/system/guix-gc.service \
        ~root/.config/guix/current/lib/systemd/system/guix-gc.timer \
        /etc/systemd/system/
   # systemctl enable --now guix-gc.timer
   ```

   Vielleicht möchten Sie die in `guix-gc.service` verwendeten Befehlszeilenoptionen an Ihre Bedürfnisse anpassen (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61).

   Wenn Ihre Wirts-Distribution als „init“-System Upstart verwendet:

   ```
   # initctl reload-configuration
   # cp ~root/.config/guix/current/lib/upstart/system/guix-daemon.conf \
        /etc/init/
   # start guix-daemon
   ```

   Andernfalls können Sie den Daemon immer noch manuell starten, mit:

   ```
   # ~root/.config/guix/current/bin/guix-daemon \
         --build-users-group=guixbuild
   ```

6. Stellen Sie den `guix`-Befehl auch anderen Nutzern Ihrer Maschine zur Verfügung, zum Beispiel so:

   ```
   # mkdir -p /usr/local/bin
   # cd /usr/local/bin
   # ln -s /var/guix/profiles/per-user/root/current-guix/bin/guix
   ```

   Es ist auch eine gute Idee, die Info-Version dieses Handbuchs ebenso verfügbar zu machen:

   ```
   # mkdir -p /usr/local/share/info
   # cd /usr/local/share/info
   # for i in /var/guix/profiles/per-user/root/current-guix/share/info/* ;
     do ln -s $i ; done
   ```

   Auf diese Art wird, unter der Annahme, dass bei Ihnen `/usr/local/share/info` im Suchpfad eingetragen ist, das Ausführen von `info guix.de` dieses Handbuch öffnen (siehe Abschnitt “Other Info Directories” in *GNU Texinfo* hat weitere Details, wie Sie den Info-Suchpfad ändern können).

7. Um Substitute von `ci.guix.gnu.org`, `bordeaux.guix.gnu.org` oder einem Spiegelserver davon zu benutzen (siehe Abschnitt 6.3 [Substitute], Seite 56), müssen sie erst autorisiert werden:

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

   Anmerkung: Wenn Sie Substitute abgeschaltet lassen, muss Guix *alles* auf Ihrem Rechner aus dem Quellcode heraus erstellen, wodurch jede Installation und jede Aktualisierung sehr aufwendig wird. Siehe Abschnitt 6.3.7 [Vom Vertrauen gegenüber Binärdateien], Seite 60, für eine Erörterung, aus welchen Gründen man Substitute abschalten wollen könnte.

8. Alle Nutzer müssen womöglich ein paar zusätzliche Schritte ausführen, damit ihre Guix-Umgebung genutzt werden kann, siehe Abschnitt 2.6 [Anwendungen einrichten], Seite 23.

Voilà, die Installation ist fertig!

Sie können nachprüfen, dass Guix funktioniert, indem Sie ein Beispielpaket in das root-Profil installieren:

```
# guix install hello
```

Der Tarball zur Installation aus einer Binärdatei kann einfach durch Ausführung des folgenden Befehls im Guix-Quellbaum (re-)produziert und verifiziert werden:

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

... was wiederum dies ausführt:

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

Siehe Abschnitt 8.3 [Aufruf von guix pack], Seite 99, für weitere Informationen zu diesem praktischen Werkzeug.

2.2 Voraussetzungen

Dieser Abschnitt listet Voraussetzungen auf, um Guix aus seinem Quellcode zu erstellen. Der Erstellungsprozess für Guix ist derselbe wie für andere GNU-Software und wird hier nicht beschrieben. Bitte lesen Sie die Dateien `README` und `INSTALL` im Guix-Quellbaum, um weitere Details zu erfahren.

GNU Guix kann von seinem Webauftritt unter `http://www.gnu.org/software/guix/` heruntergeladen werden.

GNU Guix hat folgende Pakete als Abhängigkeiten:

- GNU Guile (`https://gnu.org/software/guile/`), Version 3.0.x, Version 3.0.3 oder neuer,
- Guile-Gcrypt (`https://notabug.org/cwebber/guile-gcrypt`), Version 0.1.0 oder neuer,

- Guile-GnuTLS (`https://gitlab.com/gnutls/guile/`) (siehe die Abschnitt "Guile Preparations" in *GnuTLS-Guile*)[2],
- Guile-SQLite3 (`https://notabug.org/guile-sqlite3/guile-sqlite3`), Version 0.1.0 oder neuer,
- Guile-zlib (`https://notabug.org/guile-zlib/guile-zlib`), Version 0.1.0 oder neuer,
- 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 oder neuer,
- Guile-JSON (`https://savannah.nongnu.org/projects/guile-json/`) 4.3.0 oder später,
- GNU Make (`https://www.gnu.org/software/make/`).

Folgende Abhängigkeiten sind optional:

- Unterstützung für das Auslagern von Erstellungen (siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13) und `guix copy` (siehe Abschnitt 10.13 [Aufruf von guix copy], Seite 241) hängt von Guile-SSH (`https://github.com/artyom-poptsov/guile-ssh`), Version 0.13.0 oder neuer, ab.
- Guile-zstd (`https://notabug.org/guile-zstd/guile-zstd`), für die Kompression und Dekompression mit zstd in `guix publish` und für Substitute (siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233).
- Guile-Semver (`https://ngyro.com/software/guile-semver.html`) für den `crate`-Importer (siehe Abschnitt 10.5 [Aufruf von guix import], Seite 206).
- Guile-Lib (`https://www.nongnu.org/guile-lib/doc/ref/htmlprag/`) für den `go`-Importer (siehe Abschnitt 10.5 [Aufruf von guix import], Seite 206) und für einige der Aktualisierungsprogramme (siehe Abschnitt 10.6 [Aufruf von guix refresh], Seite 214).
- Wenn libbz2 (`http://www.bzip.org`) verfügbar ist, kann `guix-daemon` damit Erstellungsprotokolle komprimieren.

Sofern nicht `--disable-daemon` beim Aufruf von `configure` übergeben wurde, benötigen Sie auch folgende Pakete:

- GNU libgcrypt (`https://gnupg.org/`),
- SQLite 3 (`https://sqlite.org`),
- GCC's g++ (`https://gcc.gnu.org`) mit Unterstützung für den C++11-Standard.

Sollten Sie Guix auf einem System konfigurieren, auf dem Guix bereits installiert ist, dann stellen Sie sicher, dasselbe Zustandsverzeichnis wie für die bestehende Installation zu verwenden. Benutzen Sie dazu die Befehlszeilenoption `--localstatedir` des `configure`-Skripts (siehe Abschnitt "Directory Variables" in *GNU Coding Standards*). Die *localstatedir*-Option wird normalerweise auf den Wert `/var` festgelegt. Das `configure`-Skript schützt vor ungewollter Fehlkonfiguration der *localstatedir*, damit Sie nicht versehentlich Ihren Store verfälschen (siehe Abschnitt 9.9 [Der Store], Seite 165).

[2] Bis einschließlich Version 3.7.8 wurden die Guile-Anbindungen für GnuTLS (`https://gnutls.org/`) bei GnuTLS mitgeliefert.

2.3 Den Testkatalog laufen lassen

Nachdem `configure` und `make` erfolgreich durchgelaufen sind, ist es ratsam, den Testkatalog auszuführen. Er kann dabei helfen, Probleme mit der Einrichtung oder Systemumgebung zu finden, oder auch Probleme in Guix selbst – und Testfehler zu melden ist eine wirklich gute Art und Weise, bei der Verbesserung von Guix mitzuhelfen. Um den Testkatalog auszuführen, geben Sie Folgendes ein:

```
make check
```

Testfälle können parallel ausgeführt werden. Sie können die Befehlszeiltenoption `-j` von GNU make benutzen, damit es schneller geht. Der erste Durchlauf kann auf neuen Maschinen ein paar Minuten dauern, nachfolgende Ausführungen werden schneller sein, weil der für die Tests erstellte Store schon einige Dinge zwischengespeichert haben wird.

Es ist auch möglich, eine Teilmenge der Tests laufen zu lassen, indem Sie die `TESTS`-Variable des Makefiles ähnlich wie in diesem Beispiel definieren:

```
make check TESTS="tests/store.scm tests/cpio.scm"
```

Standardmäßig werden Testergebnisse pro Datei angezeigt. Um die Details jedes einzelnen Testfalls zu sehen, können Sie wie in diesem Beispiel die `SCM_LOG_DRIVER_FLAGS`-Variable des Makefiles definieren:

```
make check TESTS="tests/base64.scm" SCM_LOG_DRIVER_FLAGS="--brief=no"
```

Mit dem eigens geschriebenen SRFI-64-Testtreiber für Automake, über den der „`check`“-Testkatalog läuft (zu finden in `build-aux/test-driver.scm`), können auch die Testfälle genauer ausgewählt werden, die ausgeführt werden sollen. Dazu dienen dessen Befehlszeilenoptionen `--select` und `--exclude`. Hier ist ein Beispiel, um alle Testfälle aus der Testdatei `tests/packages.scm` auszuführen, deren Name mit „transaction-upgrade-entry“ beginnt:

```
export SCM_LOG_DRIVER_FLAGS="--select=^transaction-upgrade-entry"
make check TESTS="tests/packages.scm"
```

Möchte man die Ergebnisse fehlgeschlagener Tests direkt über die Befehlszeile einsehen, fügt man die Befehlszeilenoption `--errors-only=yes` in die Makefile-Variable `SCM_LOG_DRIVER_FLAGS` ein und setzt Automakes Makefile-Variable `VERBOSE`, etwa so:

```
make check SCM_LOG_DRIVER_FLAGS="--brief=no --errors-only=yes" VERBOSE=1
```

Sie können die Befehlszeilenoption `--show-duration=yes` benutzen, damit ausgegeben wird, wie lange jeder einzelne Testfall gebraucht hat, in Kombination mit `--brief=no`:

```
make check SCM_LOG_DRIVER_FLAGS="--brief=no --show-duration=yes"
```

Siehe Abschnitt “Parallel Test Harness” in *GNU Automake* für mehr Informationen über den parallelen Testrahmen von Automake.

Kommt es zum Fehlschlag, senden Sie bitte eine E-Mail an `bug-guix@gnu.org` und fügen Sie die Datei `test-suite.log` als Anhang bei. Bitte geben Sie dabei in Ihrer Nachricht die benutzte Version von Guix an sowie die Versionsnummern der Abhängigkeiten (siehe Abschnitt 2.2 [Voraussetzungen], Seite 8).

Guix wird auch mit einem Testkatalog für das ganze System ausgeliefert, der vollständige Instanzen des „Guix System“-Betriebssystems testet. Er kann nur auf Systemen benutzt werden, auf denen Guix bereits installiert ist, mit folgendem Befehl:

```
make check-system
```

Oder, auch hier, indem Sie `TESTS` definieren, um eine Teilmenge der auszuführenden Tests anzugeben:

```
make check-system TESTS="basic mcron"
```

Diese Systemtests sind in den `(gnu tests ...)`-Modulen definiert. Sie funktionieren, indem Sie das getestete Betriebssystem mitsamt schlichter Instrumentierung in einer virtuellen Maschine (VM) ausführen. Die Tests können aufwendige Berechnungen durchführen oder sie günstig umgehen, je nachdem, ob für ihre Abhängigkeiten Substitute zur Verfügung stehen (siehe Abschnitt 6.3 [Substitute], Seite 56). Manche von ihnen nehmen viel Speicherplatz in Anspruch, um die VM-Abbilder zu speichern.

Auch hier gilt: Falls Testfehler auftreten, senden Sie bitte alle Details an `bug-guix@gnu.org`.

2.4 Den Daemon einrichten

Operationen wie das Erstellen eines Pakets oder Laufenlassen des Müllsammlers werden alle durch einen spezialisierten Prozess durchgeführt, den *Erstellungs-Daemon*, im Auftrag seiner Kunden (den Clients). Nur der Daemon darf auf den Store und seine zugehörige Datenbank zugreifen. Daher wird jede den Store verändernde Operation durch den Daemon durchgeführt. Zum Beispiel kommunizieren Befehlszeilenwerkzeuge wie `guix package` und `guix build` mit dem Daemon (mittels entfernter Prozeduraufrufe), um ihm Anweisungen zu geben, was er tun soll.

Folgende Abschnitte beschreiben, wie Sie die Umgebung des Erstellungs-Daemons ausstatten sollten. Siehe auch Abschnitt 6.3 [Substitute], Seite 56, für Informationen darüber, wie Sie es dem Daemon ermöglichen, vorerstellte Binärdateien herunterzuladen.

2.4.1 Einrichten der Erstellungsumgebung

In einem normalen Mehrbenutzersystem werden Guix und sein Daemon – das Programm `guix-daemon` – vom Systemadministrator installiert; `/gnu/store` gehört `root` und `guix-daemon` läuft als `root`. Nicht mit erweiterten Rechten ausgestattete Nutzer können Guix-Werkzeuge benutzen, um Pakete zu erstellen oder anderweitig auf den Store zuzugreifen, und der Daemon wird dies für sie erledigen und dabei sicherstellen, dass der Store in einem konsistenten Zustand verbleibt und sich die Nutzer erstellte Pakete teilen.

Wenn `guix-daemon` als Administratornutzer `root` läuft, wollen Sie aber vielleicht dennoch nicht, dass Paketerstellungsprozesse auch als `root` ablaufen, aus offensichtlichen Sicherheitsgründen. Um dies zu vermeiden, sollte ein besonderer Pool aus *Erstellungsbenutzern* geschaffen werden, damit vom Daemon gestartete Erstellungsprozesse ihn benutzen. Diese Erstellungsbenutzer müssen weder eine Shell noch ein Persönliches Verzeichnis zugewiesen bekommen, sie werden lediglich benutzt, wenn der Daemon `root`-Rechte in Erstellungsprozessen ablegt. Mehrere solche Benutzer zu haben, ermöglicht es dem Daemon, verschiedene Erstellungsprozessen unter verschiedenen Benutzeridentifikatoren (UIDs) zu starten, was garantiert, dass sie einander nicht stören – eine essenzielle Funktionalität, da Erstellungen als reine Funktionen angesehen werden (siehe Kapitel 1 [Einführung], Seite 1).

Auf einem GNU/Linux-System kann ein Pool von Erstellungsbenutzern wie folgt erzeugt werden (mit Bash-Syntax und den Befehlen von `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-Erstellungsbenutzer $i" --system  \
            guixbuilder$i;
  done
```

Die Anzahl der Erstellungsbenutzer entscheidet, wie viele Erstellungsaufträge parallel ausgeführt werden können, wie es mit der Befehlszeilenoption `--max-jobs` vorgegeben werden kann (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18). Um `guix system vm` und ähnliche Befehle nutzen zu können, müssen Sie die Erstellungsbenutzer unter Umständen zur `kvm`-Benutzergruppe hinzufügen, damit sie Zugriff auf `/dev/kvm` haben, mit `-G guixbuild,kvm` statt `-G guixbuild` (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

Das Programm `guix-daemon` kann mit dem folgenden Befehl als `root` gestartet werden[3]:

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

Auf diese Weise startet der Daemon Erstellungsprozesse in einem chroot als einer der `guixbuilder`-Benutzer. Auf GNU/Linux enthält die chroot-Umgebung standardmäßig nichts außer:

- einem minimalen `/dev`-Verzeichnis, was größtenteils vom `/dev` des Wirtssystems unabhängig erstellt wurde[4],
- dem `/proc`-Verzeichnis, es zeigt nur die Prozesse des Containers, weil ein separater Namensraum für Prozess-IDs (PIDs) benutzt wird,
- `/etc/passwd` mit einem Eintrag für den aktuellen Benutzer und einem Eintrag für den Benutzer `nobody`,
- `/etc/group` mit einem Eintrag für die Gruppe des Benutzers,
- `/etc/hosts` mit einem Eintrag, der `localhost` auf `127.0.0.1` abbildet,
- einem `/tmp`-Verzeichnis mit Schreibrechten.

Im chroot ist kein `/home`-Verzeichnis enthalten und die Umgebungsvariable `HOME` ist auf das *nicht* existierende Verzeichnis `/homeless-shelter` festgelegt. Dadurch fallen unangemessene Verwendungen von `HOME` in den Erstellungs-Skripts von Paketen auf.

Sie können beeinflussen, in welchem Verzeichnis der Daemon Verzeichnisbäume zur Erstellung unterbringt, indem sie den Wert der Umgebungsvariablen `TMPDIR` ändern. Allerdings heißt innerhalb des chroots der Erstellungsbaum immer `/tmp/guix-build-`*Name*`.drv-0`, wobei *Name* der Ableitungsname ist – z.B. `coreutils-8.24`. Dadurch hat der Wert von `TMPDIR` keinen Einfluss auf die Erstellungsumgebung, wodurch Unterschiede vermieden werden, falls Erstellungsprozesse den Namen ihres Erstellungsbaumes einfangen.

---

[3] Wenn Ihre Maschine systemd als „init“-System verwendet, genügt es, die Datei *prefix*`/lib/systemd/system/guix-daemon.service` nach `/etc/systemd/system` zu kopieren, damit `guix-daemon` automatisch gestartet wird. Ebenso können Sie, wenn Ihre Maschine Upstart als „init“-System benutzt, die Datei *prefix*`/lib/upstart/system/guix-daemon.conf` nach `/etc/init` kopieren.

[4] „Größtenteils“, denn obwohl die Menge an Dateien, die im `/dev` des chroots vorkommen, fest ist, können die meisten dieser Dateien nur dann erstellt werden, wenn das Wirtssystem sie auch hat.

Der Daemon befolgt außerdem den Wert der Umgebungsvariablen `http_proxy` und `https_proxy` für von ihm durchgeführte HTTP- und HTTPS-Downloads, sei es für Ableitungen mit fester Ausgabe (siehe Abschnitt 9.10 [Ableitungen], Seite 167) oder für Substitute (siehe Abschnitt 6.3 [Substitute], Seite 56).

Wenn Sie Guix als ein Benutzer ohne erweiterte Rechte installieren, ist es dennoch möglich, `guix-daemon` auszuführen, sofern Sie `--disable-chroot` übergeben. Allerdings können Erstellungsprozesse dann nicht voneinander und vom Rest des Systems isoliert werden. Daher können sich Erstellungsprozesse gegenseitig stören und auf Programme, Bibliotheken und andere Dateien zugreifen, die dem restlichen System zur Verfügung stehen – was es deutlich schwerer macht, sie als *reine* Funktionen aufzufassen.

2.4.2 Nutzung der Auslagerungsfunktionalität

Wenn erwünscht, kann der Erstellungs-Daemon Ableitungserstellungen auf andere Maschinen *auslagern*, auf denen Guix läuft, mit Hilfe des `offload`-*Build-Hooks*[5]. Wenn diese Funktionalität aktiviert ist, wird eine nutzerspezifizierte Liste von Erstellungsmaschinen aus `/etc/guix/machines.scm` gelesen. Wann immer eine Erstellung angefragt wird, zum Beispiel durch `guix build`, versucht der Daemon, sie an eine der Erstellungsmaschinen auszulagern, die die Einschränkungen der Ableitung erfüllen, insbesondere ihre Systemtypen – z.B. `x86_64-linux`. Eine einzelne Maschine kann mehrere Systemtypen haben, entweder weil ihre Architektur eine native Unterstützung vorsieht, weil Emulation eingerichtet wurde (siehe [transparent-emulation-qemu], Seite 543) oder beides. Fehlende Voraussetzungen für die Erstellung werden über SSH auf die Zielmaschine kopiert, welche dann mit der Erstellung weitermacht. Hat sie Erfolg damit, so werden die Ausgabe oder Ausgaben der Erstellung zurück auf die ursprüngliche Maschine kopiert. Die Auslagerungsfunktion verfügt über einen einfachen Planungsalgorithmus (einen Scheduler), mit dem versucht wird, die jeweils beste Maschine auszuwählen. Unter den verfügbaren wird die beste Maschine nach Kriterien wie diesen ausgewählt:

1. Ob Zeitfenster für Erstellungen frei sind („build slots“). Eine Erstellungsmaschine kann so viele Erstellungszeitfenster (Verbindungen) unterhalten wie es dem Wert des `parallel-builds`-Feldes des `build-machine`-Objekts entspricht.
2. Was ihre relative Geschwindigkeit ist, gemäß ihrer Definition im `speed`-Feld ihres `build-machine`-Objekts.
3. Ihrer Auslastung („load“). Die normalisierte Auslastung der Maschine darf einen Schwellwert nicht überschreiten. Dieser ist über das `overload-threshold`-Feld ihres `build-machine`-Objekts einstellbar.
4. Verfügbarem Speicherplatz auf ihrem Datenträger. Es müssen mehr als 100 MiB verfügbar sein.

Die Datei `/etc/guix/machines.scm` sieht normalerweise so aus:

```
(list (build-machine
        (name "eightysix.example.org")
        (systems (list "x86_64-linux" "i686-linux"))
        (host-key "ssh-ed25519 AAAAC3Nza..."))
```

[5] Diese Funktionalität ist nur verfügbar, wenn Guile-SSH (`https://github.com/artyom-poptsov/guile-ssh`) vorhanden ist.

```
  (user "bob")
  (speed 2.))     ;unglaublich schnell!

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

  ;; Weil 'guix offload' vom 'guix-daemon' als
  ;; Administratornutzer root gestartet wird.
  (private-key "/root/.ssh/identität-für-guix")))
```

Im obigen Beispiel geben wir eine Liste mit zwei Erstellungsmaschinen vor, eine für die `x86_64`- und `i686`-Architektur und eine für die `aarch64`-Architektur.

Tatsächlich ist diese Datei – wenig überraschend! – eine Scheme-Datei, die ausgewertet wird, wenn der `offload`-Hook gestartet wird. Der Wert, den sie zurückliefert, muss eine Liste von `build-machine`-Objekten sein. Obwohl dieses Beispiel eine feste Liste von Erstellungsmaschinen zeigt, könnte man auch auf die Idee kommen, etwa mit DNS-SD eine Liste möglicher im lokalen Netzwerk entdeckter Erstellungsmaschinen zu liefern (siehe Abschnitt "Introduction" in *Using Avahi in Guile Scheme Programs*). Der Datentyp `build-machine` wird im Folgenden weiter ausgeführt.

`build-machine` [Datentyp]

Dieser Datentyp repräsentiert Erstellungsmaschinen, an die der Daemon Erstellungen auslagern darf. Die wichtigen Felder sind:

`name` Der Rechnername (d.h. der Hostname) der entfernten Maschine.

`systems` Die Systemtypen, die die entfernte Maschine unterstützt – z.B. `(list "x86_64-linux" "i686-linux")`.

`user` Das Benutzerkonto, mit dem eine Verbindung zur entfernten Maschine über SSH aufgebaut werden soll. Beachten Sie, dass das SSH-Schlüsselpaar *nicht* durch eine Passphrase geschützt sein darf, damit nicht-interaktive Anmeldungen möglich sind.

`host-key` Dies muss der *öffentliche SSH-Rechnerschlüssel* („Host Key") der Maschine im OpenSSH-Format sein. Er wird benutzt, um die Identität der Maschine zu prüfen, wenn wir uns mit ihr verbinden. Er ist eine lange Zeichenkette, die ungefähr so aussieht:

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

Wenn auf der Maschine der OpenSSH-Daemon, `sshd`, läuft, ist der Rechnerschlüssel in einer Datei wie `/etc/ssh/ssh_host_ed25519_key.pub` zu finden.

Wenn auf der Maschine der SSH-Daemon von GNU lsh, nämlich `lshd`, läuft, befindet sich der Rechnerschlüssel in `/etc/lsh/host-key.pub` oder einer ähnlichen Datei. Er kann ins OpenSSH-Format umgewandelt werden durch `lsh-export-key` (siehe Abschnitt "Converting keys" in *LSH Manual*):

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

Eine Reihe optionaler Felder kann festgelegt werden:

`port` (Vorgabe: `22`)
: Portnummer des SSH-Servers auf der Maschine.

`private-key` (Vorgabe: `~root/.ssh/id_rsa`)
: Die Datei mit dem privaten SSH-Schlüssel, der beim Verbinden zur Maschine genutzt werden soll, im OpenSSH-Format. Dieser Schlüssel darf nicht mit einer Passphrase geschützt sein.

  Beachten Sie, dass als Vorgabewert der private Schlüssel *des root-Benutzers* genommen wird. Vergewissern Sie sich, dass er existiert, wenn Sie die Standardeinstellung verwenden.

`compression` (Vorgabe: `"zlib@openssh.com,zlib"`)
`compression-level` (Vorgabe: `3`)
: Die Kompressionsmethoden auf SSH-Ebene und das angefragte Kompressionsniveau.

  Beachten Sie, dass Auslagerungen SSH-Kompression benötigen, um beim Übertragen von Dateien an Erstellungsmaschinen und zurück weniger Bandbreite zu benutzen.

`daemon-socket` (Vorgabe: `"/var/guix/daemon-socket/socket"`)
: Dateiname des Unix-Sockets, auf dem `guix-daemon` auf der Maschine lauscht.

`overload-threshold` (Vorgabe: `0.8`)
: Der Schwellwert für die Auslastung (englisch „load“), ab der eine potentielle Auslagerungsmaschine für den Auslagerungsplaner nicht mehr in Betracht kommt. Der Wert entspricht grob der gesamten Prozessornutzung der Erstellungsmaschine. Er reicht von 0.0 (0%) bis 1.0 (100%). Wenn er ignoriert werden soll, dann setzen Sie `overload-threshold` auf `#f`.

`parallel-builds` (Vorgabe: `1`)
: Die Anzahl der Erstellungen, die auf der Maschine parallel ausgeführt werden können.

`speed` (Vorgabe: `1.0`)
: Ein „relativer Geschwindigkeitsfaktor“. Der Auslagerungsplaner gibt tendenziell Maschinen mit höherem Geschwindigkeitsfaktor den Vorrang.

`features` (Vorgabe: `'()`)
: Eine Liste von Zeichenketten, die besondere von der Maschine unterstützte Funktionalitäten bezeichnen. Ein Beispiel ist `"kvm"` für Maschinen, die über die KVM-Linux-Module zusammen mit entsprechender Hardware-Unterstützung verfügen. Ableitungen können Funktionalitäten dem Namen nach anfragen und werden dann auf passenden Erstellungsmaschinen eingeplant.

Der Befehl `guix` muss sich im Suchpfad der Erstellungsmaschinen befinden. Um dies nachzuprüfen, können Sie Folgendes ausführen:

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

Es gibt noch eine weitere Sache zu tun, sobald `machines.scm` eingerichtet ist. Wie zuvor erklärt, werden beim Auslagern Dateien zwischen den Stores der Maschinen hin- und hergeschickt. Damit das funktioniert, müssen Sie als Erstes ein Schlüsselpaar auf jeder Maschine erzeugen, damit der Daemon signierte Archive mit den Dateien aus dem Store versenden kann (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74):

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

Jede Erstellungsmaschine muss den Schlüssel der Hauptmaschine autorisieren, damit diese Store-Objekte von der Hauptmaschine empfangen kann:

```
# guix archive --authorize < öffentlicher-schlüssel-hauptmaschine.txt
```

Andersherum muss auch die Hauptmaschine den jeweiligen Schlüssel jeder Erstellungsmaschine autorisieren.

Der ganze Umstand mit den Schlüsseln soll ausdrücken, dass sich Haupt- und Erstellungsmaschinen paarweise gegenseitig vertrauen. Konkret kann der Erstellungs-Daemon auf der Hauptmaschine die Unverfälschtheit von den Erstellungsmaschinen empfangener Dateien gewährleisten (und umgekehrt), und auch dass sie nicht sabotiert wurden und mit einem autorisierten Schlüssel signiert wurden.

Um zu testen, ob Ihr System funktioniert, führen Sie diesen Befehl auf der Hauptmaschine aus:

```
# guix offload test
```

Dadurch wird versucht, zu jeder Erstellungsmaschine eine Verbindung herzustellen, die in `/etc/guix/machines.scm` angegeben wurde, sichergestellt, dass auf jeder Guix nutzbar ist, und jeweils versucht, etwas auf die Erstellungsmaschine zu exportieren und von dort zu importieren. Dabei auftretende Fehler werden gemeldet.

Wenn Sie stattdessen eine andere Maschinendatei verwenden möchten, geben Sie diese einfach auf der Befehlszeile an:

```
# guix offload test maschinen-qualif.scm
```

Letztendlich können Sie hiermit nur die Teilmenge der Maschinen testen, deren Name zu einem regulären Ausdruck passt:

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

Um die momentane Auslastung aller Erstellungsrechner anzuzeigen, führen Sie diesen Befehl auf dem Hauptknoten aus:

```
# guix offload status
```

2.4.3 SELinux-Unterstützung

Guix enthält eine SELinux-Richtliniendatei („Policy“) unter `etc/guix-daemon.cil`, die auf einem System installiert werden kann, auf dem SELinux aktiviert ist, damit Guix-Dateien gekennzeichnet sind und um das erwartete Verhalten des Daemons anzugeben. Da Guix System keine Grundrichtlinie („Base Policy“) für SELinux bietet, kann diese Richtlinie für den Daemon auf Guix System nicht benutzt werden.

2.4.3.1 Installieren der SELinux-Policy

Um die Richtlinie (Policy) zu installieren, führen Sie folgenden Befehl mit Administratorrechten aus:

```
semodule -i etc/guix-daemon.cil
```

Kennzeichnen Sie dann das Dateisystem neu mit `restorecon` oder einem anderen, von Ihrem System angebotenen Mechanismus.

Sobald die Richtlinie installiert ist, das Dateisystem neu gekennzeichnet wurde und der Daemon neugestartet wurde, sollte er im Kontext `guix_daemon_t` laufen. Sie können dies mit dem folgenden Befehl nachprüfen:

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

Beobachten Sie die Protokolldateien von SELinux, wenn Sie einen Befehl wie `guix build hello` ausführen, um sich zu überzeugen, dass SELinux alle notwendigen Operationen gestattet.

2.4.3.2 Einschränkungen

Diese Richtlinie ist nicht perfekt. Im Folgenden finden Sie eine Liste von Einschränkungen oder merkwürdigen Verhaltensweisen, die bedacht werden sollten, wenn man die mitgelieferte SELinux-Richtlinie für den Guix-Daemon einspielt.

1. `guix_daemon_socket_t` wird nicht wirklich benutzt. Keine der Socket-Operationen benutzt Kontexte, die irgendetwas mit `guix_daemon_socket_t` zu tun haben. Es schadet nicht, diese ungenutzte Kennzeichnung zu haben, aber es wäre besser, für die Kennzeichnung auch Socket-Regeln festzulegen.
2. `guix gc` kann nicht auf beliebige Verknüpfungen zu Profilen zugreifen. Die Kennzeichnung des Ziels einer symbolischen Verknüpfung ist notwendigerweise unabhängig von der Dateikennzeichnung der Verknüpfung. Obwohl alle Profile unter $localstatedir gekennzeichnet sind, erben die Verknüpfungen auf diese Profile die Kennzeichnung desjenigen Verzeichnisses, in dem sie sich befinden. Für Verknüpfungen im Persönlichen Verzeichnis des Benutzers ist das `user_home_t`, aber Verknüpfungen aus dem Persönlichen Verzeichnis des Administratornutzers, oder `/tmp`, oder das Arbeitsverzeichnis des HTTP-Servers, etc., funktioniert das nicht. `guix gc` würde es nicht gestattet, diese Verknüpfungen auszulesen oder zu verfolgen.
3. Die vom Daemon gebotene Funktionalität, auf TCP-Verbindungen zu lauschen, könnte nicht mehr funktionieren. Dies könnte zusätzliche Regeln brauchen, weil SELinux Netzwerk-Sockets anders behandelt als Dateien.
4. Derzeit wird allen Dateien mit einem Namen, der zum regulären Ausdruck `/gnu/store/.+-(guix-.+|profile)/bin/guix-daemon` passt, die Kennzeichnung `guix_daemon_exec_t` zugewiesen, wodurch *jeder beliebigen* Datei mit diesem Namen in irgendeinem Profil gestattet wäre, in der Domäne `guix_daemon_t` ausgeführt zu werden. Das ist nicht ideal. Ein Angreifer könnte ein Paket erstellen, dass solch eine ausführbare Datei enthält, und den Nutzer überzeugen, es zu installieren und auszuführen. Dadurch käme es in die Domäne `guix_daemon_t`. Ab diesem Punkt könnte SELinux nicht mehr verhindern, dass es auf Dateien zugreift, auf die Prozesse in dieser Domäne zugreifen dürfen.

Nach jeder Aktualisierung des `guix-daemon`, z.B. nachdem Sie `guix pull` ausgeführt haben, müssen Sie das Store-Verzeichnis neu kennzeichnen. Angenommen, der Store befindet sich unter `/gnu`, dann können Sie das mit `restorecon -vR /gnu` bewerkstelligen oder durch andere Mittel, die Ihr Betriebssystem Ihnen zur Verfügung stellt.

Wir könnten zum Zeitpunkt der Installation eine wesentlich restriktivere Richtlinie generieren, für die nur *genau derselbe* Dateiname des gerade installierten `guix-daemon`-Programms als `guix_daemon_exec_t` gekennzeichnet würde, statt einen vieles umfassenden regulären Ausdruck zu benutzen. Aber dann müsste der Administratornutzer zum Zeitpunkt der Installation jedes Mal die Richtlinie installieren oder aktualisieren müssen, sobald das Guix-Paket aktualisiert wird, dass das tatsächlich in Benutzung befindliche `guix-daemon`-Programm enthält.

2.5 Aufruf von `guix-daemon`

Das Programm `guix-daemon` implementiert alle Funktionalitäten, um auf den Store zuzugreifen. Dazu gehört das Starten von Erstellungsprozessen, das Ausführen des Müllsammlers, das Abfragen, ob ein Erstellungsergebnis verfügbar ist, etc. Normalerweise wird er so als Administratornutzer (`root`) gestartet:

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

Sie können den Daemon auch über das systemd-Protokoll zur „Socket-Aktivierung“ starten (siehe Abschnitt “Service De- and Constructors” in *The GNU Shepherd Manual*).

Details, wie Sie ihn einrichten, finden Sie im Abschnitt Abschnitt 2.4 [Den Daemon einrichten], Seite 11.

Standardmäßig führt `guix-daemon` Erstellungsprozesse mit unterschiedlichen UIDs aus, die aus der Erstellungsgruppe stammen, deren Name mit `--build-users-group` übergeben wurde. Außerdem läuft jeder Erstellungsprozess in einer chroot-Umgebung, die nur die Teilmenge des Stores enthält, von der der Erstellungsprozess abhängt, entsprechend seiner Ableitung (siehe Kapitel 9 [Programmierschnittstelle], Seite 108), und ein paar bestimmte Systemverzeichnisse, darunter standardmäßig auch `/dev` und `/dev/pts`. Zudem ist die Erstellungsumgebung auf GNU/Linux eine isolierte Umgebung, d.h. ein *Container*: Nicht nur hat sie ihren eigenen Dateisystembaum, sie hat auch einen separaten Namensraum zum Einhängen von Dateisystemen, ihren eigenen Namensraum für PIDs, für Netzwerke, etc. Dies hilft dabei, reproduzierbare Erstellungen zu garantieren (siehe Abschnitt 6.1 [Funktionalitäten], Seite 44).

Wenn der Daemon im Auftrag des Nutzers eine Erstellung durchführt, erzeugt er ein Erstellungsverzeichnis, entweder in `/tmp` oder im Verzeichnis, das durch die Umgebungsvariable `TMPDIR` angegeben wurde. Dieses Verzeichnis wird mit dem Container geteilt, solange die Erstellung noch läuft, allerdings trägt es im Container stattdessen immer den Namen „/tmp/guix-build-NAME.drv-0“.

Nach Abschluss der Erstellung wird das Erstellungsverzeichnis automatisch entfernt, außer wenn die Erstellung fehlgeschlagen ist und der Client `--keep-failed` angegeben hat (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

Der Daemon lauscht auf Verbindungen und erstellt jeweils einen Unterprozess für jede von einem Client begonnene Sitzung (d.h. von einem der `guix`-Unterbefehle). Der Befehl `guix processes` zeigt Ihnen eine Übersicht solcher Systemaktivitäten; damit werden Ih-

nen alle aktiven Sitzungen und Clients gezeigt. Weitere Informationen finden Sie unter Abschnitt 10.16 [Aufruf von guix processes], Seite 244.

Die folgenden Befehlszeilenoptionen werden unterstützt:

`--build-users-group=`*Gruppe*
: Verwende die Benutzerkonten aus der *Gruppe*, um Erstellungsprozesse auszuführen (siehe Abschnitt 2.4 [Den Daemon einrichten], Seite 11).

`--no-substitutes`
: Benutze keine Substitute für Erstellungsergebnisse. Das heißt, dass alle Objekte lokal erstellt werden müssen, und kein Herunterladen von vorab erstellten Binärdateien erlaubt ist (siehe Abschnitt 6.3 [Substitute], Seite 56).

  Wenn der Daemon mit `--no-substitutes` ausgeführt wird, können Clients trotzdem Substitute explizit aktivieren über den entfernten Prozeduraufruf `set-build-options` (siehe Abschnitt 9.9 [Der Store], Seite 165).

`--substitute-urls=`*URLs*
: *URLs* als standardmäßige, leerzeichengetrennte Liste der Quell-URLs für Substitute benutzen. Wenn diese Befehlszeilenoption *nicht* angegeben wird, wird '`https://ci.guix.gnu.org` `https://bordeaux.guix.gnu.org`' verwendet.

  Das hat zur Folge, dass Substitute von den *URLs* heruntergeladen werden können, solange sie mit einer Signatur versehen sind, der vertraut wird (siehe Abschnitt 6.3 [Substitute], Seite 56).

  Siehe Abschnitt 6.3.3 [Substitute von anderen Servern holen], Seite 57, für weitere Informationen, wie der Daemon konfiguriert werden kann, um Substitute von anderen Servern zu beziehen.

`--no-offload`
: Nicht versuchen, an andere Maschinen ausgelagerte Erstellungen zu benutzen (siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13). Somit wird lokal erstellt, statt Erstellungen auf entfernte Maschinen auszulagern.

`--cache-failures`
: Fehler bei der Erstellung zwischenspeichern. Normalerweise werden nur erfolgreiche Erstellungen gespeichert.

  Wenn diese Befehlszeilenoption benutzt wird, kann `guix gc --list-failures` benutzt werden, um die Menge an Store-Objekten abzufragen, die als Fehlschläge markiert sind; `guix gc --clear-failures` entfernt Store-Objekte aus der Menge zwischengespeicherter Fehlschläge. Siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61.

`--cores=`*n*
`-c` *n*
: *n* CPU-Kerne zum Erstellen jeder Ableitung benutzen; `0` heißt, so viele wie verfügbar sind.

  Der Vorgabewert ist `0`, jeder Client kann jedoch eine abweichende Anzahl vorgeben, zum Beispiel mit der Befehlszeilenoption `--cores` von `guix build` (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189).

  Dadurch wird die Umgebungsvariable `NIX_BUILD_CORES` im Erstellungsprozess definiert, welcher sie benutzen kann, um intern parallele Ausführungen zuzulassen – zum Beispiel durch Nutzung von `make -j$NIX_BUILD_CORES`.

`--max-jobs=`*n*

`-M` *n* Höchstenss *n* Erstellungsaufträge parallel bearbeiten. Der Vorgabewert liegt bei `1`. Wird er auf `0` gesetzt, werden keine Erstellungen lokal durchgeführt, stattdessen lagert der Daemon sie nur aus (siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13) oder sie schlagen einfach fehl.

`--max-silent-time=`*Sekunden*

Wenn der Erstellungs- oder Substitutionsprozess länger als *Sekunden*-lang keine Ausgabe erzeugt, wird er abgebrochen und ein Fehler beim Erstellen gemeldet.

Der Vorgabewert ist `0`, was bedeutet, dass es keine Zeitbeschränkung gibt.

Clients können einen anderen Wert als den hier angegebenen verwenden lassen (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

`--timeout=`*Sekunden*

Entsprechend wird hier der Erstellungs- oder Substitutionsprozess abgebrochen und als Fehlschlag gemeldet, wenn er mehr als *Sekunden*-lang dauert.

Der Vorgabewert ist `0`, was bedeutet, dass es keine Zeitbeschränkung gibt.

Clients können einen anderen Wert verwenden lassen (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

`--rounds=`*N*

Jede Ableitung *n*-mal hintereinander erstellen und einen Fehler melden, wenn nacheinander ausgewertete Erstellungsergebnisse nicht Bit für Bit identisch sind. Beachten Sie, dass Clients wie `guix build` einen anderen Wert verwenden lassen können (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189).

Wenn dies zusammen mit `--keep-failed` benutzt wird, bleiben die sich unterscheidenden Ausgaben im Store unter dem Namen `/gnu/store/...-check`. Dadurch können Unterschiede zwischen den beiden Ergebnissen leicht erkannt werden.

`--debug` Informationen zur Fehlersuche ausgeben.

Dies ist nützlich, um Probleme beim Starten des Daemons nachzuvollziehen; Clients könn aber auch ein abweichenden Wert verwenden lassen, zum Beispiel mit der Befehlszeilenoption `--verbosity` von `guix build` (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189).

`--chroot-directory=`*Verzeichnis*

Füge das *Verzeichnis* zum chroot von Erstellungen hinzu.

Dadurch kann sich das Ergebnis von Erstellungsprozessen ändern – zum Beispiel, wenn diese optionale Abhängigkeiten aus dem *Verzeichnis* verwenden, wenn sie verfügbar sind, und nicht, wenn es fehlt. Deshalb ist es nicht empfohlen, dass Sie diese Befehlszeilenoption verwenden, besser sollten Sie dafür sorgen, dass jede Ableitung alle von ihr benötigten Eingabgen deklariert.

`--disable-chroot`

Erstellungen ohne chroot durchführen.

Diese Befehlszeilenoption zu benutzen, wird nicht empfohlen, denn auch dadurch bekämen Erstellungsprozesse Zugriff auf nicht deklarierte

Abhängigkeiten. Sie ist allerdings unvermeidlich, wenn `guix-daemon` auf einem Benutzerkonto ohne ausreichende Berechtigungen ausgeführt wird.

`--log-compression=`*Typ*
: Erstellungsprotokolle werden entsprechend dem *Typ* komprimiert, der entweder `gzip`, `bzip2` oder `none` (für keine Kompression) sein muss.

  Sofern nicht `--lose-logs` angegeben wurde, werden alle Erstellungsprotokolle in der *localstatedir* gespeichert. Um Platz zu sparen, komprimiert sie der Daemon standardmäßig automatisch mit gzip.

`--discover[=yes|no]`
: Ob im lokalen Netzwerk laufende Substitutserver mit mDNS und DNS-SD ermittelt werden sollen oder nicht.

  Diese Funktionalität ist noch experimentell. Trotzdem sollten Sie bedenken:

  1. Es könnte schneller bzw. günstiger sein, als Substitute von entfernten Servern zu beziehen.
  2. Es gibt keine Sicherheitsrisiken, weil nur echte Substitute benutzt werden können (siehe Abschnitt 6.3.4 [Substitutauthentifizierung], Seite 59).
  3. Wenn ein Angreifer Ihnen sein `guix publish` in Ihrem LAN mitteilt, kann er Ihnen keine bösartigen Programmdateien unterjubeln, aber er kann lernen, welche Software Sie installieren.
  4. Server können Ihnen Substitute über unverschlüsseltes HTTP anbieten, wodurch auch jeder andere in Ihrem LAN vielleicht mitschneiden könnte, welche Software Sie installieren.

  Das Erkennen von Substitutservern können Sie auch nachträglich zur Laufzeit an- oder abschalten („on“ oder „off“), indem Sie dies ausführen:

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

`--disable-deduplication`
: Automatische Dateien-„Deduplizierung“ im Store ausschalten.

  Standardmäßig werden zum Store hinzugefügte Objekte automatisch „dedupliziert“: Wenn eine neue Datei mit einer anderen im Store übereinstimmt, wird die neue Datei stattdessen als harte Verknüpfung auf die andere Datei angelegt. Dies reduziert den Speicherverbrauch auf der Platte merklich, jedoch steigt andererseits die Auslastung bei der Ein-/Ausgabe im Erstellungsprozess geringfügig. Durch diese Option wird keine solche Optimierung durchgeführt.

`--gc-keep-outputs[=yes|no]`
: Gibt an, ob der Müllsammler (Garbage Collector, GC) die Ausgaben lebendiger Ableitungen behalten muss („yes“) oder nicht („no“).

  Für `yes` behält der Müllsammler die Ausgaben aller lebendigen Ableitungen im Store – die `.drv`-Dateien. Der Vorgabewert ist aber `no`, so dass Ableitungsausgaben nur vorgehalten werden, wenn sie von einer Müllsammlerwurzel aus erreichbar sind. Siehe den Abschnitt Abschnitt 6.5 [Aufruf von guix gc], Seite 61, für weitere Informationen zu Müllsammlerwurzeln.

`--gc-keep-derivations[=yes|no]`
: Gibt an, ob der Müllsammler (GC) Ableitungen behalten muss („yes“), wenn sie lebendige Ausgaben haben, oder nicht („no“).

  Für `yes`, den Vorgabewert, behält der Müllsammler Ableitungen – z.B. `.drv`-Dateien –, solange zumindest eine ihrer Ausgaben lebendig ist. Dadurch können Nutzer den Ursprung der Dateien in ihrem Store nachvollziehen. Setzt man den Wert auf `no`, wird ein bisschen weniger Speicher auf der Platte verbraucht.

  Auf diese Weise überträgt sich, wenn `--gc-keep-derivations` auf `yes` steht, die Lebendigkeit von Ausgaben auf Ableitungen, und wenn `--gc-keep-outputs` auf `yes` steht, die Lebendigkeit von Ableitungen auf Ausgaben. Stehen beide auf `yes`, bleiben so alle Erstellungsvoraussetzungen wie Quelldateien, Compiler, Bibliotheken und andere Erstellungswerkzeuge lebendiger Objekte im Store erhalten, ob sie von einer Müllsammlerwurzel aus erreichbar sind oder nicht. Entwickler können sich so erneute Erstellungen oder erneutes Herunterladen sparen.

`--impersonate-linux-2.6`
: Auf Linux-basierten Systemen wird hiermit vorgetäuscht, dass es sich um Linux 2.6 handeln würde, indem der Kernel für einen `uname`-Systemaufruf als Version der Veröffentlichung mit 2.6 antwortet.

  Dies kann hilfreich sein, um Programme zu erstellen, die (normalerweise zu Unrecht) von der Kernel-Versionsnummer abhängen.

`--lose-logs`
: Keine Protokolle der Erstellungen vorhalten. Normalerweise würden solche in *`localstatedir`*`/guix/log` gespeichert.

`--system=`*System*
: Verwende *System* als aktuellen Systemtyp. Standardmäßig ist dies das Paar aus Befehlssatz und Kernel, welches beim Aufruf von `configure` erkannt wurde, wie zum Beispiel `x86_64-linux`.

`--listen=`*Endpunkt*
: Lausche am *Endpunkt* auf Verbindungen. Dabei wird der *Endpunkt* als Dateiname eines Unix-Sockets verstanden, wenn er mit einem `/` (Schrägstrich) beginnt. Andernfalls wird der *Endpunkt* als Rechnername (d.h. Hostname) oder als Rechnername-Port-Paar verstanden, auf dem gelauscht wird. Hier sind ein paar Beispiele:

  `--listen=/gnu/var/daemon`
  : Lausche auf Verbindungen am Unix-Socket `/gnu/var/daemon`, falls nötig wird er dazu erstellt.

  `--listen=localhost`
  : Lausche auf TCP-Verbindungen an der Netzwerkschnittstelle, die `localhost` entspricht, auf Port 44146.

  `--listen=128.0.0.42:1234`
  : Lausche auf TCP-Verbindungen an der Netzwerkschnittstelle, die `128.0.0.42` entspricht, auf Port 1234.

Diese Befehlszeilenoption kann mehrmals wiederholt werden. In diesem Fall akzeptiert `guix-daemon` Verbindungen auf allen angegebenen Endpunkten. Benutzer können bei Client-Befehlen angeben, mit welchem Endpunkt sie sich verbinden möchten, indem sie die Umgebungsvariable `GUIX_DAEMON_SOCKET` festlegen (siehe Abschnitt 9.9 [Der Store], Seite 165).

> **Anmerkung:** Das Daemon-Protokoll ist *weder authentifiziert noch verschlüsselt*. Die Benutzung von `--listen=`*`Rechner`* eignet sich für lokale Netzwerke, wie z.B. in Rechen-Clustern, wo sich nur solche Knoten mit dem Daemon verbinden, denen man vertraut. In Situationen, wo ein Fernzugriff auf den Daemon durchgeführt wird, empfehlen wir, über Unix-Sockets in Verbindung mit SSH zuzugreifen.

Wird `--listen` nicht angegeben, lauscht `guix-daemon` auf Verbindungen auf dem Unix-Socket, der sich unter *`localstatedir`*`/guix/daemon-socket/socket` befindet.

2.6 Anwendungen einrichten

Läuft Guix aufgesetzt auf einer GNU/Linux-Distribution außer Guix System – einer sogenannten *Fremddistribution* –, so sind ein paar zusätzliche Schritte bei der Einrichtung nötig. Hier finden Sie manche davon.

2.6.1 Locales

Über Guix installierte Pakete benutzen nicht die Daten zu Regions- und Spracheinstellungen (Locales) des Wirtssystems. Stattdessen müssen Sie erst eines der Locale-Pakete installieren, die für Guix verfügbar sind, und dann den Wert Ihrer Umgebungsvariablen `GUIX_LOCPATH` passend festlegen:

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

Beachten Sie, dass das Paket `glibc-locales` Daten für alle von GNU libc unterstützten Locales enthält und deswegen um die 930 MiB wiegt[6]. Wenn alles, was Sie brauchen, einige wenige Locales sind, können Sie Ihr eigenes Locale-Paket mit der Prozedur `make-glibc-utf8-locales` aus dem Modul `(gnu packages base)` definieren. Folgendes Beispiel definiert ein Paket mit den verschiedenen kanadischen UTF-8-Locales, die der GNU libc bekannt sind, das nur um die 14 MiB schwer ist:

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

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

[6] Das `glibc-locales`-Paket fällt durch Deduplizierung im Store nur noch mit 213 MiB ins Gewicht und nimmt auf einem zstd-komprimierten Btrfs-Dateisystem dann nur 67 MiB weg.

Die Variable `GUIX_LOCPATH` spielt eine ähnliche Rolle wie `LOCPATH` (siehe Abschnitt "Locale Names" in *Referenzhandbuch der GNU-C-Bibliothek*). Es gibt jedoch zwei wichtige Unterschiede:

1. `GUIX_LOCPATH` wird nur von der libc in Guix beachtet und nicht der von Fremddistributionen bereitgestellten libc. Mit `GUIX_LOCPATH` können Sie daher sicherstellen, dass die Programme der Fremddistribution keine inkompatiblen Locale-Daten von Guix laden.
2. libc hängt an jeden `GUIX_LOCPATH`-Eintrag `/X.Y` an, wobei `X.Y` die Version von libc ist – z.B. `2.22`. Sollte Ihr Guix-Profil eine Mischung aus Programmen enthalten, die an verschiedene libc-Versionen gebunden sind, wird jede nur die Locale-Daten im richtigen Format zu laden versuchen.

Das ist wichtig, weil das Locale-Datenformat verschiedener libc-Versionen inkompatibel sein könnte.

2.6.2 Name Service Switch

Wenn Sie Guix auf einer Fremddistribution verwenden, *empfehlen wir stärkstens*, dass Sie den *Name Service Cache Daemon* der GNU-C-Bibliothek, `nscd`, laufen lassen, welcher auf dem Socket `/var/run/nscd/socket` lauschen sollte. Wenn Sie das nicht tun, könnten mit Guix installierte Anwendungen Probleme beim Auflösen von Hostnamen (d.h. Rechnernamen) oder Benutzerkonten haben, oder sogar abstürzen. Die nächsten Absätze erklären warum.

Die GNU-C-Bibliothek implementiert einen *Name Service Switch* (NSS), welcher einen erweiterbaren Mechanismus zur allgemeinen „Namensauflösung“ darstellt: Hostnamensauflösung, Benutzerkonten und weiteres (siehe Abschnitt "Name Service Switch" in *Referenzhandbuch der GNU-C-Bibliothek*).

Für die Erweiterbarkeit unterstützt der NSS *Plugins*, welche neue Implementierungen zur Namensauflösung bieten: Zum Beispiel ermöglicht das Plugin `nss-mdns` die Namensauflösung für `.local`-Hostnamen, das Plugin `nis` gestattet die Auflösung von Benutzerkonten über den Network Information Service (NIS) und so weiter. Diese zusätzlichen „Auflösungsdienste“ werden systemweit konfiguriert in `/etc/nsswitch.conf` und alle auf dem System laufenden Programme halten sich an diese Einstellungen (siehe Abschnitt "NSS Configuration File" in *GNU-C-Referenzhandbuch*).

Wenn sie eine Namensauflösung durchführen – zum Beispiel, indem sie die `getaddrinfo`-Funktion in C aufrufen –, versuchen die Anwendungen als Erstes, sich mit dem nscd zu verbinden; ist dies erfolgreich, führt nscd für sie die weiteren Namensauflösungen durch. Falls nscd nicht läuft, führen sie selbst die Namensauflösungen durch, indem sie die Namensauflösungsdienste in ihren eigenen Adressraum laden und ausführen. Diese Namensauflösungsdienste – die `libnss_*.so`-Dateien – werden mit `dlopen` geladen, aber sie kommen von der C-Bibliothek des Wirtssystems und nicht von der C-Bibliothek, an die die Anwendung gebunden wurde (also der C-Bibliothek von Guix).

Und hier kommt es zum Problem: Wenn die Anwendung an die C-Bibliothek von Guix (etwa glibc 2.24) gebunden wurde und die NSS-Plugins von einer anderen C-Bibliothek (etwa `libnss_mdns.so` für glibc 2.22) zu laden versucht, wird sie vermutlich abstürzen oder die Namensauflösungen werden unerwartet fehlschlagen.

Durch das Ausführen von `nscd` auf dem System wird, neben anderen Vorteilen, dieses Problem der binären Inkompatibilität vermieden, weil diese `libnss_*.so`-Dateien vom `nscd`-Prozess geladen werden, nicht in den Anwendungen selbst.

2.6.3 X11-Schriftarten

Die Mehrheit der grafischen Anwendungen benutzen Fontconfig zum Finden und Laden von Schriftarten und für die Darstellung im X11-Client. Im Paket `fontconfig` in Guix werden Schriftarten standardmäßig in `$HOME/.guix-profile` gesucht. Um es grafischen Anwendungen, die mit Guix installiert wurden, zu ermöglichen, Schriftarten anzuzeigen, müssen Sie die Schriftarten auch mit Guix installieren. Essenzielle Pakete für Schriftarten sind unter anderem `font-ghostscript`, `font-dejavu` und `font-gnu-freefont`.

Sobald Sie Schriftarten installiert oder wieder entfernt haben oder wenn Ihnen auffällt, dass eine Anwendung Schriftarten nicht finden kann, dann müssen Sie vielleicht Fontconfig installieren und den folgenden Befehl ausführen, damit dessen Zwischenspeicher für Schriftarten aktualisiert wird:

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

Um auf Chinesisch, Japanisch oder Koreanisch verfassten Text in grafischen Anwendungen anzeigen zu können, möchten Sie vielleicht `font-adobe-source-han-sans` oder `font-wqy-zenhei` installieren. Ersteres hat mehrere Ausgaben, für jede Sprachfamilie eine (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60). Zum Beispiel installiert folgender Befehl Schriftarten für chinesische Sprachen:

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

Ältere Programme wie `xterm` benutzen kein Fontconfig, sondern X-Server-seitige Schriftartendarstellung. Solche Programme setzen voraus, dass der volle Name einer Schriftart mit XLFD (X Logical Font Description) angegeben wird, z.B. so:

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

Um solche vollen Namen für die in Ihrem Guix-Profil installierten TrueType-Schriftarten zu verwenden, müssen Sie den Pfad für Schriftarten (Font Path) des X-Servers anpassen:

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

Danach können Sie den Befehl `xlsfonts` ausführen (aus dem Paket `xlsfonts`), um sicherzustellen, dass dort Ihre TrueType-Schriftarten aufgeführt sind.

2.6.4 X.509-Zertifikate

Das Paket `nss-certs` bietet X.509-Zertifikate, womit Programme die Identität von Web-Servern authentifizieren können, auf die über HTTPS zugegriffen wird.

Wenn Sie Guix auf einer Fremddistribution verwenden, können Sie dieses Paket installieren und die relevanten Umgebungsvariablen festlegen, damit Pakete wissen, wo sie Zertifikate finden. Unter Abschnitt 12.11 [X.509-Zertifikate], Seite 606, stehen genaue Informationen.

2.6.5 Emacs-Pakete

Wenn Sie Emacs-Pakete mit Guix installieren, werden die Elisp-Dateien innerhalb des Verzeichnisses `share/emacs/site-lisp/` in demjenigen Profil platziert, wohin sie

installiert werden. Die Elisp-Bibliotheken werden in Emacs über die `EMACSLOADPATH`-Umgebungsvariable verfügbar gemacht, die durch die Installation von Emacs eingerichtet wird.

Bei der Initialisierung von Emacs werden „Autoload“-Definitionen automatisch über die Guix-spezifische Prozedur `guix-emacs-autoload-packages` ausgewertet. Wenn Sie aber aus irgendeinem Grund die mit Guix installierten Pakete nicht automatisch laden lassen möchten, können Sie Emacs mit der Befehlszeilenoption `--no-site-file` starten (siehe Abschnitt “Init File” in *The GNU Emacs Manual*).

> **Anmerkung:** Emacs kann nun native Maschinenbefehle erzeugen. Standardgemäß kompiliert es nun Pakete „just in time“, während Sie diese laden, und platziert die so erzeugten nativen Bibliotheken in einem Unterverzeichnis Ihres `user-emacs-directory`.
>
> Darüber hinaus unterstützt das Erstellungssystem für Emacs-Pakete die Erzeugung nativer Maschinenbefehle. Beachten Sie jedoch, dass `emacs-minimal` – die Emacs-Variante, mit der normalerweise Emacs-Pakete erstellt werden – weiterhin keine nativen Befehle generiert. Um native Befehle für Ihre Emacs-Pakete schon im Voraus zu erzeugen, nutzen Sie eine Transformation, z.B. `--with-input=emacs-minimal=emacs`.

2.7 Aktualisieren von Guix

Um Guix zu aktualisieren, führen Sie aus:

```
guix pull
```

Siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65, für weitere Informationen.

Auf einer Fremddistribution können Sie den Erstellungsdaemon aktualisieren, indem Sie diesen Befehl:

```
sudo -i guix pull
```

gefolgt von diesem ausführen (unter der Annahme, dass Ihre Distribution zur Dienstverwaltung das systemd-Werkzeug benutzt):

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

Auf Guix System wird der Daemon aktualisiert, indem Sie das System rekonfigurieren (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

3 Systeminstallation

Dieser Abschnitt beschreibt, wie Sie „Guix System“ auf einer Maschine installieren. Guix kann auch als Paketverwaltungswerkzeug ein bestehendes GNU/Linux-System ergänzen, mehr dazu finden Sie im Abschnitt Kapitel 2 [Installation], Seite 5.

3.1 Einschränkungen

Wir denken, dass Guix System für viele Anwendungszwecke von Heim- und Bürorechnern bis hin zu Servern geeignet ist. Die Verlässlichkeitsgarantien, die es einem bietet – transaktionelle Aktualisierungen und Rücksetzungen, Reproduzierbarkeit – machen es zu einer soliden Grundlage.

Bevor Sie mit der Installation fortfahren, sollten Sie dennoch die folgenden merklichen Einschränkungen der Version 1.4.0 beachten:

- Immer mehr Systemdienste sind verfügbar (siehe Abschnitt 12.9 [Dienste], Seite 280), aber manche könnten noch fehlen.
- GNOME, Xfce, LXDE und Enlightenment stehen zur Verfügung (siehe Abschnitt 12.9.9 [Desktop-Dienste], Seite 366), ebenso eine Reihe von X11-Fensterverwaltungsprogrammen, allerdings fehlt KDE zurzeit noch.

Dies soll allerdings nicht nur ein Hinweis sein, sondern auch als Einladung aufgefasst werden, uns Fehler (und Erfolgsgeschichten!) zu melden und bei uns mitzumachen, um Guix zu verbessern. Siehe den Abschnitt Kapitel 22 [Mitwirken], Seite 708.

3.2 Hardware-Überlegungen

GNU Guix legt den Fokus darauf, die Freiheit des Nutzers auf seinem Rechner zu respektieren. Es baut auf Linux-libre als Kernel auf, wodurch nur Hardware unterstützt wird, für die Treiber und Firmware existieren, die freie Software sind. Heutzutage wird ein großer Teil der handelsüblichen Hardware von GNU/Linux-libre unterstützt – von Tastaturen bis hin zu Grafikkarten, Scannern und Ethernet-Adaptern. Leider gibt es noch Bereiche, wo die Hardwareanbieter ihren Nutzern die Kontrolle über ihren eigenen Rechner verweigern. Solche Hardware wird von Guix System nicht unterstützt.

Einer der wichtigsten Bereiche, wo es an freien Treibern und freier Firmware mangelt, sind WLAN-Geräte. WLAN-Geräte, von denen wir wissen, dass sie funktionieren, sind unter anderem solche, die die Atheros-Chips AR9271 und AR7010 verbauen, welche der Linux-libre-Treiber `ath9k` unterstützt, und die, die Broadcom/AirForce-Chips BCM43xx (mit Wireless-Core Revision 5) verbauen, welche der Linux-libre-Treiber `b43-open` unterstützt. Freie Firmware gibt es für beide und sie wird von Haus aus mit Guix System als ein Teil von `%base-firmware` mitgeliefert (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258).

Das Installationsprogramm zeigt zu Beginn eine Warnung an, wenn es Geräte erkennt, die bekanntlich *nicht* funktionieren, weil es keine freie Firmware oder keine freien Treiber dafür gibt.

Die Free Software Foundation (`https://www.fsf.org/`) betreibt *Respects Your Freedom* (`https://www.fsf.org/ryf`) (RYF), ein Zertifizierungsprogramm für Hardware-Produkte, die Ihre Freiheit respektieren, Datenschutz gewährleisten und

sicherstellen, dass Sie die Kontrolle über Ihr Gerät haben. Wir ermutigen Sie dazu, die Liste RYF-zertifizierter Geräte zu beachten.

Eine weitere nützliche Ressource ist die Website H-Node (`https://h-node.org/home/index/de`). Dort steht ein Katalog von Hardware-Geräten mit Informationen darüber, wie gut sie von GNU/Linux unterstützt werden.

3.3 Installation von USB-Stick oder DVD

Sie können ein ISO-9660-Installationsabbild von ‘`https://ftp.gnu.org/gnu/guix/guix-system-install-1.` herunterladen, dass Sie auf einen USB-Stick aufspielen oder auf eine DVD brennen können, wobei Sie anstelle von `x86_64-linux` eines der folgenden schreiben können:

`x86_64-linux`
: für ein GNU/Linux-System auf Intel/AMD-kompatiblen 64-Bit-Prozessoren,

`i686-linux`
: für ein 32-Bit-GNU/Linux-System auf Intel-kompatiblen Prozessoren.

Laden Sie auch die entsprechende `.sig`-Datei herunter und verifizieren Sie damit die Authentizität Ihres Abbilds, indem Sie diese Befehle eingeben:

```
$ wget https://ftp.gnu.org/gnu/guix/guix-system-install-1.4.0.x86_64-linux.iso.sig
$ gpg --verify guix-system-install-1.4.0.x86_64-linux.iso.sig
```

Falls dieser Befehl fehlschlägt, weil Sie nicht über den nötigen öffentlichen Schlüssel verfügen, können Sie ihn mit diesem Befehl importieren:

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

und den Befehl `gpg --verify` erneut ausführen.

Beachten Sie, dass eine Warnung wie „Dieser Schlüssel trägt keine vertrauenswürdige Signatur!“ normal ist.

Dieses Abbild enthält die Werkzeuge, die Sie zur Installation brauchen. Es ist dafür gedacht, *so wie es ist* auf einen hinreichend großen USB-Stick oder eine DVD kopiert zu werden.

Kopieren auf einen USB-Stick

Stecken Sie einen USB-Stick in Ihren Rechner ein, der mindestens 1 GiB groß ist, und bestimmen Sie seinen Gerätenamen. Ist der Gerätename des USB-Sticks `/dev/sdX`, dann kopieren Sie das Abbild mit dem Befehl:

```
dd if=guix-system-install-1.4.0.x86_64-linux.iso of=/dev/sdX status=progress
sync
```

Sie benötigen in der Regel Administratorrechte, um auf `/dev/sdX` zuzugreifen.

Auf eine DVD brennen

Legen Sie eine unbespielte DVD in Ihren Rechner ein und bestimmen Sie ihren Gerätenamen. Angenommen der Name des DVD-Laufwerks ist `/dev/srX`, kopieren Sie das Abbild mit:

```
growisofs -dvd-compat -Z /dev/srX=guix-system-install-1.4.0.x86_64-linux.iso
```

Der Zugriff auf `/dev/srX` setzt in der Regel Administratorrechte voraus.

Das System starten

Sobald das erledigt ist, sollten Sie Ihr System neu starten und es vom USB-Stick oder der DVD hochfahren („booten“) können. Dazu müssen Sie wahrscheinlich beim Starten des Rechners in das BIOS- oder UEFI-Boot-Menü gehen, von wo aus Sie auswählen können, dass vom USB-Stick gebootet werden soll. Um aus Libreboot heraus zu booten, wechseln Sie in den Befehlsmodus, indem Sie die `c`-Taste drücken, und geben Sie `search_grub usb` ein.

Lesen Sie den Abschnitt Abschnitt 3.8 [Guix in einer VM installieren], Seite 37, wenn Sie Guix System stattdessen in einer virtuellen Maschine (VM) installieren möchten.

3.4 Vor der Installation

Wenn Sie Ihren Rechner gebootet haben, können Sie sich vom grafischen Installationsprogramm durch den Installationsvorgang führen lassen, was den Einstieg leicht macht (siehe Abschnitt 3.5 [Geführte grafische Installation], Seite 29). Alternativ können Sie sich auch für einen „manuellen“ Installationsvorgang entscheiden, wenn Sie bereits mit GNU/Linux vertraut sind und mehr Kontrolle haben möchten, als sie das grafische Installationsprogramm bietet (siehe Abschnitt 3.6 [Manuelle Installation], Seite 31).

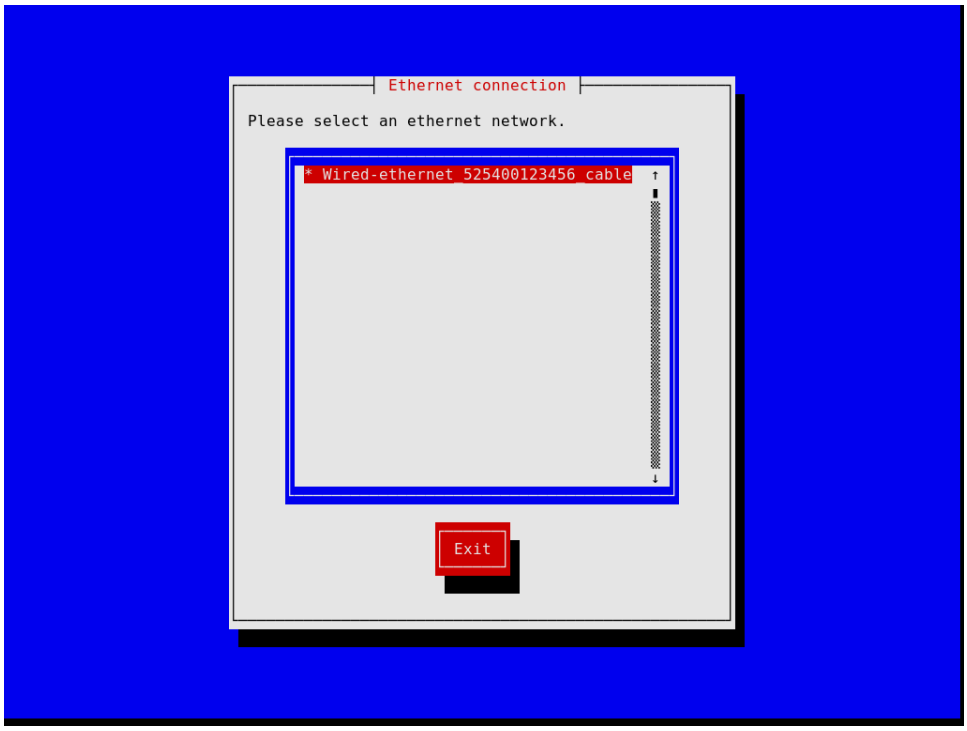
Das grafische Installationsprogramm steht Ihnen auf TTY1 zur Verfügung. Auf den TTYs 3 bis 6 können Sie vor sich eine Eingabeaufforderung für den Administratornutzer „root“ sehen, nachdem Sie *`strg-alt-f3`*, *`strg-alt-f4`* usw. gedrückt haben. TTY2 zeigt Ihnen dieses Handbuch, das Sie über die Tastenkombination *`strg-alt-f2`* erreichen. In dieser Dokumentation können Sie mit den Steuerungsbefehlen Ihres Info-Betrachters blättern (siehe *Stand-alone GNU Info*). Auf dem Installationssystem läuft der GPM-Maus-Daemon, wodurch Sie Text mit der linken Maustaste markieren und ihn mit der mittleren Maustaste einfügen können.

> **Anmerkung:** Für die Installation benötigen Sie Zugang zum Internet, damit fehlende Abhängigkeiten Ihrer Systemkonfiguration heruntergeladen werden können. Im Abschnitt „Netzwerkkonfiguration“ weiter unten finden Sie mehr Informationen dazu.

3.5 Geführte grafische Installation

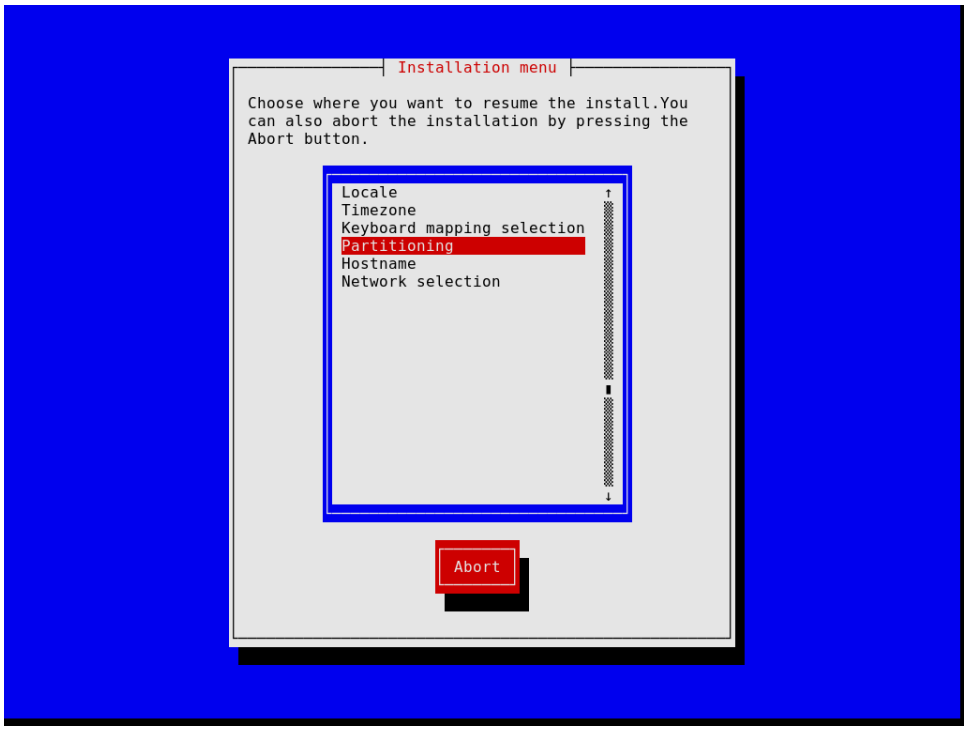
Das grafische Installationsprogramm ist mit einer textbasierten Benutzeroberfläche ausgestattet. Es kann Sie mit Dialogfeldern durch die Schritte führen, mit denen Sie GNU Guix System installieren.

Die ersten Dialogfelder ermöglichen es Ihnen, das System aufzusetzen, wie Sie es bei der Installation benutzen: Sie können die Sprache und Tastaturbelegung festlegen und die Netzwerkanbindung einrichten, die während der Installation benutzt wird. Das folgende Bild zeigt den Dialog zur Einrichtung der Netzwerkanbindung.



Mit den danach kommenden Schritten können Sie Ihre Festplatte partitionieren, wie im folgenden Bild gezeigt, und auswählen, ob Ihre Dateisysteme verschlüsselt werden sollen oder nicht. Sie können Ihren Rechnernamen und das Administratorpasswort (das „root“-Passwort) festlegen und ein Benutzerkonto einrichten, und noch mehr.

Beachten Sie, dass Sie mit dem Installationsprogramm jederzeit den aktuellen Installationsschritt verlassen und zu einem vorherigen Schritt zurückkehren können, wie Sie im folgenden Bild sehen können.



Sobald Sie fertig sind, erzeugt das Installationsprogramm eine Betriebssystemkonfiguration und zeigt sie an (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250). Zu diesem Zeitpunkt können Sie auf „OK“ drücken und die Installation wird losgehen. Ist sie erfolgreich, können Sie neu starten und Ihr neues System genießen. Siehe Abschnitt 3.7 [Nach der Systeminstallation], Seite 37, für Informationen, wie es weitergeht!

3.6 Manuelle Installation

Dieser Abschnitt beschreibt, wie Sie GNU Guix System auf manuelle Weise auf Ihrer Maschine installieren. Diese Alternative setzt voraus, dass Sie bereits mit GNU/Linux, der Shell und üblichen Administrationswerkzeugen vertraut sind. Wenn Sie glauben, dass das nichts für Sie ist, dann möchten Sie vielleicht das geführte grafische Installationsprogramm benutzen (siehe Abschnitt 3.5 [Geführte grafische Installation], Seite 29).

Das Installationssystem macht Eingabeaufforderungen auf den TTYs 3 bis 6 zugänglich, auf denen Sie als Administratornutzer Befehle eingeben können; Sie erreichen diese, indem Sie die Tastenkombinationen *`strg-alt-f3`*, *`strg-alt-f4`* und so weiter benutzen. Es enthält viele übliche Werkzeuge, mit denen Sie diese Aufgabe bewältigen können. Da es sich auch um ein vollständiges „Guix System“-System handelt, können Sie aber auch andere Pakete mit dem Befehl `guix package` nachinstallieren, wenn Sie sie brauchen (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

3.6.1 Tastaturbelegung, Netzwerkanbindung und Partitionierung

Bevor Sie das System installieren können, wollen Sie vielleicht die Tastaturbelegung ändern, eine Netzwerkverbindung herstellen und die Zielfestplatte partitionieren. Dieser Abschnitt wird Sie durch diese Schritte führen.

3.6.1.1 Tastaturbelegung

Das Installationsabbild verwendet die US-amerikanische QWERTY-Tastaturbelegung. Wenn Sie dies ändern möchten, können Sie den `loadkeys`-Befehl benutzen. Mit folgendem Befehl würden Sie zum Beispiel die Dvorak-Tastaturbelegung auswählen:

```
loadkeys dvorak
```

Schauen Sie sich an, welche Dateien im Verzeichnis `/run/current-system/profile/share/keymaps` stehen, um eine Liste verfügbarer Tastaturbelegungen zu sehen. Wenn Sie mehr Informationen brauchen, führen Sie `man loadkeys` aus.

3.6.1.2 Netzwerkkonfiguration

Führen Sie folgenden Befehl aus, um zu sehen, wie Ihre Netzwerkschnittstellen benannt sind:

```
ifconfig -a
```

... oder mit dem GNU/Linux-eigenen `ip`-Befehl:

```
ip address
```

Der Name kabelgebundener Schnittstellen (engl. Interfaces) beginnt mit dem Buchstaben '`e`', zum Beispiel heißt die dem ersten fest eingebauten Ethernet-Adapter entsprechende Schnittstelle '`eno1`'. Drahtlose Schnittstellen werden mit einem Namen bezeichnet, der mit dem Buchstaben '`w`' beginnt, etwa '`w1p2s0`'.

Kabelverbindung
: Um ein kabelgebundenes Netzwerk einzurichten, führen Sie den folgenden Befehl aus, wobei Sie statt *Schnittstelle* den Namen der kabelgebundenen Schnittstelle eintippen, die Sie benutzen möchten.

  ```
  ifconfig Schnittstelle up
  ```

  ... oder mit dem GNU/Linux-eigenen `ip`-Befehl:

  ```
  ip link set Schnittstelle up
  ```

Drahtlose Verbindung
: Um Drahtlosnetzwerke einzurichten, können Sie eine Konfigurationsdatei für das Konfigurationswerkzeug des `wpa_supplicant` schreiben (wo Sie sie speichern, ist nicht wichtig), indem Sie eines der verfügbaren Textbearbeitungsprogramme wie etwa `nano` benutzen:

  ```
  nano wpa_supplicant.conf
  ```

  Zum Beispiel können Sie die folgende Formulierung in der Datei speichern, die für viele Drahtlosnetzwerke funktioniert, sofern Sie die richtige SSID und Passphrase für das Netzwerk eingeben, mit dem Sie sich verbinden möchten:

  ```
  network={
    ssid="meine-ssid"
    key_mgmt=WPA-PSK
  ```

```
  psk="geheime Passphrase des Netzwerks"
}
```

Starten Sie den Dienst für Drahtlosnetzwerke und lassen Sie ihn im Hintergrund laufen, indem Sie folgenden Befehl eintippen (ersetzen Sie dabei *Schnittstelle* durch den Namen der Netzwerkschnittstelle, die Sie benutzen möchten):

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

Führen Sie `man wpa_supplicant` aus, um mehr Informationen zu erhalten.

Zu diesem Zeitpunkt müssen Sie sich eine IP-Adresse beschaffen. Auf einem Netzwerk, wo IP-Adressen automatisch *via* DHCP zugewiesen werden, können Sie das hier ausführen:

```
dhclient -v Schnittstelle
```

Versuchen Sie, einen Server zu pingen, um zu prüfen, ob sie mit dem Internet verbunden sind und alles richtig funktioniert:

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

Einen Internetzugang herzustellen, ist in jedem Fall nötig, weil das Abbild nicht alle Software und Werkzeuge enthält, die nötig sein könnten.

Wenn HTTP- und HTTPS-Zugriffe bei Ihnen über einen Proxy laufen sollen, führen Sie folgenden Befehl aus:

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

Dabei ist *URL* die URL des Proxys, zum Beispiel `http://example.org:8118`.

Wenn Sie möchten, können Sie die weitere Installation auch per Fernwartung durchführen, indem Sie einen SSH-Server starten:

```
herd start ssh-daemon
```

Vergewissern Sie sich vorher, dass Sie entweder ein Passwort mit `passwd` festgelegt haben, oder dass Sie für OpenSSH eine Authentifizierung über öffentliche Schlüssel eingerichtet haben, bevor Sie sich anmelden.

3.6.1.3 Plattenpartitionierung

Sofern nicht bereits geschehen, ist der nächste Schritt, zu partitionieren und dann die Zielpartition zu formatieren.

Auf dem Installationsabbild sind mehrere Partitionierungswerkzeuge zu finden, einschließlich (siehe Abschnitt “Overview” in *GNU Parted User Manual*), `fdisk` und `cfdisk`. Starten Sie eines davon und partitionieren Sie Ihre Festplatte oder sonstigen Massenspeicher, wie Sie möchten:

```
cfdisk
```

Wenn Ihre Platte mit einer „GUID Partition Table“ (GPT) formatiert ist, und Sie vorhaben, die BIOS-basierte Variante des GRUB-Bootloaders zu installieren (was der Vorgabe entspricht), stellen Sie sicher, dass eine Partition als BIOS-Boot-Partition ausgewiesen ist (siehe Abschnitt “BIOS installation” in *GNU GRUB manual*).

Falls Sie stattdessen einen EFI-basierten GRUB installieren möchten, muss auf der Platte eine FAT32-formatierte *EFI-Systempartition* (ESP) vorhanden sein. Diese Partition kann unter dem Pfad `/boot/efi` eingebunden („gemountet“) werden und die `esp`-Flag der Partition muss gesetzt sein. Dazu würden Sie beispielsweise in `parted` eintippen:

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

Anmerkung:

Falls Sie nicht wissen, ob Sie einen EFI- oder BIOS-basierten GRUB installieren möchten: Wenn bei Ihnen das Verzeichnis `/sys/firmware/efi` im Dateisystem existiert, möchten Sie vermutlich eine EFI-Installation durchführen, wozu Sie in Ihrer Konfiguration `grub-efi-bootloader` benutzen. Ansonsten sollten Sie den BIOS-basierten GRUB benutzen, der mit `grub-bootloader` bezeichnet wird. Siehe Abschnitt 12.14 [Bootloader-Konfiguration], Seite 613, wenn Sie mehr Informationen über Bootloader brauchen.

Sobald Sie die Platte fertig partitioniert haben, auf die Sie installieren möchten, müssen Sie ein Dateisystem auf Ihrer oder Ihren für Guix System vorgesehenen Partition(en) erzeugen[1]. Wenn Sie eine ESP brauchen und dafür die Partition `/dev/sda1` vorgesehen haben, müssen Sie diesen Befehl ausführen:

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

Für das Wurzeldateisystem ist ext4 das am häufigsten genutzte Format. Andere Dateisysteme, wie Btrfs, unterstützen Kompression, wovon berichtet wird, es habe sich als sinnvolle Ergänzung zur Dateideduplikation erwiesen, die der Daemon unabhängig vom Dateisystem bietet (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

Geben Sie Ihren Dateisystemen auch besser eine Bezeichnung („Label“), damit Sie sie zuverlässig wiedererkennen und später in den `file-system`-Deklarationen darauf Bezug nehmen können (siehe Abschnitt 12.3 [Dateisysteme], Seite 263). Dazu benutzen Sie typischerweise die Befehlszeilenoption `-L` des Befehls `mkfs.ext4` oder entsprechende Optionen für andere Befehle. Wenn wir also annehmen, dass `/dev/sda2` die Partition ist, auf der Ihr Wurzeldateisystem (englisch „root“) wohnen soll, können Sie dort mit diesem Befehl ein Dateisystem mit der Bezeichnung `my-root` erstellen:

```
mkfs.ext4 -L my-root /dev/sda2
```

Falls Sie aber vorhaben, die Partition mit dem Wurzeldateisystem zu verschlüsseln, können Sie dazu die Cryptsetup-/LUKS-Werkzeuge verwenden (siehe `man cryptsetup`, um mehr darüber zu erfahren).

Warnung: GRUB kann seit Version 2.06 LUKS2-Geräte entsperren, aber nur die Schlüsselableitungsfunktion PBKDF2 wird unterstützt, was *nicht* die Voreinstellung von `cryptsetup luksFormat` ist. Sie können die eingesetzte Schlüsselableitungsfunktion eines Geräts anzeigen lassen, indem Sie `cryptsetup luksDump` *`Gerät`* ausführen, und nachsehen, ob dort ein PBKDF-Feld in Ihren Schlüsselfächern („Keyslots“) zu finden ist.

Angenommen Sie wollen die Partition für das Wurzeldateisystem verschlüsselt auf `/dev/sda2` installieren, dann brauchen Sie eine Befehlsfolge ähnlich wie diese, um sie als LUKS2-Partition zu formatieren:

```
cryptsetup luksFormat --type luks2 --pbkdf pbkdf2 /dev/sda2
cryptsetup open /dev/sda2 my-partition
mkfs.ext4 -L my-root /dev/mapper/my-partition
```

[1] Derzeit unterstützt Guix System nur die Dateisystemtypen ext4, btrfs, JFS, F2FS und XFS. Insbesondere funktioniert Guix-Code, der Dateisystem-UUIDs und -Bezeichnungen ausliest, nur auf diesen Dateisystemtypen.

Sobald das erledigt ist, binden Sie dieses Dateisystem als Installationsziel mit dem Einhängepunkt `/mnt` ein, wozu Sie einen Befehl wie hier eintippen (auch hier unter der Annahme, dass `my-root` die Bezeichnung des künftigen Wurzeldateisystems ist):

```
mount LABEL=my-root /mnt
```

Binden Sie auch alle anderen Dateisysteme ein, die Sie auf dem Zielsystem benutzen möchten, mit Einhängepunkten relativ zu diesem Pfad. Wenn Sie sich zum Beispiel für einen Einhängepunkt `/boot/efi` für die EFI-Systempartition entschieden haben, binden Sie sie jetzt als `/mnt/boot/efi` ein, damit `guix system init` sie später findet.

Wenn Sie zudem auch vorhaben, eine oder mehrere Swap-Partitionen zu benutzen (siehe Abschnitt 12.5 [Swap-Speicher], Seite 271), initialisieren Sie diese nun mit `mkswap`. Angenommen Sie haben eine Swap-Partition auf `/dev/sda3`, dann würde der Befehl so lauten:

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

Alternativ können Sie eine Swap-Datei benutzen. Angenommen, Sie wollten die Datei `/swapdatei` im neuen System als eine Swapdatei benutzen, dann müssten Sie Folgendes ausführen[2]:

```
# Das bedeutet 10 GiB Swapspeicher. "count" anpassen zum ändern.
dd if=/dev/zero of=/mnt/swapfile bs=1MiB count=10240
# Zur Sicherheit darf nur der Administrator lesen und schreiben.
chmod 600 /mnt/swapfile
mkswap /mnt/swapfile
swapon /mnt/swapfile
```

Bedenken Sie, dass, wenn Sie die Partition für das Wurzeldateisystem („root“) verschlüsselt und eine Swap-Datei in diesem Dateisystem wie oben beschrieben erstellt haben, die Verschlüsselung auch die Swap-Datei schützt, genau wie jede andere Datei in dem Dateisystem.

3.6.2 Fortfahren mit der Installation

Wenn die Partitionen des Installationsziels bereit sind und dessen Wurzeldateisystem unter `/mnt` eingebunden wurde, kann es losgehen mit der Installation. Führen Sie zuerst aus:

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

Dadurch wird `/gnu/store` copy-on-write, d.h. dorthin von Guix erstellte Pakete werden in ihrer Installationsphase auf dem unter `/mnt` befindlichen Zieldateisystem gespeichert, statt den Arbeitsspeicher auszulasten. Das ist nötig, weil die erste Phase des Befehls `guix system init` (siehe unten) viele Dateien nach `/gnu/store` herunterlädt oder sie erstellt, Änderungen am `/gnu/store` aber bis dahin wie das übrige Installationssystem nur im Arbeitsspeicher gelagert werden konnten.

Als Nächstes müssen Sie eine Datei bearbeiten und dort eine Deklaration des Betriebssystems, das Sie installieren möchten, hineinschreiben. Zu diesem Zweck sind im Installationssystem drei Texteditoren enthalten. Wir empfehlen, dass Sie GNU nano benutzen (siehe *GNU nano Manual*), welcher Syntax und zueinander gehörende Klammern hervorheben

[2] Dieses Beispiel wird auf vielen Arten von Dateisystemen funktionieren (z.B. auf ext4). Auf Dateisystemen mit Copy-on-Write (wie z.B. btrfs) können sich die nötigen Schritte unterscheiden. Details finden Sie in der Dokumentation auf den Handbuchseiten von `mkswap` und `swapon`.

kann. Andere mitgelieferte Texteditoren, die Sie benutzen können, sind mg (ein Emacs-Klon) und nvi (ein Klon des ursprünglichen `vi`-Editors von BSD). Wir empfehlen sehr, dass Sie diese Datei im Zieldateisystem der Installation speichern, etwa als `/mnt/etc/config.scm`, weil Sie Ihre Konfigurationsdatei im frisch installierten System noch brauchen werden.

Der Abschnitt Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250, gibt einen Überblick über die Konfigurationsdatei. Die in dem Abschnitt diskutierten Beispielkonfigurationen sind im Installationsabbild im Verzeichnis `/etc/configuration` zu finden. Um also mit einer Systemkonfiguration anzufangen, die einen grafischen Anzeigeserver (einen „Display-Server“ zum Darstellen einer „Desktop“-Arbeitsumgebung) bietet, könnten Sie so etwas ausführen:

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

Achten Sie darauf, was in Ihrer Konfigurationsdatei steht, und besonders auf Folgendes:

- Ihre `bootloader-configuration`-Form muss sich auf die Ziele beziehen, auf die Sie GRUB installieren möchten. Sie sollte genau dann `grub-bootloader` nennen, wenn Sie GRUB im alten BIOS-Modus installieren, und für neuere UEFI-Systeme sollten Sie `grub-efi-bootloader` nennen. Bei Altsystemen bezeichnet das `targets`-Feld die Geräte wie `(list "/dev/sda")`, bei UEFI-Systemen bezeichnet es die Pfade zu eingebundenen EFI-Partitionen (ESP) wie `(list "/boot/efi")`; stellen Sie sicher, dass die Pfade tatsächlich dort eingebunden sind und ein `file-system`-Eintrag dafür in Ihrer Konfiguration festgelegt wurde.
- Dateisystembezeichnungen müssen mit den jeweiligen `device`-Feldern in Ihrer `file-system`-Konfiguration übereinstimmen, sofern Sie in Ihrer `file-system`-Konfiguration die Prozedur `file-system-label` für ihre `device`-Felder benutzen.
- Gibt es verschlüsselte Partitionen oder RAID-Partitionen, dann müssen sie im `mapped-devices`-Feld genannt werden (siehe Abschnitt 12.4 [Zugeordnete Geräte], Seite 269).

Wenn Sie damit fertig sind, Ihre Konfigurationsdatei vorzubereiten, können Sie das neue System initialisieren (denken Sie daran, dass zukünftige Wurzeldateisystem muss unter `/mnt` wie bereits beschrieben eingebunden sein):

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

Dies kopiert alle notwendigen Dateien und installiert GRUB auf `/dev/sdX`, sofern Sie nicht noch die Befehlszeilenoption `--no-bootloader` benutzen. Weitere Informationen finden Sie im Abschnitt Abschnitt 12.15 [Aufruf von guix system], Seite 619. Der Befehl kann das Herunterladen oder Erstellen fehlender Softwarepakete auslösen, was einige Zeit in Anspruch nehmen kann.

Sobald der Befehl erfolgreich – hoffentlich! – durchgelaufen ist, können Sie mit dem Befehl `reboot` das neue System booten lassen. Der Administratornutzer `root` hat im neuen System zunächst ein leeres Passwort, und Passwörter der anderen Nutzer müssen Sie später setzen, indem Sie den Befehl `passwd` als `root` ausführen, außer Ihre Konfiguration enthält schon Passwörter (siehe [user-account-password], Seite 275). Siehe Abschnitt 3.7 [Nach der Systeminstallation], Seite 37, für Informationen, wie es weiter geht!

3.7 Nach der Systeminstallation

Sie haben es geschafft: Sie haben Guix System erfolgreich gebootet! Von jetzt an können Sie Guix System aktualisieren, wann Sie möchten, indem Sie zum Beispiel das hier ausführen:

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

Dadurch wird eine neue Systemgeneration aus den neuesten Paketen und Diensten erstellt (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619). Wir empfehlen, diese Schritte regelmäßig zu wiederholen, damit Ihr System die aktuellen Sicherheitsaktualisierungen benutzt (siehe Kapitel 19 [Sicherheitsaktualisierungen], Seite 697).

Anmerkung:

Beachten Sie, dass bei Nutzung von `sudo guix` der `guix`-Befehl des aktiven Benutzers ausgeführt wird und *nicht* der des Administratornutzers „root“, weil `sudo` die Umgebungsvariable `PATH` unverändert lässt. Um ausdrücklich das `guix`-Programm des Administrators aufzurufen, müssen Sie `sudo -i guix ...` eintippen.

Das macht hier einen Unterschied, weil `guix pull` den `guix`-Befehl und Paketdefinitionen nur für dasjenige Benutzerkonto aktualisiert, mit dem es ausgeführt wird. Würden Sie stattdessen in der Login-Shell des „root“-Nutzers `guix system reconfigure` ausführen, müssten Sie auch für ihn `guix pull` ausführen.

Werfen Sie nun einen Blick auf Kapitel 5 [Einstieg in Guix], Seite 41, und besuchen Sie uns auf `#guix` auf dem Libera-Chat-IRC-Netzwerk oder auf der Mailing-Liste `guix-devel@gnu.org`, um uns Rückmeldung zu geben!

3.8 Guix in einer virtuellen Maschine installieren

Wenn Sie Guix System auf einer virtuellen Maschine (VM) oder einem „Virtual Private Server“ (VPS) statt auf Ihrer echten Maschine installieren möchten, ist dieser Abschnitt hier richtig für Sie.

Um eine virtuelle Maschine für QEMU (`https://qemu.org/`) aufzusetzen, mit der Sie Guix System in ein „Disk-Image“ installieren können (also in eine Datei mit einem Abbild eines Plattenspeichers), gehen Sie so vor:

1. Zunächst laden Sie das Installationsabbild des Guix-Systems wie zuvor beschrieben herunter und entpacken es (siehe Abschnitt 3.3 [Installation von USB-Stick oder DVD], Seite 28).
2. Legen Sie nun ein Disk-Image an, das das System nach der Installation enthalten soll. Um ein qcow2-formatiertes Disk-Image zu erstellen, benutzen Sie den Befehl `qemu-img`:

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

   Die Datei, die Sie herausbekommen, wird wesentlich kleiner als 50 GB sein (typischerweise kleiner als 1 MB), vergrößert sich aber, wenn der virtualisierte Speicher gefüllt wird.
3. Starten Sie das USB-Installationsabbild auf einer virtuellen Maschine:

   ```
   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,file=guix-system-install-1.4.0.System.iso
```

`-enable-kvm` ist optional, verbessert die Rechenleistung aber erheblich, siehe Abschnitt 12.17 [Guix in einer VM starten], Seite 633.

4. Sie sind nun in der virtuellen Maschine als Administratornutzer `root` angemeldet und können mit der Installation wie gewohnt fortfahren. Folgen Sie der Anleitung im Abschnitt Abschnitt 3.4 [Vor der Installation], Seite 29.

Wurde die Installation abgeschlossen, können Sie das System starten, das sich nun als Abbild in der Datei `guix-system.img` befindet. Der Abschnitt Abschnitt 12.17 [Guix in einer VM starten], Seite 633, erklärt, wie Sie das tun können.

3.9 Ein Abbild zur Installation erstellen

Das oben beschriebene Installationsabbild wurde mit dem Befehl `guix system` erstellt, genauer gesagt mit:

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

Die Datei `gnu/system/install.scm` finden Sie im Quellbaum von Guix. Schauen Sie sich die Datei und auch den Abschnitt Abschnitt 12.15 [Aufruf von guix system], Seite 619, an, um mehr Informationen über das Installationsabbild zu erhalten.

3.10 Abbild zur Installation für ARM-Rechner erstellen

Viele ARM-Chips funktionieren nur mit ihrer eigenen speziellen Variante des U-Boot-Bootloaders (`https://www.denx.de/wiki/U-Boot/`).

Wenn Sie ein Disk-Image erstellen und der Bootloader nicht anderweitig schon installiert ist (auf einem anderen Laufwerk), ist es ratsam, ein Disk-Image zu erstellen, was den Bootloader enthält, mit dem Befehl:

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

`A20-OLinuXino-Lime2` ist der Name des Chips. Wenn Sie einen ungültigen Namen eingeben, wird eine Liste möglicher Chip-Namen ausgegeben.

4 Problembehandlung bei Guix System

Guix System macht möglich, dass Sie einfach eine alte Systemgeneration hochfahren, wenn sich die neueste als kaputt herausstellt. Dadurch ist Guix System gegenüber unwiederbringlichem Gebrauchsverlust recht robust aufgestellt. Das setzt allerdings voraus, dass GRUB richtig läuft. Wenn aus irgendeinem Grund Ihre GRUB-Installation bei einer Rekonfiguration des Systems gestört wird, sind Sie ausgesperrt und nicht mehr in der Lage, den Rechner ohne Handgriffe in eine alte Generation zu starten. Für solche Fälle schafft eine andere Technik Abhilfe: Mit `chroot` kommen Sie noch in Ihr System hinein und können es mit `reconfigure` retten. Wie das geht, wird hier erklärt.

4.1 Chroot ins vorliegende System

In diesem Abschnitt wird erklärt, wie Sie mit `chroot` in ein kaputtes Guix System hineinkommen, um es durch `reconfigure` auf eine gültige Konfiguration zu bringen, etwa wenn GRUB beim letzten Mal falsch installiert wurde. Auf anderen GNU/Linux-Systemen würde man ähnlich vorgehen, aber Guix System ist in mancher Hinsicht anders, insbesondere muss der Daemon laufen und Sie müssen Ihre Profile erreichen. Darauf geht diese Erklärung ein.

1. Beschaffen Sie ein bootfähiges Abbild von Guix System. Das neueste Entwicklungs-Image wäre geeignet, weil darin der Kernel und die Werkzeuge mindestens so neu sind wie im installierten System. Sie können es von der URL 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`) bekommen. Im Abschnitt Abschnitt 3.3 [Installation von USB-Stick oder DVD], Seite 28, wird erklärt, wie Sie das Image auf ein bootfähiges Medium überspielen.
2. Fahren Sie vom Medium mit dem Abbild den Rechner hoch und machen Sie die ersten Schritte im grafischen textbasierten Installationsprogramm, bis die Netzwerkverbindung eingerichtet ist. Alternativ können Sie die Netzwerkverbindung manuell einrichten wie es im Abschnitt [manual-installation-networking], Seite 32, beschrieben ist. Sofern Sie den Fehler '`RTNETLINK answers: Operation not possible due to RF-kill`' zu sehen bekommen, hilft vielleicht '`rfkill list`' gefolgt von '`rfkill unblock 0`', wobei '`0`' Ihr Geräteidentifikator (ID) ist.
3. Wechseln Sie zu einer virtuellen Konsole (TTY), falls noch nicht geschehen, indem Sie die Tasten *`Steuerungstaste`* + *`Alt`* + *`F4`* gleichzeitig drücken. Binden Sie Ihr Dateisystem unter `/mnt` ein. Angenommen Sie haben Ihr Wurzeldateisystem auf der Partition `/dev/sda2`, dann geht das mit diesen Befehlen:

   ```
   mount /dev/sda2 /mnt
   ```
4. Binden Sie die blockorientierten Spezialgeräte und Linux-eigene Verzeichnisse ein:

   ```
   mount --bind /proc /mnt/proc
   mount --bind /sys /mnt/sys
   mount --bind /dev /mnt/dev
   ```

   Sofern Ihr System EFI benutzt, müssen Sie auch die als ESP dienende Partition einbinden. Wenn Sie bei Ihnen `/dev/sda1` heißt, lautet der Befehl:

   ```
   mount /dev/sda1 /mnt/boot/efi
   ```
5. Nun betreten Sie Ihr System als eine `chroot`-Umgebung:

   ```
   chroot /mnt /bin/sh
   ```

6. Binden Sie das Systemprofil sowie das Profil von Ihrem *benutzer* ein, damit die Umgebung vollständig ist. Mit *benutzer* ist hier der Benutzername gemeint, den Sie auf dem Guix System haben, das Sie zu reparieren versuchen:

   ```
   source /var/guix/profiles/system/profile/etc/profile
   source /home/benutzer/.guix-profile/etc/profile
   ```

   Damit Sie auch dieselbe Guix-Version zur Verfügung haben, die Sie normalerweise in Ihrem Benutzerkonto haben, laden Sie mit `source` auch Ihr aktuelles Guix-Profil:

   ```
   source /home/benutzer/.config/guix/current/etc/profile
   ```

7. Starten Sie einen minimal funktionsfähigen `guix-daemon` als Hintergrundprozess:

   ```
   guix-daemon --build-users-group=guixbuild --disable-chroot &
   ```

8. Wenn es nötig ist, ändern Sie die Konfigurationsdatei Ihres Guix System, damit sie wieder stimmt. Dann rekonfigurieren Sie etwa so:

   ```
   guix system reconfigure Ihre-config.scm
   ```

9. Endlich sollte Ihr System so weit sein, dass Sie neustarten können und ausprobieren, ob Sie es so in Ordnung gebracht haben.

5 Einstieg in Guix

Wenn Sie in diesem Abschnitt angekommen sind, haben Sie vermutlich zuvor entweder Guix auf einer fremden Distribution installiert (siehe Kapitel 2 [Installation], Seite 5) oder Sie haben das eigenständige „Guix System“ installiert (siehe Kapitel 3 [Systeminstallation], Seite 27). Nun wird es Zeit für Sie, Guix kennenzulernen. Darum soll es in diesem Abschnitt gehen. Wir möchten Ihnen ein Gefühl vermitteln, wie sich Guix anfühlt.

Bei Guix geht es darum, Software zu installieren, also wollen Sie vermutlich zunächst Software finden, die Sie haben möchten. Sagen wir, Sie suchen ein Programm zur Textverarbeitung. Dann führen Sie diesen Befehl aus:

```
guix search text editor
```

Durch diesen Befehl stoßen Sie auf eine Reihe passender *Pakete*. Für jedes werden Ihnen der Name des Pakets, seine Version, eine Beschreibung und weitere Informationen angezeigt. Sobald Sie das Paket gefunden haben, dass Sie benutzen möchten, sagen wir mal Emacs (aha!), dann können Sie weitermachen und es installieren (führen Sie diesen Befehl als normaler Benutzer aus, *Guix braucht hierfür keine Administratorrechte*!):

```
guix install emacs
```

Sie haben soeben Ihr erstes Paket installiert, Glückwunsch! Das Paket ist nun Teil Ihres voreingestellten Profils in `$HOME/.guix-profile` – ein *Profil* ist ein Verzeichnis, das installierte Pakete enthält. Während Sie das getan haben, ist Ihnen wahrscheinlich aufgefallen, dass Guix vorerstellte Programmbinärdateien herunterlädt, statt sie auf Ihrem Rechner zu kompilieren, außer wenn Sie diese Funktionalität vorher ausdrücklich *abgeschaltet* haben. In diesem Fall ist Guix wahrscheinlich noch immer mit der Erstellung beschäftigt. Siehe Abschnitt 6.3 [Substitute], Seite 56, für mehr Informationen.

Wenn Sie *nicht* Guix System benutzen, wird der Befehl `guix install` diesen Hinweis ausgegeben haben:

```
Hinweis: Vielleicht möchten Sie die nötigen Umgebungsvariablen
festlegen, indem Sie dies ausführen:

     GUIX_PROFILE="$HOME/.guix-profile"
     . "$GUIX_PROFILE/etc/profile"


Sie können sie auch mit `guix package --search-paths -p "$HOME/.guix-profile"' nachles
```

Tatsächlich ist es erforderlich, dass Sie Ihrer Shell erst einmal mitteilen, wo sich `emacs` und andere mit Guix installierte Programme befinden. Wenn Sie die obigen zwei Zeilen in die Shell einfügen, wird genau das bewerkstelligt: `$HOME/.guix-profile/bin` – wo sich das installierte Paket befindet – wird in die `PATH`-Umgebungsvariable eingefügt. Sie können diese zwei Zeilen in Ihre Shell hineinkopieren, damit sie sich sofort auswirken, aber wichtiger ist, dass Sie sie in `~/.bash_profile` kopieren (oder eine äquivalente Datei, wenn Sie *nicht* Bash benutzen), damit die Umgebungsvariablen nächstes Mal gesetzt sind, wenn Sie wieder eine Shell öffnen. Das müssen Sie nur einmal machen und es sorgt auch bei anderen Umgebungsvariablen mit Suchpfaden dafür, dass die installierte Software gefunden wird. Wenn Sie zum Beispiel `python` und Bibliotheken für Python installieren, wird `GUIX_PYTHONPATH` definiert.

Sie können nun weitere Pakete installieren, wann immer Sie möchten. Um die installierten Pakete aufzulisten, führen Sie dies aus:

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

Um ein Paket wieder zu entfernen, benutzen Sie wenig überraschend `guix remove`. Guix hebt sich von anderen Paketverwaltern darin ab, dass Sie jede Operation, die Sie durchgeführt haben, auch wieder *zurücksetzen* können, egal ob `install`, `remove` oder `upgrade`, indem Sie dies ausführen:

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

Das geht deshalb, weil jede Operation eigentlich als *Transaktion* durchgeführt wird, die eine neue *Generation* erstellt. Über folgenden Befehl bekommen Sie diese Generationen und den Unterschied zwischen ihnen angezeigt:

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

Jetzt kennen Sie die Grundlagen der Paketverwaltung!

> **Vertiefung:** Lesen Sie den Abschnitt zur Kapitel 6 [Paketverwaltung], Seite 44, für mehr Informationen dazu. Vielleicht gefällt Ihnen *deklarative* Paketverwaltung mit `guix package --manifest`, die Verwaltung separater *Profile* mit `--profile`, das Löschen alter Generationen und das „Müllsammeln“ (Garbage Collection) sowie andere raffinierte Funktionalitäten, die sich als nützlich erweisen werden, wenn Sie sich länger mit Guix befassen. Wenn Sie Software entwickeln, lernen Sie im Abschnitt Kapitel 8 [Entwicklung], Seite 86, hilfreiche Werkzeuge kennen. Und wenn Sie neugierig sind, wird Ihnen im Abschnitt Abschnitt 6.1 [Funktionalitäten], Seite 44, ein Überblick gegeben, wie und nach welchen Prinzipien Guix aufgebaut ist.

Sobald Sie eine Reihe von Paketen installiert haben, werden Sie diese regelmäßig auf deren neueste und beste Version *aktualisieren* wollen. Dazu ist es erforderlich, dass Sie zunächst die neueste Version von Guix und seiner Paketsammlung mit einem „Pull“ laden:

```
guix pull
```

Das Endergebnis ist ein neuer `guix`-Befehl innerhalb von `~/.config/guix/current/bin`. Sofern Sie nicht Guix System benutzen, sollten Sie dem Hinweis Folge leisten, den Sie bei der ersten Ausführung von `guix pull` angezeigt bekommen, d.h. ähnlich wie oben beschrieben, fügen Sie diese zwei Zeilen in Ihr Terminal und Ihre `.bash_profile` ein:

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

Ebenso müssen Sie Ihre Shell auf diesen neuen `guix`-Befehl hinweisen:

```
hash guix
```

Wenn das geschafft ist, benutzen Sie ein brandneues Guix. Jetzt können Sie damit fortfahren, alle zuvor installierten Pakete zu aktualisieren:

```
guix upgrade
```

Während dieser Befehl ausgeführt wird, werden Sie sehen, dass Binärdateien heruntergeladen werden (vielleicht müssen manche Pakete auch auf Ihrem Rechner erstellt werden). Letztendlich stehen Ihnen die aktualisierten Pakete zur Verfügung. Falls eine der Aktualisierungen nicht Ihren Wünschen entspricht, haben Sie immer auch die Möglichkeit, sie auf den alten Stand zurückzusetzen!

Sie können sich anzeigen lassen, welche Version von Guix Sie genau verwenden, indem Sie dies ausführen:

```
guix describe
```

Die angezeigten Informationen sind *alles, was nötig ist, um genau dasselbe Guix zu installieren*. Damit können Sie zu einem anderen Zeitpunkt oder auf einer anderen Maschine genau dieselbe Software nachbilden.

> **Vertiefung:** Siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65, für weitere Informationen. Siehe Kapitel 7 [Kanäle], Seite 77, für Erklärungen, wie man zusätzliche *Kanäle* als Paketquellen angibt, wie man Guix nachbilden kann, und mehr. Vielleicht hilft Ihnen auch die `time-machine`-Funktion (siehe Abschnitt 6.7 [Aufruf von time-machine], Seite 69).

Wenn Sie Guix System installieren, dürfte eines der ersten Dinge, das Sie tun wollen, eine Aktualisierung Ihres Systems sein. Sobald Sie `guix pull` ausgeführt haben, können Sie mit diesem Befehl das System aktualisieren:

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

Nach Abschluss läuft auf dem System die neueste Version seiner Software-Pakete. Sobald Sie den Rechner neu starten, wird Ihnen ein Untermenü im Bootloader auffallen, das mit „Old system generations“ bezeichnet wird. Über dieses können Sie *eine ältere Generation Ihres Systems* starten, falls die neueste Generation auf irgendeine Weise „kaputt“ ist oder auf andere Weise nicht Ihren Wünschen entspricht. Genau wie bei Paketen können Sie jederzeit *das gesamte System* auf eine vorherige Generation *zurücksetzen*:

```
sudo guix system roll-back
```

Es gibt viele Feineinstellungen, die Sie an Ihrem System wahrscheinlich vornehmen möchten. Vielleicht möchten Sie neue Benutzerkonten oder neue Systemdienste hinzufügen, mit anderen Konfigurationen solcher Dienste experimentieren oder Ähnliches. Die *gesamte* Systemkonfiguration wird durch die Datei `/etc/config.scm` beschrieben. Siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250, wo erklärt wird, wie man sie ändert.

Jetzt wissen Sie genug, um anzufangen!

> **Weitere Ressourcen:** Der Rest dieses Handbuchs stellt eine Referenz für alles rund um Guix dar. Hier sind ein paar andere Ressourcen, die sich als nützlich erweisen könnten:
>
> - Siehe *das GNU-Guix-Kochbuch* für eine Liste von anleitungsartigen Rezepten zu einer Vielzahl von Anwendungszwecken.
> - In der GNU Guix Reference Card (`https://guix.gnu.org/guix-refcard.pdf`) sind in zwei Seiten die meisten Befehle und Befehlszeilenoptionen aufgeführt, die Sie jemals brauchen werden.
> - Auf dem Webauftritt gibt es Lehrvideos (`https://guix.gnu.org/de/videos/`) zu Themen wie der alltäglichen Nutzung von Guix, wo man Hilfe bekommt und wie man mitmachen kann.
> - Siehe Kapitel 14 [Dokumentation], Seite 680, um zu erfahren, wie Sie von Ihrem Rechner aus auf Dokumentation von Software zugreifen können.
>
> Wir hoffen, dass Ihnen Guix genauso viel Spaß bereitet wie der Guix-Gemeinde bei seiner Entwicklung!

6 Paketverwaltung

Der Zweck von GNU Guix ist, Benutzern die leichte Installation, Aktualisierung und Entfernung von Software-Paketen zu ermöglichen, ohne dass sie ihre Erstellungsprozeduren oder Abhängigkeiten kennen müssen. Guix kann natürlich noch mehr als diese offensichtlichen Funktionalitäten.

Dieses Kapitel beschreibt die Hauptfunktionalitäten von Guix, sowie die von Guix angebotenen Paketverwaltungswerkzeuge. Zusätzlich zu den im Folgenden beschriebenen Befehlszeilen-Benutzerschnittstellen (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45) können Sie auch mit der Emacs-Guix-Schnittstelle (siehe *Referenzhandbuch von Emacs-Guix*) arbeiten, nachdem Sie das Paket `emacs-guix` installiert haben (führen Sie zum Einstieg in Emacs-Guix den Emacs-Befehl *`M-x guix-help`* aus):

```
guix install emacs-guix
```

6.1 Funktionalitäten

Wir nehmen hier an, dass Sie schon Ihre ersten Erfahrungen mit Guix gemacht haben (siehe Kapitel 5 [Einstieg in Guix], Seite 41) und gerne einen Überblick darüber bekommen würden, wie Guix im Detail funktioniert.

Wenn Sie Guix benutzen, landet jedes Paket schließlich im *Paket-Store* in seinem eigenen Verzeichnis – der Name ist ähnlich wie `/gnu/store/xxx-package-1.2`, wobei `xxx` eine Zeichenkette in Base32-Darstellung ist.

Statt diese Verzeichnisse direkt anzugeben, haben Nutzer ihr eigenes *Profil*, welches auf diejenigen Pakete zeigt, die sie tatsächlich benutzen wollen. Diese Profile sind im Persönlichen Verzeichnis des jeweiligen Nutzers gespeichert als `$HOME/.guix-profile`.

Zum Beispiel installiert `alice` GCC 4.7.2. Dadurch zeigt dann `/home/alice/.guix-profile/bin/gcc` auf `/gnu/store/...-gcc-4.7.2/bin/gcc`. Auf demselben Rechner hat `bob` bereits GCC 4.8.0 installiert. Das Profil von `bob` zeigt dann einfach weiterhin auf `/gnu/store/...-gcc-4.8.0/bin/gcc` – d.h. beide Versionen von GCC koexistieren auf demselben System, ohne sich zu stören.

Der Befehl `guix package` ist das zentrale Werkzeug, um Pakete zu verwalten (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Es arbeitet auf dem eigenen Profil jedes Nutzers und kann *mit normalen Benutzerrechten* ausgeführt werden.

Der Befehl stellt die offensichtlichen Installations-, Entfernungs- und Aktualisierungsoperationen zur Verfügung. Jeder Aufruf ist tatsächlich eine eigene *Transaktion*: Entweder die angegebene Operation wird erfolgreich durchgeführt, oder gar nichts passiert. Wenn also der Prozess von `guix package` während der Transaktion beendet wird, oder es zum Stromausfall während der Transaktion kommt, dann bleibt der alte, nutzbare Zustands des Nutzerprofils erhalten.

Zudem kann jede Pakettransaktion *zurückgesetzt* werden (Rollback). Wird also zum Beispiel durch eine Aktualisierung eine neue Version eines Pakets installiert, die einen schwerwiegenden Fehler zur Folge hat, können Nutzer ihr Profil einfach auf die vorherige Profilinstanz zurücksetzen, von der sie wissen, dass sie gut lief. Ebenso unterliegt bei Guix auch die globale Systemkonfiguration transaktionellen Aktualisierungen und Rücksetzungen (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250).

Alle Pakete im Paket-Store können vom *Müllsammler* (Garbage Collector) gelöscht werden. Guix ist in der Lage, festzustellen, welche Pakete noch durch Benutzerprofile referenziert werden, und entfernt nur diese, die nachweislich nicht mehr referenziert werden (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61). Benutzer können auch ausdrücklich alte Generationen ihres Profils löschen, damit die zugehörigen Pakete vom Müllsammler gelöscht werden können.

Guix verfolgt einen *rein funktionalen* Ansatz bei der Paketverwaltung, wie er in der Einleitung beschrieben wurde (siehe Kapitel 1 [Einführung], Seite 1). Jedes Paketverzeichnis im `/gnu/store` hat einen Hash all seiner bei der Erstellung benutzten Eingaben im Namen – Compiler, Bibliotheken, Erstellungs-Skripts etc. Diese direkte Entsprechung ermöglicht es Benutzern, eine Paketinstallation zu benutzen, die sicher dem aktuellen Stand ihrer Distribution entspricht. Sie hilft auch dabei, die *Reproduzierbarkeit der Erstellungen* zu maximieren: Dank den isolierten Erstellungsumgebungen, die benutzt werden, resultiert eine Erstellung wahrscheinlich in bitweise identischen Dateien, auch wenn sie auf unterschiedlichen Maschinen durchgeführt wird (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

Auf dieser Grundlage kann Guix *transparent Binär- oder Quelldateien ausliefern*. Wenn eine vorerstellte Binärdatei für ein `/gnu/store`-Objekt von einer externen Quelle verfügbar ist – ein *Substitut* –, lädt Guix sie einfach herunter und entpackt sie, andernfalls erstellt Guix das Paket lokal aus seinem Quellcode (siehe Abschnitt 6.3 [Substitute], Seite 56). Weil Erstellungsergebnisse normalerweise Bit für Bit reproduzierbar sind, müssen die Nutzer den Servern, die Substitute anbieten, nicht blind vertrauen; sie können eine lokale Erstellung erzwingen und Substitute *anfechten* (siehe Abschnitt 10.12 [Aufruf von guix challenge], Seite 238).

Kontrolle über die Erstellungsumgebung ist eine auch für Entwickler nützliche Funktionalität. Der Befehl `guix shell` ermöglicht es Entwicklern eines Pakets, schnell die richtige Entwicklungsumgebung für ihr Paket einzurichten, ohne manuell die Abhängigkeiten des Pakets in ihr Profil installieren zu müssen (siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86).

Ganz Guix und all seine Paketdefinitionen stehen unter Versionskontrolle und `guix pull` macht es möglich, auf dem Verlauf der Entwicklung von Guix selbst „in der Zeit zu reisen“ (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65). Dadurch kann eine Instanz von Guix auf einer anderen Maschine oder zu einem späteren Zeitpunkt genau nachgebildet werden, wodurch auch *vollständige Software-Umgebungen gänzlich nachgebildet* werden können, mit genauer *Provenienzverfolgung*, wo diese Software herkommt.

6.2 `guix package` aufrufen

Der Befehl `guix package` ist ein Werkzeug, womit Nutzer Pakete installieren, aktualisieren, entfernen und auf vorherige Konfigurationen zurücksetzen können. Diese Operationen arbeiten auf einem *Benutzerprofil*, d.h. einem Verzeichnis, das installierte Pakete enthält. Jeder Benutzer verfügt über ein Standardprofil in `$HOME/.guix-profile`. Dabei wird nur das eigene Profil des Nutzers verwendet, und es funktioniert mit normalen Benutzerrechten, ohne Administratorrechte (siehe Abschnitt 6.1 [Funktionalitäten], Seite 44). Die Syntax ist:

```
guix package Optionen
```

In erster Linie geben die *Optionen* an, welche Operationen in der Transaktion durchgeführt werden sollen. Nach Abschluss wird ein neues Profil erzeugt, aber vorherige *Generationen* des Profils bleiben verfügbar, falls der Benutzer auf sie zurückwechseln will.

Um zum Beispiel `lua` zu entfernen und `guile` und `guile-cairo` in einer einzigen Transaktion zu installieren:

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

Um es Ihnen einfacher zu machen, bieten wir auch die folgenden Alias-Namen an:

- `guix search` ist eine andere Schreibweise für `guix package -s`,
- `guix install` ist eine andere Schreibweise für `guix package -i`,
- `guix remove` ist eine andere Schreibweise für `guix package -r`,
- `guix upgrade` ist eine andere Schreibweise für `guix package -u`
- und `guix show` ist eine andere Schreibweise für `guix package --show=`.

Diese Alias-Namen sind weniger ausdrucksstark als `guix package` und stellen weniger Befehlszeilenoptionen bereit, deswegen werden Sie vermutlich manchmal `guix package` direkt benutzen wollen.

`guix package` unterstützt auch ein *deklaratives Vorgehen*, wobei der Nutzer die genaue Menge an Paketen, die verfügbar sein sollen, festlegt und über die Befehlszeilenoption `--manifest` übergibt (siehe [profile-manifest], Seite 50).

Für jeden Benutzer wird automatisch eine symbolische Verknüpfung zu seinem Standardprofil angelegt als `$HOME/.guix-profile`. Diese symbolische Verknüpfung zeigt immer auf die aktuelle Generation des Standardprofils des Benutzers. Somit können Nutzer `$HOME/.guix-profile/bin` z.B. zu ihrer Umgebungsvariablen `PATH` hinzufügen.

Wenn Sie *nicht* Guix System benutzen, sollten Sie in Betracht ziehen, folgende Zeilen zu Ihrem `~/.bash_profile` hinzuzufügen (siehe Abschnitt "Bash Startup Files" in *Referenzhandbuch von GNU Bash*), damit in neu erzeugten Shells alle Umgebungsvariablen richtig definiert werden:

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

Ist Ihr System für mehrere Nutzer eingerichtet, werden Nutzerprofile an einem Ort gespeichert, der als *Müllsammlerwurzel* registriert ist, auf die `$HOME/.guix-profile` zeigt (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61). Dieses Verzeichnis ist normalerweise *localstatedir*`/guix/profiles/per-user/`*Benutzer*, wobei *localstatedir* der an `configure` als `--localstatedir` übergebene Wert ist und *Benutzer* für den jeweiligen Benutzernamen steht. Das `per-user`-Verzeichnis wird erstellt, wenn `guix-daemon` gestartet wird, und das Unterverzeichnis *Benutzer* wird durch `guix package` erstellt.

Als *Optionen* kann vorkommen:

`--install=`*Paket* `...`
`-i` *Paket* `...`

Die angegebenen *Pakete* installieren.

Jedes *Paket* kann einfach durch seinen Paketnamen aufgeführt werden, wie `guile`, optional gefolgt von einem At-Zeichen @ und einer Versionsnummer, wie `guile@3.0.7` oder auch nur `guile@3.0`. In letzterem Fall wird die neueste Version mit Präfix `3.0` ausgewählt.

Wird keine Versionsnummer angegeben, wird die neueste verfügbare Version ausgewählt. Zudem kann in der Spezifikation von *Paket* ein Doppelpunkt auftauchen, gefolgt vom Namen einer der Ausgaben des Pakets, wie `gcc:doc` oder `binutils@2.22:lib` (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60).

Pakete mit zugehörigem Namen (und optional der Version) werden unter den Modulen der GNU-Distribution gesucht (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

Alternativ können Sie für *Paket* auch einen Dateinamen aus dem Store direkt angeben, etwa `/gnu/store/...-guile-3.0.7`. Den Dateinamen erfahren Sie zum Beispiel mit `guix build`.

Manchmal haben Pakete *propagierte Eingaben*: Als solche werden Abhängigkeiten bezeichnet, die automatisch zusammen mit dem angeforderten Paket installiert werden (im Abschnitt [package-propagated-inputs], Seite 114, sind weitere Informationen über propagierte Eingaben in Paketdefinitionen zu finden).

Ein Beispiel ist die GNU-MPC-Bibliothek: Ihre C-Headerdateien verweisen auf die der GNU-MPFR-Bibliothek, welche wiederum auf die der GMP-Bibliothek verweisen. Wenn also MPC installiert wird, werden auch die MPFR- und GMP-Bibliotheken in das Profil installiert; entfernt man MPC, werden auch MPFR und GMP entfernt – außer sie wurden noch auf andere Art ausdrücklich vom Nutzer installiert.

Abgesehen davon setzen Pakete manchmal die Definition von Umgebungsvariablen für ihre Suchpfade voraus (siehe die Erklärung von `--search-paths` weiter unten). Alle fehlenden oder womöglich falschen Definitionen von Umgebungsvariablen werden hierbei gemeldet.

`--install-from-expression=`*Ausdruck*
`-e` *Ausdruck*

Das Paket installieren, zu dem der *Ausdruck* ausgewertet wird.

Beim *Ausdruck* muss es sich um einen Scheme-Ausdruck handeln, der zu einem `<package>`-Objekt ausgewertet wird. Diese Option ist besonders nützlich, um zwischen gleichnamigen Varianten eines Pakets zu unterscheiden, durch Ausdrücke wie `(@ (gnu packages base) guile-final)`.

Beachten Sie, dass mit dieser Option die erste Ausgabe des angegebenen Pakets installiert wird, was unzureichend sein kann, wenn eine bestimmte Ausgabe eines Pakets mit mehreren Ausgaben gewünscht ist.

`--install-from-file=`*Datei*
`-f` *Datei* Das Paket installieren, zu dem der Code in der *Datei* ausgewertet wird.

Zum Beispiel könnte die *Datei* eine Definition wie diese enthalten (siehe Abschnitt 9.2 [Pakete definieren], Seite 109):

```
(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+))
```

Entwickler könnten es für nützlich erachten, eine solche `guix.scm`-Datei im Quellbaum ihres Projekts abzulegen, mit der Zwischenstände der Entwicklung getestet und reproduzierbare Erstellungsumgebungen aufgebaut werden können (siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86).

Es ist auch möglich, eine *Datei* mit einer JSON-Repräsentation von einer oder mehr Paketdefinitionen anzugeben. Wenn Sie `guix package -f` auf `hello.json` mit folgendem Inhalt ausführen würden, würde das Paket `greeter` installiert, nachdem `myhello` erstellt wurde:

```
[
  {
    "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": "https://example.com/greeter-1.0.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=`*Paket* `...`
`-r` *Paket* `...`
: Die angegebenen *Paket*e entfernen.

  Wie auch bei `--install` kann jedes *Paket* neben dem Paketnamen auch eine Versionsnummer und/oder eine Ausgabe benennen. Zum Beispiel würde '`-r glibc:debug`' die `debug`-Ausgabe von `glibc` aus dem Profil entfernen.

`--upgrade[=`*Regexp* `...]`
`-u [`*Regexp* `...]`
: Alle installierten Pakete aktualisieren. Wenn einer oder mehr reguläre Ausdrücke (Regexps) angegeben wurden, werden nur diejenigen installierten Pakete aktualisiert, deren Name zu einer der *Regexp*s passt. Siehe auch weiter unten die Befehlszeilenoption `--do-not-upgrade`.

  Beachten Sie, dass das Paket so auf die neueste Version unter den Paketen gebracht wird, die in der aktuell installierten Distribution vorliegen. Um jedoch Ihre Distribution zu aktualisieren, sollten Sie regelmäßig `guix pull` ausführen (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65).

  Wenn Sie Ihre Pakete aktualisieren, werden die bei der Erstellung des vorherigen Profils angewandten Paketumwandlungen automatisch erneut angwandt (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192). Nehmen wir zum Beispiel an, Sie haben zuerst Emacs von der Spitze seines Entwicklungs-Branches installiert:

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

  Wenn Sie das nächste Mal `guix upgrade` ausführen, lädt Guix von neuem die Spitze des Emacs-Entwicklungsbranches, um aus diesem Checkout `emacs-next` zu erstellen.

  Beachten Sie, dass Umwandlungsoptionen wie `--with-branch` und `--with-source` von externem Zustand abhängen. Es liegt an Ihnen, sicherzustellen, dass sie wie erwartet funktionieren. Sie können auf Pakete angewandte Umwandlungen auch weglassen, indem Sie das ausführen:

  ```
  guix install Paket
  ```

`--do-not-upgrade[=`*Regexp* `...]`
: In Verbindung mit der Befehlszeilenoption `--upgrade`, führe *keine* Aktualisierung von Paketen durch, deren Name zum regulären Ausdruck *Regexp* passt. Um zum Beispiel alle Pakete im aktuellen Profil zu aktualisieren mit Ausnahme derer, die „emacs“ im Namen haben:

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

`--manifest=`*Datei*
`-m` *Datei* Erstellt eine neue Generation des Profils aus dem vom Scheme-Code in *Datei* gelieferten Manifest-Objekt. Wenn diese Befehlszeilenoption mehrmals wiederholt angegeben wird, werden die Manifeste aneinandergehängt.

Dadurch könnrn Sie den Inhalt des Profils *deklarieren*, statt ihn durch eine Folge von Befehlen wie `--install` u.Ä. zu generieren. Der Vorteil ist, dass die *Datei* unter Versionskontrolle gestellt werden kann, auf andere Maschinen zum Reproduzieren desselben Profils kopiert werden kann und Ähnliches.

Der Code in der *Datei* muss ein *Manifest*-Objekt liefern, was ungefähr einer Liste von Paketen entspricht:

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

(packages->manifest
 (list emacs
       guile-2.0
       ;; Eine bestimmte Paketausgabe nutzen.
       (list guile-2.0 "debug")))
```

Siehe Abschnitt 9.4 [Manifeste verfassen], Seite 125, für Informationen dazu, wie man ein Manifest schreibt. Siehe [export-manifest], Seite 55, um zu erfahren, wie Sie aus einem bestehenden Profil eine Manifestdatei erzeugen können.

`--roll-back`
Wechselt zur vorherigen *Generation* des Profils zurück – d.h. macht die letzte Transaktion rückgängig.

In Verbindung mit Befehlszeilenoptionen wie `--install` wird zuerst zurückgesetzt, bevor andere Aktionen durchgeführt werden.

Ein Rücksetzen der ersten Generation, die installierte Pakete enthält, wechselt das Profil zur *nullten Generation*, die keinerlei Dateien enthält, abgesehen von Metadaten über sich selbst.

Nach dem Zurücksetzen überschreibt das Installieren, Entfernen oder Aktualisieren von Paketen vormals zukünftige Generationen, d.h. der Verlauf der Generationen eines Profils ist immer linear.

`--switch-generation=`*Muster*
`-S` *Muster* Wechselt zu der bestimmten Generation, die durch das *Muster* bezeichnet wird.

Als *Muster* kann entweder die Nummer einer Generation oder eine Nummer mit vorangestelltem „`+`“ oder „`-`“ dienen. Letzteres springt die angegebene Anzahl an Generationen vor oder zurück. Zum Beispiel kehrt `--switch-generation=+1` nach einem Zurücksetzen wieder zur neueren Generation zurück.

Der Unterschied zwischen `--roll-back` und `--switch-generation=-1` ist, dass `--switch-generation` keine nullte Generation erzeugen wird; existiert die angegebene Generation nicht, bleibt schlicht die aktuelle Generation erhalten.

`--search-paths[=`*Art*`]`
Führe die Definitionen von Umgebungsvariablen auf, in Bash-Syntax, die nötig sein könnten, um alle installierten Pakete nutzen zu können. Diese Umgebungs-

variablen werden benutzt, um die *Suchpfade* für Dateien festzulegen, die von einigen installierten Paketen benutzt werden.

Zum Beispiel braucht GCC die Umgebungsvariablen `CPATH` und `LIBRARY_PATH`, um zu wissen, wo sich im Benutzerprofil Header und Bibliotheken befinden (siehe Abschnitt "Environment Variables" in *Using the GNU Compiler Collection (GCC)*). Wenn GCC und, sagen wir, die C-Bibliothek im Profil installiert sind, schlägt `--search-paths` also vor, diese Variablen jeweils auf *`profile`*`/include` und *`profile`*`/lib` verweisen zu lassen (siehe Abschnitt 9.8 [Suchpfade], Seite 161, für Informationen, wie Paketen Suchpfadspezifikationen zugeordnet werden).

Die typische Nutzung ist, in der Shell diese Variablen zu definieren:

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

Als *Art* kann entweder `exact`, `prefix` oder `suffix` gewählt werden, wodurch die gelieferten Definitionen der Umgebungsvariablen entweder exakt die Einstellungen für Guix meldet, oder sie als Präfix oder Suffix an den aktuellen Wert dieser Variablen anhängt. Gibt man keine *Art* an, wird der Vorgabewert `exact` verwendet.

Diese Befehlszeilenoption kann auch benutzt werden, um die *kombinierten* Suchpfade mehrerer Profile zu berechnen. Betrachten Sie dieses Beispiel:

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

Der letzte Befehl oben meldet auch die Definition der Umgebungsvariablen `GUILE_LOAD_PATH`, obwohl für sich genommen weder `foo` noch `bar` zu dieser Empfehlung führen würden.

`--profile=`*`Profil`*

`-p` *`Profil`* Auf *Profil* anstelle des Standardprofils des Benutzers arbeiten.

Als *Profil* muss der Name einer Datei angegeben werden, die dann nach Abschluss der Transaktion erzeugt wird. Konkret wird *Profil* nur zu einer symbolischen Verknüpfung („Symlink“) auf das eigentliche Profil gemacht, in das Pakete installiert werden.

```
$ guix install hello -p ~/code/mein-profil
...
$ ~/code/mein-profil/bin/hello
Hallo, Welt!
```

Um das Profil loszuwerden, genügt es, die symbolische Verknüpfung und damit einhergehende Verknüpfungen, die auf bestimmte Generationen verweisen, zu entfernen:

```
$ rm ~/code/mein-profil ~/code/mein-profil-*-link
```

`--list-profiles`

Alle Profile des Benutzers auflisten:

```
$ guix package --list-profiles
/home/charlie/.guix-profile
```

```
/home/charlie/code/my-profile
/home/charlie/code/devel-profile
/home/charlie/tmp/test
```

Wird es als Administratornutzer „root“ ausgeführt, werden die Profile aller Benutzer aufgelistet.

`--allow-collisions`
: Kollidierende Pakete im neuen Profil zulassen. Benutzung auf eigene Gefahr!

  Standardmäßig wird `guix package` *Kollisionen* als Fehler auffassen und melden. Zu Kollisionen kommt es, wenn zwei oder mehr verschiedene Versionen oder Varianten desselben Pakets im Profil landen.

`--bootstrap`
: Erstellt das Profil mit dem Bootstrap-Guile. Diese Option ist nur für Entwickler der Distribution nützlich.

Zusätzlich zu diesen Aktionen unterstützt `guix package` folgende Befehlszeilenoptionen, um den momentanen Zustand eines Profils oder die Verfügbarkeit von Paketen nachzulesen:

`--search=`*`Regexp`*
`-s` *`Regexp`*
: Führt alle verfügbaren Pakete auf, deren Name, Zusammenfassung oder Beschreibung zum regulären Ausdruck *Regexp* passt, ohne Groß- und Kleinschreibung zu unterscheiden und sortiert nach ihrer Relevanz. Alle Metadaten passender Pakete werden im `recutils`-Format geliefert (siehe *GNU recutils manual*).

  So können bestimmte Felder mit dem Befehl `recsel` extrahiert werden, zum Beispiel:

```
$ 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
```

  Ebenso kann der Name aller zu den Bedingungen der GNU LGPL, Version 3, verfügbaren Pakete ermittelt werden:

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

name: gmp
...
```

  Es ist auch möglich, Suchergebnisse näher einzuschränken, indem Sie `-s` mehrmals an `guix package` übergeben, oder mehrere Argumente an `guix search` übergeben. Zum Beispiel liefert folgender Befehl eines Liste von Brettspielen:

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

Würden wir `-s game` weglassen, bekämen wir auch Software-Pakete aufgelistet, die mit „printed circuit boards“ (elektronischen Leiterplatten) zu tun haben; ohne die spitzen Klammern um `board` bekämen wir auch Pakete, die mit „keyboards“ (Tastaturen, oder musikalischen Keyboard) zu tun haben.

Es ist Zeit für ein komplexeres Beispiel. Folgender Befehl sucht kryptografische Bibliotheken, filtert Haskell-, Perl-, Python- und Ruby-Bibliotheken heraus und gibt Namen und Zusammenfassung passender Pakete aus:

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

Siehe Abschnitt “Selection Expressions” in *GNU recutils manual*, es enthält weitere Informationen über *Auswahlausdrücke* mit `recsel -e`.

`--show=`*Paket*

Zeigt Details über das *Paket* aus der Liste verfügbarer Pakete, im `recutils`-Format (siehe *GNU recutils manual*).

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

name: guile
version: 3.0.2

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

Sie können auch den vollständigen Namen eines Pakets angeben, um Details nur über diese Version angezeigt zu bekommen (diesmal benutzen wir die andere Schreibweise `guix show`):

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

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

Listet die derzeit installierten Pakete im angegebenen Profil auf, die zuletzt installierten Pakete zuletzt. Wenn ein regulärer Ausdruck *Regexp* angegeben wird, werden nur installierte Pakete aufgeführt, deren Name zu *Regexp* passt.

Zu jedem installierten Paket werden folgende Informationen angezeigt, durch Tabulatorzeichen getrennt: der Paketname, die Version als Zeichenkette, welche Teile des Pakets installiert sind (zum Beispiel `out`, wenn die Standard-Paketausgabe installiert ist, `include`, wenn seine Header installiert sind, usw.) und an welchem Pfad das Paket im Store zu finden ist.

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

Listet Pakete auf, die in der aktuell installierten Distribution dieses Systems verfügbar sind (siehe Abschnitt 1.2 [GNU-Distribution], Seite 2). Wenn ein regulärer Ausdruck *Regexp* angegeben wird, werden nur Pakete aufgeführt, deren Name zum regulären Ausdruck *Regexp* passt.

Zu jedem Paket werden folgende Informationen getrennt durch Tabulatorzeichen ausgegeben: der Name, die Version als Zeichenkette, die Teile des Programms (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60) und die Stelle im Quellcode, an der das Paket definiert ist.

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

Liefert eine Liste der Generationen zusammen mit dem Datum, an dem sie erzeugt wurden; zu jeder Generation werden zudem die installierten Pakete angezeigt, zuletzt installierte Pakete zuletzt. Beachten Sie, dass die nullte Generation niemals angezeigt wird.

Zu jedem installierten Paket werden folgende Informationen durch Tabulatorzeichen getrennt angezeigt: der Name des Pakets, die Version als Zeichenkette, welcher Teil des Pakets installiert ist (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60) und an welcher Stelle sich das Paket im Store befindet.

Wenn ein *Muster* angegeben wird, liefert der Befehl nur dazu passende Generationen. Gültige Muster sind zum Beispiel:

- *Ganze Zahlen und kommagetrennte ganze Zahlen.* Beide Muster bezeichnen Generationsnummern. Zum Beispiel liefert `--list-generations=1` die erste Generation.

  Durch `--list-generations=1,8,2` werden drei Generationen in der angegebenen Reihenfolge angezeigt. Weder Leerzeichen noch ein Komma am Schluss der Liste ist erlaubt.

- *Bereiche.* `--list-generations=2..9` gibt die angegebenen Generationen und alles dazwischen aus. Beachten Sie, dass der Bereichsanfang eine kleinere Zahl als das Bereichsende sein muss.

  Sie können auch kein Bereichsende angeben, zum Beispiel liefert `--list-generations=2..` alle Generationen ab der zweiten.

- *Zeitdauern.* Sie können auch die letzten *N* Tage, Wochen oder Monate angeben, indem Sie eine ganze Zahl gefolgt von jeweils „d“, „w“ oder „m“ angeben (dem ersten Buchstaben der Maßeinheit der Dauer im Englischen). Zum Beispiel listet `--list-generations=20d` die Generationen auf, die höchstens 20 Tage alt sind.

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

Wird kein *Muster* angegeben, werden alle Generationen außer der aktuellen entfernt.

Dieser Befehl akzeptiert dieselben Muster wie `--list-generations`. Wenn ein *Muster* angegeben wird, werden die passenden Generationen gelöscht.

Wenn das *Muster* für eine Zeitdauer steht, werden diejenigen Generationen gelöscht, die *älter* als die angegebene Dauer sind. Zum Beispiel löscht `--delete-generations=1m` die Generationen, die mehr als einen Monat alt sind.

Falls die aktuelle Generation zum Muster passt, wird sie *nicht* gelöscht. Auch die nullte Generation wird niemals gelöscht.

Beachten Sie, dass Sie auf gelöschte Generationen nicht zurückwechseln können. Dieser Befehl sollte also nur mit Vorsicht benutzt werden.

`--export-manifest`

Auf die Standardausgabe ein Manifest ausgeben, das mit `--manifest` genutzt werden kann und dem/den gewählten Profil(en) entspricht.

Diese Befehlszeilenoption erleichtert Ihnen den Wechsel vom „imperativen“ Betriebsmodus, wo Sie immer wieder `guix install`, `guix upgrade` etc. ausführen, zum deklarativen Modus, der mit `--manifest` zur Verfügung steht.

Seien Sie sich bewusst, dass das erzeugte Manifest nur eine *Annäherung* des Inhalts Ihres Profils ist. Je nachdem, wie Ihr Profil erzeugt wurde, kann es auf andere Pakete oder Paketversionen verweisen.

Beachten Sie, dass ein Manifest rein symbolisch ist; es enthält nur die Namen und vielleicht Versionen der Pakete, aber die Bedeutung davon wandelt sich mit der Zeit. Wenn Sie wollen, dass die Pakete aus immer derselben Kanalversion stammen, mit der das Profil oder die Profile erstellt wurden, siehe `--export-channels` unten.

`--export-channels`

Auf die Standardausgabe die Liste der Kanäle schreiben, die das gewählte Profil benutzt bzw. die die gewählten Profile benutzen. Das Format der Kanalspezifikation ist für `guix pull --channels` und `guix time-machine --channels` geeignet (siehe Kapitel 7 [Kanäle], Seite 77).

Zusammen mit `--export-manifest` macht diese Befehlszeilenoption Informationen verfügbar, um das aktuelle Profil nachzubilden (siehe Abschnitt 7.3 [Guix nachbilden], Seite 78).

Beachten Sie jedoch, dass die Ausgabe dieses Befehls nur eine *Annäherung* dessen ist, woraus ein Profil tatsächlich erstellt wurde. Insbesondere kann ein Profil aus mehreren Versionen desselben Kanals aufgebaut worden sein. In diesem Fall wählt `--export-manifest` die neueste aus und schreibt die anderen Versionen in einem Kommentar dazu. Wenn Sie wirklich Pakete aus unterschiedlichen Kanalversionen zu nehmen brauchen, können Sie dazu in Ihrem Manifest Untergeordnete angeben (siehe Abschnitt 6.8 [Untergeordnete], Seite 70).

Dies stellt zusammen mit `--export-manifest` eine gute Gelegenheit dar, wenn Sie bereit sind, vom „imperativen“ Modell auf das vollständig deklarative Modell zu wechseln, wo Sie eine Manifestdatei zusammen mit einer Kanaldatei benutzen, die ganz genau festlegt, welche Kanalversion(en) Sie wollen.

Zu guter Letzt können Sie, da `guix package` Erstellungsprozesse zu starten vermag, auch alle gemeinsamen Erstellungsoptionen (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189) verwenden. Auch Paketumwandlungsoptionen wie `--with-source` sind

möglich und bleiben über Aktualisierungen hinweg erhalten (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192).

6.3 Substitute

Guix kann transparent Binär- oder Quelldateien ausliefern. Das heißt, Dinge können sowohl lokal erstellt als auch als vorerstellte Objekte von einem Server heruntergeladen werden, oder beides gemischt. Wir bezeichnen diese vorerstellten Objekte als *Substitute* – sie substituieren lokale Erstellungsergebnisse. In vielen Fällen geht das Herunterladen eines Substituts wesentlich schneller, als Dinge lokal zu erstellen.

Substitute können alles sein, was das Ergebnis einer Ableitungserstellung ist (siehe Abschnitt 9.10 [Ableitungen], Seite 167). Natürlich sind sie üblicherweise vorerstellte Paket-Binärdateien, aber wenn zum Beispiel ein Quell-Tarball das Ergebnis einer Ableitungserstellung ist, kann auch er als Substitut verfügbar sein.

6.3.1 Offizielle Substitut-Server

`ci.guix.gnu.org` und `bordeaux.guix.gnu.org` sind jeweils Fassaden für offizielle Erstellungsfarmen („Build Farms“), die kontinuierlich Guix-Pakete für einige Prozessorarchitekturen erstellen und sie als Substitute zur Verfügung stellen. Sie sind die standardmäßige Quelle von Substituten; durch Übergeben der Befehlszeilenoption `--substitute-urls` an entweder den `guix-daemon` (siehe [`guix-daemon --substitute-urls`], Seite 19) oder Client-Werkzeuge wie `guix package` (siehe [die Befehlszeilenoption `--substitute-urls` beim Client], Seite 190) kann eine abweichende Einstellung benutzt werden.

Substitut-URLs können entweder HTTP oder HTTPS sein. HTTPS wird empfohlen, weil die Kommunikation verschlüsselt ist; umgekehrt kann bei HTTP die Kommunikation belauscht werden, wodurch der Angreifer zum Beispiel erfahren könnte, ob Ihr System über noch nicht behobene Sicherheitsschwachstellen verfügt.

Substitute von den offiziellen Erstellungsfarmen sind standardmäßig erlaubt, wenn Sie Guix System verwenden (siehe Abschnitt 1.2 [GNU-Distribution], Seite 2). Auf Fremddistributionen sind sie allerdings standardmäßig ausgeschaltet, solange Sie sie nicht ausdrücklich in einem der empfohlenen Installationsschritte erlaubt haben (siehe Kapitel 2 [Installation], Seite 5). Die folgenden Absätze beschreiben, wie Sie Substitute für die offizielle Erstellungsfarm an- oder ausschalten; dieselbe Prozedur kann auch benutzt werden, um Substitute für einen beliebigen anderen Substitutserver zu erlauben.

6.3.2 Substitut-Server autorisieren

Um es Guix zu gestatten, Substitute von `ci.guix.gnu.org`, `bordeaux.guix.gnu.org` oder einem Spiegelserver herunterzuladen, müssen Sie die zugehörigen öffentlichen Schlüssel zur Access Control List (ACL, Zugriffssteuerungsliste) für Archivimporte hinzufügen, mit Hilfe des Befehls `guix archive` (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74). Dies impliziert, dass Sie darauf vertrauen, dass der Substitutserver nicht kompromittiert wurde und unverfälschte Substitute liefert.

> **Anmerkung:** Wenn Sie Guix System benutzen, können Sie diesen Abschnitt hier überspringen, denn Guix System ist so voreingestellt, Substitute von `ci.guix.gnu.org` und `bordeaux.guix.gnu.org` zu autorisieren.

Der öffentliche Schlüssel für jeden vom Guix-Projekt verwalteten Substitutserver wird zusammen mit Guix installiert, in das Verzeichnis *prefix*`/share/guix/`, wobei *prefix* das bei der Installation angegebene Präfix von Guix ist. Wenn Sie Guix aus seinem Quellcode heraus installieren, sollten Sie sichergehen, dass Sie die GPG-Signatur (auch „Beglaubigung“ genannt) von `guix-1.4.0.tar.gz` prüfen, worin sich dieser öffentliche Schlüssel befindet. Dann können Sie so etwas wie hier ausführen:

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

Sobald es eingerichtet wurde, sollte sich die Ausgabe eines Befehls wie `guix build` von so etwas:

```
$ guix build emacs --dry-run
Folgende Ableitungen würden erstellt:
   /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
...
```

in so etwas verwandeln:

```
$ guix build emacs --dry-run
112.3 MB würden heruntergeladen:
   /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
...
```

Der Text hat sich von „Folgende Ableitungen würden erstellt“ zu „112.3 MB würden heruntergeladen“ geändert. Das zeigt an, dass Substitute von den festgelegten Substitutservern nutzbar sind und für zukünftige Erstellungen heruntergeladen werden, wann immer es möglich ist.

Der Substitutsmechanismus kann global ausgeschaltet werden, indem Sie dem `guix-daemon` beim Starten die Befehlszeilenoption `--no-substitutes` übergeben (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18). Er kann auch temporär ausgeschaltet werden, indem Sie `--no-substitutes` an `guix package`, `guix build` und andere Befehlszeilenwerkzeuge übergeben.

6.3.3 Substitute von anderen Servern holen

Guix kann auf unterschiedlichen Servern nach Substituten schauen und sie von dort laden. Das lohnt sich, wenn Sie Pakete von zusätzlichen Kanälen laden, für die es auf dem offiziellen Server keine Substitute gibt, auf einem anderen aber schon. Eine andere Situation, wo es sich anbietet, ist, wenn Sie Substitute lieber vom Substitutserver beziehen, der zu Ihrer Organisation gehört, und den offiziellen Server nur im Ausnahmefall einsetzen oder ganz weglassen möchten.

Sie können Guix eine Liste von URLs der Substitutserver mitgeben, damit es sie in der angegebenen Reihenfolge anfragt. Außerdem müssen Sie die öffentlichen Schlüssel der Substitutserver ausdrücklich autorisieren, damit Guix von ihnen signierte Substitute annimmt.

Auf Guix System ginge dies vonstatten, indem Sie die Konfiguration des `guix`-Dienstes modifizieren. Weil der `guix`-Dienst zu der Liste der vorgegebenen Dienste gehört, sowohl für `%base-services` als auch für `%desktop-services`, können Sie dessen Konfiguration mit `modify-services` ändern und dort die URLs und Substitutschlüssel eintragen, die Sie möchten (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639).

Ein Beispiel: Nehmen wir an, Sie möchten Substitute zusätzlich zu den vorgegebenen `ci.guix.gnu.org` und `bordeaux.guix.gnu.org` auch von `guix.example.org` beziehen und den Signierschlüssel dieses Servers autorisieren. Die Betriebssystemkonfiguration, die sich damit ergibt, wird ungefähr so aussehen:

```
(operating-system
  ;; ...
  (services
    ;; Wir nehmen hier '%desktop-services' als Grundlage; geben Sie
    ;; an deren Stelle die Liste der bei Ihnen benutzten Dienste an.
    (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 "./key.pub"))
                                     %default-authorized-guix-keys)))))))
```

Dabei gehen wir davon aus, dass der Signierschlüssel von `guix.example.org` in der Datei `key.pub` steht. Sobald Sie diese Änderung an der Datei mit Ihrer Betriebssystemkonfiguration vorgenommen haben (etwa `/etc/config.scm`), können Sie rekonfigurieren und den `guix-daemon`-Dienst (oder den ganzen Rechner) neu starten, damit die Änderung in Kraft tritt:

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

Wenn Sie Guix auf einer „Fremddistribution“ laufen lassen, würden Sie stattdessen nach den folgenden Schritten vorgehen, um Substitute von zusätzlichen Servern zu bekommen:

1. Bearbeiten Sie die Konfigurationsdatei für den `guix-daemon`; wenn Sie systemd benutzen, wäre das normalerweise `/etc/systemd/system/guix-daemon.service`. Schreiben Sie in die Konfigurationsdatei die Option `--substitute-urls` zur `guix-daemon`-Befehlszeile dazu und listen Sie dabei die gewünschten URLs auf (siehe [daemon-substitute-urls], Seite 19):

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

2. Starten Sie den Daemon neu. Bei systemd geht das so:

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

3. Autorisieren Sie den Schlüssel des neuen Servers (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74):

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

Wir nehmen auch hier an, dass `key.pub` den öffentlichen Schlüssel enthält, mit dem `guix.example.org` Substitute signiert.

Und wir sind fertig! Substitute werden bevorzugt von `https://guix.example.org` bezogen und `ci.guix.gnu.org` und dann `bordeaux.guix.gnu.org` bleiben notfalls als Reserve. Natürlich können Sie so viele Substitutserver auflisten, wie Sie wollen, allerdings kann das Erfragen von Substituten etwas länger dauern, wenn zu viele Server kontaktiert werden müssen.

Aber bedenken Sie, es gibt auch Situationen, wo man nur die URL eines Substitutservers hinzufügen will, *ohne* den Schlüssel zu autorisieren. Siehe Abschnitt 6.3.4 [Substitutauthentifizierung], Seite 59, um diese bestimmten Gründe zu verstehen.

6.3.4 Substitutauthentifizierung

Guix erkennt, wenn ein verfälschtes Substitut benutzt würde, und meldet einen Fehler. Ebenso werden Substitute ignoriert, die nich signiert sind, oder nicht mit einem in der ACL aufgelisteten Schlüssel signiert sind.

Es gibt nur eine Ausnahme: Wenn ein unautorisierter Server Substitute anbietet, die *Bit für Bit identisch* mit denen von einem autorisierten Server sind, können sie auch vom unautorisierten Server heruntergeladen werden. Zum Beispiel, angenommen wir haben zwei Substitutserver mit dieser Befehlszeilenoption ausgewählt:

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

Wenn in der ACL nur der Schlüssel für '`b.example.org`' aufgeführt wurde, aber '`a.example.org`' *exakt dieselben* Substitute anbietet, wird Guix auch Substitute von '`a.example.org`' herunterladen, weil es in der Liste zuerst kommt und als Spiegelserver für '`b.example.org`' aufgefasst werden kann. In der Praxis haben unabhängige Maschinen bei der Erstellung normalerweise dieselben Binärdateien als Ergebnis, dank bit-reproduzierbaren Erstellungen (siehe unten).

Wenn Sie HTTPS benutzen, wird das X.509-Zertifikat des Servers *nicht* validiert (mit anderen Worten, die Identität des Servers wird nicht authentifiziert), entgegen dem, was HTTPS-Clients wie Web-Browser normalerweise tun. Da Guix Substitutinformationen selbst überprüft, wie oben erklärt, wäre es unnötig (wohingegen mit X.509-Zertifikaten geprüft wird, ob ein Domain-Name zu öffentlichen Schlüsseln passt).

6.3.5 Proxy-Einstellungen

Substitute werden über HTTP oder HTTPS heruntergeladen. Die Umgebungsvariablen `http_proxy` und `https_proxy` können in der Umgebung von `guix-daemon` definiert werden und wirken sich dann auf das Herunterladen von Substituten aus. Beachten Sie, dass der Wert dieser Variablen in der Umgebung, in der `guix build`, `guix package` und andere Client-Befehle ausgeführt werden, *keine Rolle spielt*.

6.3.6 Fehler bei der Substitution

Selbst wenn ein Substitut für eine Ableitung verfügbar ist, schlägt die versuchte Substitution manchmal fehl. Das kann aus vielen Gründen geschehen: die Substitutserver könnten offline sein, das Substitut könnte kürzlich gelöscht worden sein, die Netzwerkverbindung könnte unterbrochen worden sein, usw.

Wenn Substitute aktiviert sind und ein Substitut für eine Ableitung zwar verfügbar ist, aber die versuchte Substitution fehlschlägt, kann Guix versuchen, die Ableitung lokal zu erstellen, je nachdem, ob `--fallback` übergeben wurde (siehe [common build option `--fallback`], Seite 190). Genauer gesagt, wird keine lokale Erstellung durchgeführt, solange kein `--fallback` angegeben wurde, und die Ableitung wird als Fehlschlag angesehen. Wenn `--fallback` übergeben wurde, wird Guix versuchen, die Ableitung lokal zu erstellen, und ob die Ableitung erfolgreich ist oder nicht, hängt davon ab, ob die lokale Erstellung erfolgreich ist oder nicht. Beachten Sie, dass, falls Substitute ausgeschaltet oder erst gar kein Substitut verfügbar ist, *immer* eine lokale Erstellung durchgeführt wird, egal ob `--fallback` übergeben wurde oder nicht.

Um eine Vorstellung zu bekommen, wie viele Substitute gerade verfügbar sind, können Sie den Befehl `guix weather` benutzen (siehe Abschnitt 10.15 [Aufruf von guix weather], Seite 242). Dieser Befehl zeigt Statistiken darüber an, wie es um die von einem Server verfügbaren Substitute steht.

6.3.7 Vom Vertrauen gegenüber Binärdateien

Derzeit hängt die Kontrolle jedes Individuums über seine Rechner von Institutionen, Unternehmen und solchen Gruppierungen ab, die über genug Macht und Entschlusskraft verfügen, die Rechnerinfrastruktur zu sabotieren und ihre Schwachstellen auszunutzen. Auch wenn es bequem ist, Substitute zu benutzen, ermuntern wir Nutzer, auch selbst Erstellungen durchzuführen oder gar ihre eigene Erstellungsfarm zu betreiben, damit die vom Guix-Projekt betriebenen Substitutserver ein weniger interessantes Ziel werden. Eine Art, uns zu helfen, ist, die von Ihnen erstellte Software mit dem Befehl `guix publish` zu veröffentlichen, damit andere eine größere Auswahl haben, von welchem Server sie Substitute beziehen möchten (siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233).

Guix hat die richtigen Grundlagen, um die Reproduzierbarkeit von Erstellungen zu maximieren (siehe Abschnitt 6.1 [Funktionalitäten], Seite 44). In den meisten Fällen sollten unabhängige Erstellungen eines bestimmten Pakets zu bitweise identischen Ergebnissen führen. Wir können also mit Hilfe einer vielschichtigen Menge an unabhängigen Paketerstellungen die Integrität unseres Systems besser gewährleisten. Der Befehl `guix challenge` hat das Ziel, Nutzern zu ermöglichen, Substitutserver zu beurteilen, und Entwickler dabei zu unterstützen, nichtdeterministische Paketerstellungen zu finden (siehe Abschnitt 10.12 [Aufruf von guix challenge], Seite 238). Ebenso ermöglicht es die Befehlszeilenoption `--check` von `guix build`, dass Nutzer bereits installierte Substitute auf Unverfälschtheit zu prüfen, indem sie durch lokales Erstellen nachgebildet werden (siehe [build-check], Seite 201).

In Zukunft wollen wir, dass Sie mit Guix Binärdateien von Netzwerkteilnehmer zu Netzwerkteilnehmer („peer-to-peer“) veröffentlichen und empfangen können. Wenn Sie mit uns dieses Projekt diskutieren möchten, kommen Sie auf unsere Mailing-Liste `guix-devel@gnu.org`.

6.4 Pakete mit mehreren Ausgaben.

Oft haben in Guix definierte Pakete eine einzige *Ausgabe* – d.h. aus dem Quellpaket entsteht genau ein Verzeichnis im Store. Wenn Sie `guix install glibc` ausführen, wird die Standard-Paketausgabe des GNU-libc-Pakets installiert; die Standardausgabe wird `out` genannt, aber ihr Name kann weggelassen werden, wie Sie am obigen Befehl sehen. In diesem

speziellen Fall enthält die Standard-Paketausgabe von `glibc` alle C-Headerdateien, gemeinsamen Bibliotheken („Shared Libraries“), statischen Bibliotheken („Static Libraries“), Dokumentation für Info sowie andere zusätzliche Dateien.

Manchmal ist es besser, die verschiedenen Arten von Dateien, die aus einem einzelnen Quellpaket hervorgehen, in getrennte Ausgaben zu unterteilen. Zum Beispiel installiert die GLib-C-Bibliothek (die von GTK und damit zusammenhängenden Paketen benutzt wird) mehr als 20 MiB an HTML-Seiten mit Referenzdokumentation. Um den Nutzern, die das nicht brauchen, Platz zu sparen, wird die Dokumentation in einer separaten Ausgabe abgelegt, genannt `doc`. Um also die Hauptausgabe von GLib zu installieren, zu der alles außer der Dokumentation gehört, ist der Befehl:

```
guix install glib
```

Der Befehl, um die Dokumentation zu installieren, ist:

```
guix install glib:doc
```

Manche Pakete installieren Programme mit unterschiedlich großem „Abhängigkeiten-Fußabdruck“. Zum Beispiel installiert das Paket WordNet sowohl Befehlszeilenwerkzeuge als auch grafische Benutzerschnittstellen (GUIs). Erstere hängen nur von der C-Bibliothek ab, während Letztere auch von Tcl/Tk und den zu Grunde liegenden X-Bibliotheken abhängen. Jedenfalls belassen wir deshalb die Befehlszeilenwerkzeuge in der Standard-Paketausgabe, während sich die GUIs in einer separaten Ausgabe befinden. So können Benutzer, die die GUIs nicht brauchen, Platz sparen. Der Befehl `guix size` kann dabei helfen, solche Situationen zu erkennen (siehe Abschnitt 10.9 [Aufruf von guix size], Seite 226). `guix graph` kann auch helfen (siehe Abschnitt 10.10 [Aufruf von guix graph], Seite 228).

In der GNU-Distribution gibt es viele solche Pakete mit mehreren Ausgaben. Andere Konventionen für Ausgabenamen sind zum Beispiel `lib` für Bibliotheken und eventuell auch ihre Header-Dateien,, `bin` für eigenständige Programme und `debug` für Informationen zur Fehlerbehandlung (siehe Kapitel 17 [Dateien zur Fehlersuche installieren], Seite 692). Die Ausgaben eines Pakets stehen in der dritten Spalte der Anzeige von `guix package --list-available` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

6.5 `guix gc` aufrufen

Pakete, die zwar installiert sind, aber nicht benutzt werden, können vom *Müllsammler* entfernt werden. Mit dem Befehl `guix gc` können Benutzer den Müllsammler ausdrücklich aufrufen, um Speicher im Verzeichnis `/gnu/store` freizugeben. Dies ist der *einzige* Weg, Dateien aus `/gnu/store` zu entfernen – das manuelle Entfernen von Dateien kann den Store irreparabel beschädigen!

Der Müllsammler kennt eine Reihe von *Wurzeln*: Jede Datei in `/gnu/store`, die von einer Wurzel aus erreichbar ist, gilt als *lebendig* und kann nicht entfernt werden; jede andere Datei gilt als *tot* und ist ein Kandidat, gelöscht zu werden. Die Menge der Müllsammlerwurzeln (kurz auch „GC-Wurzeln“, von englisch „Garbage Collector“) umfasst Standard-Benutzerprofile; standardmäßig werden diese Müllsammlerwurzeln durch symbolische Verknüpfungen in `/var/guix/gcroots` dargestellt. Neue Müllsammlerwurzeln können zum Beispiel mit `guix build --root` festgelegt werden (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189). Der Befehl `guix gc --list-roots` listet sie auf.

Bevor Sie mit `guix gc --collect-garbage` Speicher freimachen, wollen Sie vielleicht alte Generationen von Benutzerprofilen löschen, damit alte Paketerstellungen von diesen Ge-

nerationen entfernt werden können. Führen Sie dazu `guix package --delete-generations` aus (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

Unsere Empfehlung ist, dass Sie den Müllsammler regelmäßig laufen lassen und wenn Sie wenig freien Speicherplatz zur Verfügung haben. Um zum Beispiel sicherzustellen, dass Sie mindestens 5 GB auf Ihrer Platte zur Verfügung haben, benutzen Sie einfach:

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

Es ist völlig sicher, dafür eine nicht interaktive, regelmäßige Auftragsausführung vorzugeben (siehe Abschnitt 12.9.2 [Geplante Auftragsausführung], Seite 301, für eine Erklärung, wie man das tun kann). `guix gc` ohne Befehlszeilenargumente auszuführen, lässt so viel Müll wie möglich sammeln, aber das ist oft nicht, was man will, denn so muss man unter Umständen Software erneut erstellen oder erneut herunterladen, weil der Müllsammler sie als „tot“ ansieht, sie aber zur Erstellung anderer Software wieder gebraucht wird – das trifft zum Beispiel auf die Compiler-Toolchain zu.

Der Befehl `guix gc` hat drei Arbeitsmodi: Er kann benutzt werden, um als Müllsammler tote Dateien zu entfernen (das Standardverhalten), um ganz bestimmte, angegebene Datein zu löschen (mit der Befehlszeilenoption `--delete`), um Müllsammlerinformationen auszugeben oder fortgeschrittenere Anfragen zu verarbeiten. Die Müllsammler-Befehlszeilenoptionen sind wie folgt:

`--collect-garbage[=`*Minimum*`]`
`-C [`*Minimum*`]`

Lässt Müll sammeln – z.B. nicht erreichbare Dateien in `/gnu/store` und seinen Unterverzeichnissen. Wird keine andere Befehlszeilenoption angegeben, wird standardmäßig diese durchgeführt.

Wenn ein *Minimum* angegeben wurde, hört der Müllsammler auf, sobald *Minimum* Bytes gesammelt wurden. Das *Minimum* kann die Anzahl der Bytes bezeichnen oder mit einer Einheit als Suffix versehen sein, wie etwa `MiB` für Mebibytes und `GB` für Gigabytes (siehe Abschnitt “Block size” in *GNU Coreutils*).

Wird kein *Minimum* angegeben, sammelt der Müllsammler allen Müll.

`--free-space=`*Menge*
`-F` *Menge*

Sammelt Müll, bis die angegebene *Menge* an freiem Speicher in `/gnu/store` zur Verfügung steht, falls möglich; die *Menge* ist eine Speichergröße wie `500MiB`, wie oben beschrieben.

Wenn die angegebene *Menge* oder mehr bereits in `/gnu/store` frei verfügbar ist, passiert nichts.

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

Bevor der Müllsammelvorgang beginnt, werden hiermit alle Generationen von allen Benutzerprofilen und von allen Persönlichen Umgebungen gelöscht, die älter sind als die angegebene *Dauer*; wird es als Administratornutzer „root“ ausgeführt, geschieht dies mit den Profilen *von allen Benutzern*.

Zum Beispiel löscht der folgende Befehl alle Generationen Ihrer Profile, die älter als zwei Monate sind (ausgenommen die momentanen Generationen), und

schmeißt dann den Müllsammler an, um Platz freizuräumen, bis mindestens 10 GiB verfügbar sind:

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

`--delete`
`-D`
: Versucht, alle als Argumente angegebenen Dateien oder Verzeichnisse im Store zu löschen. Dies schlägt fehl, wenn manche der Dateien oder Verzeichnisse nicht im Store oder noch immer lebendig sind.

`--list-failures`
: Store-Objekte auflisten, die zwischengespeicherten Erstellungsfehlern entsprechen.

  Hierbei wird nichts ausgegeben, sofern der Daemon nicht mit `--cache-failures` gestartet wurde (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

`--list-roots`
: Die Müllsammlerwurzeln auflisten, die dem Nutzer gehören. Wird der Befehl als Administratornutzer ausgeführt, werden *alle* Müllsammlerwurzeln aufgelistet.

`--list-busy`
: Solche Store-Objekte auflisten, die von aktuell laufenden Prozessen benutzt werden. Diese Store-Objekte werden praktisch wie Müllsammlerwurzeln behandelt; sie können nicht gelöscht werden.

`--clear-failures`
: Die angegebenen Store-Objekte aus dem Zwischenspeicher für fehlgeschlagene Erstellungen entfernen.

  Auch diese Option macht nur Sinn, wenn der Daemon mit `--cache-failures` gestartet wurde. Andernfalls passiert nichts.

`--list-dead`
: Zeigt die Liste toter Dateien und Verzeichnisse an, die sich noch im Store befinden – das heißt, Dateien, die von keiner Wurzel mehr erreichbar sind.

`--list-live`
: Zeige die Liste lebendiger Store-Dateien und -Verzeichnisse.

Außerdem können Referenzen unter bestehenden Store-Dateien gefunden werden:

`--references`
`--referrers`
: Listet die referenzierten bzw. sie referenzierenden Objekte der angegebenen Store-Dateien auf.

`--requisites`
`-R`
: Listet alle Voraussetzungen der als Argumente übergebenen Store-Dateien auf. Voraussetzungen sind die Store-Dateien selbst, ihre Referenzen sowie die Referenzen davon, rekursiv. Mit anderen Worten, die zurückgelieferte Liste ist der *transitive Abschluss* dieser Store-Dateien.

  Der Abschnitt Abschnitt 10.9 [Aufruf von guix size], Seite 226, erklärt ein Werkzeug, um den Speicherbedarf des Abschlusses eines Elements zu ermitteln. Siehe

Abschnitt 10.10 [Aufruf von guix graph], Seite 228, für ein Werkzeug, um den Referenzgraphen zu veranschaulichen.

`--derivers`
: Liefert die Ableitung(en), die zu den angegebenen Store-Objekten führen (siehe Abschnitt 9.10 [Ableitungen], Seite 167).

  Zum Beispiel liefert dieser Befehl:

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

  die `.drv`-Datei(en), die zum in Ihrem Profil installierten `emacs`-Paket führen.

  Beachten Sie, dass es auch sein kann, dass keine passenden `.drv`-Dateien existieren, zum Beispiel wenn diese Dateien bereits dem Müllsammler zum Opfer gefallen sind. Es kann auch passieren, dass es mehr als eine passende `.drv` gibt, bei Ableitungen mit fester Ausgabe.

Zuletzt können Sie mit folgenden Befehlszeilenoptionen die Integrität des Stores prüfen und den Plattenspeicherverbrauch im Zaum halten.

`--verify[=`*Optionen*`]`
: Die Integrität des Stores verifizieren

  Standardmäßig wird sichergestellt, dass alle Store-Objekte, die in der Datenbank des Daemons als gültig markiert wurden, auch tatsächlich in `/gnu/store` existieren.

  Wenn angegeben, müssen die *Optionen* eine kommagetrennte Liste aus mindestens einem der Worte `contents` und `repair` sein.

  Wenn Sie `--verify=contents` übergeben, berechnet der Daemon den Hash des Inhalts jedes Store-Objekts und vergleicht ihn mit dem Hash in der Datenbank. Sind die Hashes ungleich, wird eine Datenbeschädigung gemeldet. Weil dabei *alle Dateien im Store* durchlaufen werden, kann der Befehl viel Zeit brauchen, besonders auf Systemen mit langsamer Platte.

  Mit `--verify=repair` oder `--verify=contents,repair` versucht der Daemon, beschädigte Store-Objekte zu reparieren, indem er Substitute für selbige herunterlädt (siehe Abschnitt 6.3 [Substitute], Seite 56). Weil die Reparatur nicht atomar und daher womöglich riskant ist, kann nur der Systemadministrator den Befehl benutzen. Eine weniger aufwendige Alternative, wenn Sie wissen, welches Objekt beschädigt ist, ist, `guix build --repair` zu benutzen (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189).

`--optimize`
: Den Store durch Nutzung harter Verknüpfungen für identische Dateien optimieren – mit anderen Worten wird der Store *dedupliziert*.

  Der Daemon führt Deduplizierung automatisch nach jeder erfolgreichen Erstellung und jedem Importieren eines Archivs durch, sofern er nicht mit `--disable-deduplication` (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18) gestartet wurde. Diese Befehlszeilenoption brauchen Sie also in erster Linie dann, wenn der Daemon zuvor mit `--disable-deduplication` gestartet worden ist.

`--vacuum-database`
: Für Guix wird eine SQLite-Datenbank verwendet, um über die Objekte im Store Buch zu führen (siehe Abschnitt 9.9 [Der Store], Seite 165). Mit der Zeit

kann die Datenbank jedoch eine gigantische Größe erreichen und fragmentieren. Deshalb kann es sich lohnen, den freien Speicher aufzuräumen und nur teilweise genutzte Datenbankseiten, die beim Löschen von Paketen bzw. nach dem Müllsammeln entstanden sind, zusammenzulegen. Führen Sie `sudo guix gc --vacuum-database` aus, und die Datenbank wird gesperrt und eine `VACUUM`-Operation auf dem Store durchgeführt, welche die Datenbank defragmentiert und leere Seiten wegschafft. Wenn alles fertig ist, wird die Datenbank entsperrt.

6.6 `guix pull` aufrufen

Nach der Installation oder Aktualisierung wird stets die neueste Version von Paketen verwendet, die in der aktuell installierten Distribution verfügbar ist. Um die Distribution und die Guix-Werkzeuge zu aktualisieren, führen Sie `guix pull` aus. Der Befehl lädt den neuesten Guix-Quellcode einschließlich Paketbeschreibungen herunter und installiert ihn. Quellcode wird aus einem Git-Repository (`https://git-scm.com/book/de/`) geladen, standardmäßig dem offiziellen Repository von GNU Guix, was Sie aber auch ändern können. `guix pull` stellt sicher, dass der heruntergeladene Code *authentisch* ist, indem es überprüft, dass die Commits durch Guix-Entwickler signiert worden sind.

Genauer gesagt lädt `guix pull` Code von den *Kanälen* herunter (siehe Kapitel 7 [Kanäle], Seite 77), die an einer der folgenden Stellen, in dieser Reihenfolge, angegeben wurden:

1. die Befehlszeilenoption `--channels`,
2. die Datei `~/.config/guix/channels.scm` des Benutzers,
3. die systemweite `/etc/guix/channels.scm`-Datei,
4. die eingebauten vorgegebenen Kanäle, wie sie in der Variablen `%default-channels` festgelegt sind.

Danach wird `guix package` Pakete und ihre Versionen entsprechend der gerade heruntergeladenen Kopie von Guix benutzen. Nicht nur das, auch alle Guix-Befehle und Scheme-Module werden aus der neuesten Version von Guix kommen. Neue `guix`-Unterbefehle, die durch die Aktualisierung hinzugekommen sind, werden also auch verfügbar.

Jeder Nutzer kann seine Kopie von Guix mittels `guix pull` aktualisieren, wodurch sich nur für den Nutzer etwas verändert, der `guix pull` ausgeführt hat. Wenn also zum Beispiel der Administratornutzer `root` den Befehl `guix pull` ausführt, hat das keine Auswirkungen auf die für den Benutzer `alice` sichtbare Guix-Version, und umgekehrt.

Das Ergebnis von `guix pull` ist ein als `~/.config/guix/current` verfügbares *Profil* mit dem neuesten Guix. Stellen Sie sicher, dass es am Anfang Ihres Suchpfades steht, damit Sie auch wirklich das neueste Guix und sein Info-Handbuch sehen (siehe Kapitel 14 [Dokumentation], Seite 680):

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

Die Befehlszeilenoption `--list-generations` oder kurz `-l` listet ältere von `guix pull` erzeugte Generationen auf, zusammen mit Informationen zu deren Provenienz.

```
$ guix pull -l
Generation 1 10. Juni 2018 00:18:18
  guix 65956ad
```

```
      Repository-URL: https://git.savannah.gnu.org/git/guix.git
      Branch: origin/master
      Commit: 65956ad3526ba09e1f7a40722c96c6ef7c0936fe

  Generation 2 11. Juni 2018 11:02:49
    guix e0cc7f6
      Repository-URL: https://git.savannah.gnu.org/git/guix.git
      Branch: origin/master
      Commit: e0cc7f669bec22c37481dd03a7941c7d11a64f1d

  Generation 3 13. Juni 2018 23:31:07 (aktuell)
    guix 844cc1c
      Repository-URL: https://git.savannah.gnu.org/git/guix.git
      Branch: origin/master
      Commit: 844cc1c8f394f03b404c5bb3aee086922373490c
```

Im Abschnitt Abschnitt 6.9 [Aufruf von guix describe], Seite 72, werden andere Möglichkeiten erklärt, sich den momentanen Zustand von Guix beschreiben zu lassen.

Das Profil `~/.config/guix/current` verhält sich genau wie die durch `guix package` erzeugten Profile (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Das bedeutet, Sie können seine Generationen auflisten und es auf die vorherige Generation – also das vorherige Guix – zurücksetzen und so weiter:

```
$ guix pull --roll-back
Von Generation „3“ zu „2“ gewechselt
$ guix pull --delete-generations=1
/var/guix/profiles/per-user/charlie/current-guix-1-link wird gelöscht
```

Sie können auch `guix package` benutzen (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45), um das Profil zu verwalten, indem Sie es explizit angeben.:

```
$ guix package -p ~/.config/guix/current --roll-back
switched from generation 3 to 2
$ guix package -p ~/.config/guix/current --delete-generations=1
deleting /var/guix/profiles/per-user/charlie/current-guix-1-link
```

Der Befehl `guix pull` wird in der Regel ohne Befehlszeilenargumente aufgerufen, aber er versteht auch folgende Befehlszeilenoptionen:

`--url=`*URL*
`--commit=`*Commit*
`--branch=`*Branch*
: Code wird für den `guix`-Kanal von der angegebenen *URL* für den angegebenen *Commit* (eine gültige Commit-ID, dargestellt als hexadezimale Zeichenkette oder Namen eines Git-Tags) oder *Branch* heruntergeladen.

  Diese Befehlszeilenoptionen sind manchmal bequemer, aber Sie können Ihre Konfiguration auch in der Datei `~/.config/guix/channels.scm` oder über die Option `--channels` angeben (siehe unten).

`--channels=`*Datei*
`-C` *Datei*
: Die Liste der Kanäle aus der angegebenen *Datei* statt aus `~/.config/guix/channels.scm` oder aus `/etc/guix/channels.scm`

auslesen. Die *Datei* muss Scheme-Code enthalten, der zu einer Liste von Kanalobjekten ausgewertet wird. Siehe Kapitel 7 [Kanäle], Seite 77, für nähere Informationen.

`--news`
`-N`
: Neuigkeiten anzeigen, die Kanalautoren für ihre Nutzer geschrieben haben, um sie über Veränderungen seit der vorigen Generation in Kenntnis zu setzen (siehe Kapitel 7 [Kanäle], Seite 77). Wenn Sie `--details` übergeben, werden auch neue und aktualisierte Pakete gezeigt.

  Sie können sich mit `guix pull -l` diese Informationen für vorherige Generationen ansehen.

`--list-generations[=`*Muster*`]`
`-l [`*Muster*`]`
: Alle Generationen von `~/.config/guix/current` bzw., wenn ein *Muster* angegeben wird, die dazu passenden Generationen auflisten. Die Syntax für das *Muster* ist dieselbe wie bei `guix package --list-generations` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

  Nach Voreinstellung werden Informationen über die benutzten Kanäle in der jeweiligen Version sowie die zugehörigen Neuigkeiten ausgegeben. Wenn Sie `--details` übergeben, wird außerdem die Liste der neu hinzugefügten oder aktualisierten Pakete in jeder Generation verglichen mit der vorigen angezeigt.

`--details`
: Bei `--list-generations` oder `--news` wird Guix angewiesen, mehr Informationen über die Unterschiede zwischen aufeinanderfolgenden Generationen anzuzeigen – siehe oben.

`--roll-back`
: Zur vorherigen *Generation* von `~/.config/guix/current` zurückwechseln – d.h. die letzte Transaktion rückgängig machen.

`--switch-generation=`*Muster*
`-S` *Muster*
: Wechselt zu der bestimmten Generation, die durch das *Muster* bezeichnet wird.

  Als *Muster* kann entweder die Nummer einer Generation oder eine Nummer mit vorangestelltem „`+`“ oder „`-`“ dienen. Letzteres springt die angegebene Anzahl an Generationen vor oder zurück. Zum Beispiel kehrt `--switch-generation=+1` nach einem Zurücksetzen wieder zur neueren Generation zurück.

`--delete-generations[=`*Muster*`]`
`-d [`*Muster*`]`
: Wird kein *Muster* angegeben, werden alle Generationen außer der aktuellen entfernt.

  Dieser Befehl akzeptiert dieselben Muster wie `--list-generations`. Wenn ein *Muster* angegeben wird, werden die passenden Generationen gelöscht. Wenn das *Muster* für eine Zeitdauer steht, werden diejenigen Generationen gelöscht, die *älter* als die angegebene Dauer sind. Zum Beispiel löscht `--delete-generations=1m` die Generationen, die mehr als einen Monat alt sind.

Falls die aktuelle Generation zum Muster passt, wird sie *nicht* gelöscht.

Beachten Sie, dass Sie auf gelöschte Generationen nicht zurückwechseln können. Dieser Befehl sollte also nur mit Vorsicht benutzt werden.

Im Abschnitt Abschnitt 6.9 [Aufruf von guix describe], Seite 72, wird eine Möglichkeit erklärt, sich Informationen nur über die aktuelle Generation anzeigen zu lassen.

`--profile=`*Profil*
`-p` *Profil* Auf *Profil* anstelle von `~/.config/guix/current` arbeiten.

`--dry-run`
`-n` Anzeigen, welche(r) Commit(s) für die Kanäle benutzt würde(n) und was jeweils erstellt oder substituiert würde, ohne es tatsächlich durchzuführen.

`--allow-downgrades`
Beziehen einer älteren oder mit der bisheriger Version eines Kanals *nicht* zusammenhängenden Version zulassen.

Nach Voreinstellung schützt `guix pull` den Nutzer vor Herabstufungsangriffen („Downgrade Attacks“). Damit werden Versuche bezeichnet, jemanden eine frühere oder unzusammenhängende Version des Git-Repositorys eines Kanals benutzen zu lassen, wodurch diese Person dazu gebracht werden kann, ältere Versionen von Softwarepaketen mit bekannten Schwachstellen zu installieren.

> **Anmerkung:** Sie sollten verstehen, was es für die Sicherheit Ihres Rechners bedeutet, ehe Sie `--allow-downgrades` benutzen.

`--disable-authentication`
Beziehen von Kanalcode ohne Authentifizierung zulassen.

Nach Voreinstellung wird durch `guix pull` von Kanälen heruntergeladener Code darauf überprüft, dass deren Commits durch autorisierte Entwickler signiert worden sind; andernfalls wird ein Fehler gemeldet. Mit dieser Befehlszeilenoption können Sie anweisen, keine solche Verifikation durchzuführen.

> **Anmerkung:** Sie sollten verstehen, was es für die Sicherheit Ihres Rechners bedeutet, ehe Sie `--disable-authentication` benutzen.

`--system=`*System*
`-s` *System* Versuchen, für die angegebene Art von *System* geeignete Binärdateien zu erstellen – z.B. `i686-linux` – statt für die Art von System, das die Erstellung durchführt.

`--bootstrap`
Das neueste Guix mit dem Bootstrap-Guile erstellen. Diese Befehlszeilenoption ist nur für Guix-Entwickler von Nutzen.

Mit Hilfe von *Kanälen* können Sie `guix pull` anweisen, von welchem Repository und welchem Branch Guix aktualisiert werden soll, sowie von welchen *weiteren* Repositorys Paketmodule bezogen werden sollen. Im Abschnitt Kapitel 7 [Kanäle], Seite 77, finden Sie nähere Informationen.

Außerdem unterstützt `guix pull` alle gemeinsamen Erstellungsoptionen (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

6.7 `guix time-machine` aufrufen

Der Befehl `guix time-machine` erleichtert den Zugang zu anderen Versionen von Guix. Damit können ältere Versionen von Paketen installiert werden oder eine Berechnung in einer identischen Umgebung reproduziert werden. Die zu benutzende Guix-Version wird über eine Commit-Angabe oder eine Kanalbeschreibungsdatei, wie sie durch `guix describe` erzeugt werden kann, festgelegt (siehe Abschnitt 6.9 [Aufruf von guix describe], Seite 72).

Sagen wir, Sie würden gerne in der Zeit zurückreisen zu den Tagen um November 2020, als die Version 1.2.0 von Guix veröffentlicht worden ist, und außerdem möchten Sie nach Ihrer Ankunft den damaligen `guile`-Befehl ausführen:

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

Mit obigem Befehl wird Guix 1.2.0 geladen und daraufhin dessen Befehl `guix environment` ausgeführt, um eine Umgebung, in einen Container verpackt, zu betreten, wo dann `guile` gestartet wird (`guix environment` ist mittlerweile Teil von `guix shell`, siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86). Es ist so, als würden Sie einen DeLorean fahren[1]! Der erste Aufruf von `guix time-machine` kann lange dauern, weil vielleicht viele Pakete heruntergeladen oder sogar erstellt werden müssen, aber das Ergebnis bleibt in einem Zwischenspeicher und danach geschehen Befehle für denselben Commit fast sofort.

> **Anmerkung:** Vergangene Commits zu Guix sind unveränderlich und `guix time-machine` bietet genau dieselbe Software, wie sie in einer damaligen Guix-Version bestanden hat. Demzufolge werden auch *keine* Sicherheitspatches für alte Versionen von Guix oder dessen Kanäle nachgeliefert. Unbedachter Gebrauch der `guix time-machine` lässt Sicherheitsschwachstellen freien Raum. Siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65.

Die allgemeine Syntax lautet:

```
guix time-machine Optionen... -- Befehl  Argument...
```

Dabei werden der *Befehl* und jedes *Argument*... unverändert an den `guix`-Befehl der angegebenen Version übergeben. Die *Optionen*, die die Version definieren, sind dieselben wie bei `guix pull` (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65):

`--url=`*URL*
`--commit=`*Commit*
`--branch=`*Branch*
: Den `guix`-Kanal von der angegebenen *URL* benutzen, für den angegebenen *Commit* (eine gültige Commit-ID, dargestellt als hexadezimale Zeichenkette oder Namen eines Git-Tags) oder *Branch*.

`--channels=`*Datei*
`-C` *Datei*
: Die Liste der Kanäle aus der angegebenen *Datei* auslesen. Die *Datei* muss Scheme-Code enthalten, der zu einer Liste von Kanalobjekten ausgewertet wird. Siehe Kapitel 7 [Kanäle], Seite 77, für nähere Informationen.

Wie bei `guix pull` wird in Ermangelung anderer Optionen der letzte Commit auf dem master-Branch benutzt. Mit dem Befehl

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

[1] Wenn Sie DeLorean nicht kennen, sollten Sie eine Zeitreise in die 1980er in Betracht ziehen.

wird dementsprechend das Paket `hello` erstellt, so wie es auf dem master-Branch definiert ist, was in der Regel einer neueren Guix-Version entspricht als der, die Sie installiert haben. Zeitreisen funktionieren also in beide Richtungen!

Beachten Sie, dass durch `guix time-machine` Erstellungen von Kanälen und deren Abhängigkeiten ausgelöst werden können, welche durch die gemeinsamen Erstellungsoptionen gesteuert werden können (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

6.8 Untergeordnete

> **Anmerkung:** Die hier beschriebenen Funktionalitäten sind in der Version 1.4.0 bloß eine „Technologie-Vorschau“, daher kann sich die Schnittstelle in Zukunft noch ändern.

Manchmal könnten Sie Pakete aus der gerade laufenden Fassung von Guix mit denen mischen wollen, die in einer anderen Guix-Version verfügbar sind. Guix-*Untergeordnete* ermöglichen dies, indem Sie verschiedene Guix-Versionen beliebig mischen können.

Aus technischer Sicht ist ein „Untergeordneter“ im Kern ein separater Guix-Prozess, der über eine REPL (siehe Abschnitt 9.13 [Aufruf von guix repl], Seite 185) mit Ihrem Haupt-Guix-Prozess verbunden ist. Das Modul `(guix inferior)` ermöglicht es Ihnen, Untergeordnete zu erstellen und mit ihnen zu kommunizieren. Dadurch steht Ihnen auch eine hochsprachliche Schnittstelle zur Verfügung, um die von einem Untergeordneten angebotenen Pakete zu durchsuchen und zu verändern – *untergeordnete Pakete*.

In Kombination mit Kanälen (siehe Kapitel 7 [Kanäle], Seite 77) bieten Untergeordnete eine einfache Möglichkeit, mit einer anderen Version von Guix zu interagieren. Nehmen wir zum Beispiel an, Sie wollen das aktuelle `guile`-Paket in Ihr Profil installieren, zusammen mit dem `guile-json`, wie es in einer früheren Guix-Version existiert hat – vielleicht weil das neuere `guile-json` eine inkompatible API hat und Sie daher Ihren Code mit der alten API benutzen möchten. Dazu könnten Sie ein Manifest für `guix package --manifest` schreiben (siehe Abschnitt 9.4 [Manifeste verfassen], Seite 125); in diesem Manifest würden Sie einen Untergeordneten für diese alte Guix-Version erzeugen, für die Sie sich interessieren, und aus diesem Untergeordneten das `guile-json`-Paket holen:

```
(use-modules (guix inferior) (guix channels)
             (srfi srfi-1))   ;für die Prozedur 'first'

(define channels
  ;; Dies ist die alte Version, aus der wir
  ;; guile-json extrahieren möchten.
  (list (channel
         (name 'guix)
         (url "https://git.savannah.gnu.org/git/guix.git")
         (commit
          "65956ad3526ba09e1f7a40722c96c6ef7c0936fe"))))

(define inferior
  ;; Ein Untergeordneter, der obige Version repräsentiert.
  (inferior-for-channels channels))
```

```
;; Daraus erzeugen wir jetzt ein Manifest mit dem aktuellen
;; „guile“-Paket und dem alten „guile-json“-Paket.
(packages->manifest
 (list (first (lookup-inferior-packages inferior "guile-json"))
       (specification->package "guile")))
```

Bei seiner ersten Ausführung könnte für `guix package --manifest` erst der angegebene Kanal erstellt werden müssen, bevor der Untergeordnete erstellt werden kann; nachfolgende Durchläufe sind wesentlich schneller, weil diese Guix-Version bereits zwischengespeichert ist.

Folgende Prozeduren werden im Modul `(guix inferior)` angeboten, um einen Untergeordneten zu öffnen:

`inferior-for-channels` *Kanäle* [*#:cache-directory*] [*#:ttl*] [Scheme-Prozedur]
Liefert einen Untergeordneten für die *Kanäle*, einer Liste von Kanälen. Dazu wird der Zwischenspeicher im Verzeichnis *cache-directory* benutzt, dessen Einträge nach *ttl* Sekunden gesammelt werden dürfen. Mit dieser Prozedur wird eine neue Verbindung zum Erstellungs-Daemon geöffnet.

Als Nebenwirkung erstellt oder substituiert diese Prozedur unter Umständen Binärdateien für die *Kanäle*, was einige Zeit in Anspruch nehmen kann.

`open-inferior` *Verzeichnis* [*#:command* "*bin/guix*"] [Scheme-Prozedur]
Öffnet das untergeordnete Guix mit dem Befehl *command* im angegebenen *Verzeichnis* durch Ausführung von *`Verzeichnis/command`* `repl` oder entsprechend. Liefert `#f`, wenn der Untergeordnete nicht gestartet werden konnte.

Die im Folgenden aufgeführten Prozeduren ermöglichen es Ihnen, untergeordnete Pakete abzurufen und zu verändern.

`inferior-packages` *Untergeordneter* [Scheme-Prozedur]
Liefert die Liste der Pakete in *Untergeordneter*.

`lookup-inferior-packages` *Untergeordneter Name* [Scheme-Prozedur]
[*Version*] *Liefert die sortierte Liste der untergeordneten Pakete in* *Untergeordneter*, die zum Muster *Name* in *Untergeordneter* passen, dabei kommen höhere Versionsnummern zuerst. Wenn *Version* auf wahr gesetzt ist, werden nur Pakete geliefert, deren Versionsnummer mit dem Präfix *Version* beginnt.

`inferior-package?` *Objekt* [Scheme-Prozedur]
Liefert wahr, wenn das *obj* ein Untergeordneter ist.

`inferior-package-name` *Paket* [Scheme-Prozedur]
`inferior-package-version` *Paket* [Scheme-Prozedur]
`inferior-package-synopsis` *Paket* [Scheme-Prozedur]
`inferior-package-description` *Paket* [Scheme-Prozedur]
`inferior-package-home-page` *Paket* [Scheme-Prozedur]
`inferior-package-location` *Paket* [Scheme-Prozedur]

`inferior-package-inputs` *Paket* [Scheme-Prozedur]
`inferior-package-native-inputs` *Paket* [Scheme-Prozedur]
`inferior-package-propagated-inputs` *Paket* [Scheme-Prozedur]
`inferior-package-transitive-propagated-inputs` *Paket* [Scheme-Prozedur]
`inferior-package-native-search-paths` *Paket* [Scheme-Prozedur]
`inferior-package-transitive-native-search-paths` *Paket* [Scheme-Prozedur]
`inferior-package-search-paths` *Paket* [Scheme-Prozedur]

Diese Prozeduren sind das Gegenstück zu den Zugriffsmethoden des Verbunds „package“ für Pakete (siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113). Die meisten davon funktionieren durch eine Abfrage auf dem Untergeordneten, von dem das *Paket* kommt, weshalb der Untergeordnete noch lebendig sein muss, wenn Sie diese Prozeduren aufrufen.

Untergeordnete Pakete können transparent wie jedes andere Paket oder dateiartige Objekt in G-Ausdrücken verwendet werden (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175). Sie werden auch transparent wie reguläre Pakete von der Prozedur `packages->manifest` behandelt, welche oft in Manifesten benutzt wird (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Somit können Sie ein untergeordnetes Paket ziemlich überall dort verwenden, wo Sie ein reguläres Paket einfügen würden: in Manifesten, im Feld `packages` Ihrer `operating-system`-Deklaration und so weiter.

6.9 `guix describe` aufrufen

Sie könnten sich des Öfteren Fragen stellen wie: „Welche Version von Guix benutze ich gerade?“ oder „Welche Kanäle benutze ich?“ Diese Informationen sind in vielen Situationen nützlich: wenn Sie eine Umgebung auf einer anderen Maschine oder mit einem anderen Benutzerkonto *nachbilden* möchten, wenn Sie einen Fehler melden möchten, wenn Sie festzustellen versuchen, welche Änderung an den von Ihnen verwendeten Kanälen diesen Fehler verursacht hat, oder wenn Sie Ihren Systemzustand zum Zweck der Reproduzierbarkeit festhalten möchten. Der Befehl `guix describe` gibt Ihnen Antwort auf diese Fragen.

Wenn Sie ihn aus einem mit `guix pull` bezogenen `guix` heraus ausführen, zeigt Ihnen `guix describe` die Kanäle an, aus denen es erstellt wurde, jeweils mitsamt ihrer Repository-URL und Commit-ID (siehe Kapitel 7 [Kanäle], Seite 77):

```
$ guix describe
Generation 10 03. September 2018 17:32:44 (aktuell)
  guix e0fa68c
    Repository-URL: https://git.savannah.gnu.org/git/guix.git
    Branch: master
    Commit: e0fa68c7718fffd33d81af415279d6ddb518f727
```

Wenn Sie mit dem Versionskontrollsystem Git vertraut sind, erkennen Sie vielleicht die Ähnlichkeit zu `git describe`; die Ausgabe ähnelt auch der von `guix pull --list-generations` eingeschränkt auf die aktuelle Generation (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65). Weil die oben gezeigte Git-Commit-ID eindeutig eine bestimmte Version von Guix bezeichnet, genügt diese Information, um die von Ihnen benutzte Version von Guix zu beschreiben, und auch, um sie nachzubilden.

Damit es leichter ist, Guix nachzubilden, kann Ihnen `guix describe` auch eine Liste der Kanäle statt einer menschenlesbaren Beschreibung wie oben liefern:

```
$ 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")))))
```

Sie können die Ausgabe in einer Datei speichern, die Sie an `guix pull -C` auf einer anderen Maschine oder zu einem späteren Zeitpunkt übergeben, wodurch dann eine Instanz *von genau derselben Guix-Version* installiert wird (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65). Daraufhin können Sie, weil Sie jederzeit dieselbe Version von Guix installieren können, auch gleich *eine vollständige Softwareumgebung genau nachbilden.* Wir halten das trotz aller Bescheidenheit für *klasse* und hoffen, dass Ihnen das auch gefällt!

Die genauen Befehlszeilenoptionen, die `guix describe` unterstützt, lauten wie folgt:

`--format=`*Format*
`-f` *Format* Die Ausgabe im angegebenen *Format* generieren, was eines der Folgenden sein muss:

- `human` für menschenlesbare Ausgabe,
- `channels` eine Liste von Kanalspezifikationen erzeugen, die an `guix pull -C` übergeben werden oder als `~/.config/guix/channels.scm` eingesetzt werden können (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65),
- `channels-sans-intro` verhält sich wie `channels`, aber ohne das `introduction`-Feld; benutzen Sie es, um eine Kanalspezifikation zu erzeugen, die für die Guix-Version 1.1.0 oder früher geeignet ist – das `introduction`-Feld ist für Kanalauthentifizierung gedacht (siehe Kapitel 7 [Kanäle], Seite 77), die von diesen älteren Versionen nicht unterstützt wird,
- `json` generiert eine Liste von Kanalspezifikationen im JSON-Format,
- `recutils` generiert eine Liste von Kanalspezifikationen im Recutils-Format.

`--list-formats`
Verfügbare Formate für die Befehlszeilenoption `--format` anzeigen.

`--profile=`*Profil*
`-p` *Profil* Informationen über das *Profil* anzeigen.

6.10 `guix archive` aufrufen

Der Befehl `guix archive` ermöglicht es Nutzern, Dateien im Store in eine einzelne Archivdatei zu *exportieren* und diese später auf einer Maschine, auf der Guix läuft, zu *importieren*. Insbesondere können so Store-Objekte von einer Maschine in den Store einer anderen Maschine übertragen werden.

> **Anmerkung:** Wenn Sie nach einer Möglichkeit suchen, Archivdateien für andere Werkzeuge als Guix zu erstellen, finden Sie Informationen dazu im Abschnitt Abschnitt 8.3 [Aufruf von guix pack], Seite 99.

Führen Sie Folgendes aus, um Store-Dateien als ein Archiv auf die Standardausgabe zu exportieren:

```
guix archive --export Optionen Spezifikationen...
```

Spezifikationen sind dabei entweder die Namen von Store-Dateien oder Paketspezifikationen wie bei `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Zum Beispiel erzeugt der folgende Befehl ein Archiv der `gui`-Ausgabe des Pakets `git` sowie die Hauptausgabe von `emacs`:

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

Wenn die angegebenen Pakete noch nicht erstellt worden sind, werden sie durch `guix archive` automatisch erstellt. Der Erstellungsprozess kann durch die gemeinsamen Erstellungsoptionen gesteuert werden (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

Um das `emacs`-Paket auf eine über SSH verbundene Maschine zu übertragen, würde man dies ausführen:

```
guix archive --export -r emacs | ssh die-maschine guix archive --import
```

Auf gleiche Art kann auch ein vollständiges Benutzerprofil von einer Maschine auf eine andere übertragen werden:

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

Jedoch sollten Sie in beiden Beispielen beachten, dass alles, was zu `emacs`, dem Profil oder deren Abhängigkeiten (wegen `-r`) gehört, übertragen wird, egal ob es schon im Store der Zielmaschine vorhanden ist oder nicht. Mit der Befehlszeilenoption `--missing` lässt sich herausfinden, welche Objekte im Ziel-Store noch fehlen. Der Befehl `guix copy` vereinfacht und optimiert diesen gesamten Prozess, ist also, was Sie in diesem Fall wahrscheinlich eher benutzen wollten (siehe Abschnitt 10.13 [Aufruf von guix copy], Seite 241).

Dabei wird jedes Store-Objekt als „normalisiertes Archiv“, kurz „Nar“, formatiert (was im Folgenden beschrieben wird) und die Ausgabe von `guix archive --export` (bzw. die Eingabe von `guix archive --import`) ist ein *Nar-Bündel*.

Das Nar-Format folgt einem ähnlichen Gedanken wie beim „tar“-Format, unterscheidet sich aber auf eine für unsere Zwecke geeignetere Weise. Erstens werden im Nar-Format nicht sämtliche Unix-Metadaten aller Dateien aufgenommen, sondern nur der Dateityp (ob es sich um eine reguläre Datei, ein Verzeichnis oder eine symbolische Verknüpfung handelt). Unix-Dateiberechtigungen sowie Besitzer und Gruppe werden nicht gespeichert. Zweitens entspricht die Reihenfolge, in der der Inhalt von Verzeichnissen abgelegt wird, immer der Reihenfolge, in der die Dateinamen gemäß der C-Locale sortiert würden. Dadurch wird die Erstellung von Archivdateien völlig deterministisch.

Das Nar-Bündel-Format setzt sich im Prinzip aus null oder mehr aneinandergehängten Nars zusammen mit Metadaten für jedes enthaltene Store-Objekt, nämlich dessen Dateinamen, Referenzen, der zugehörigen Ableitung sowie einer digitalen Signatur.

Beim Exportieren versieht der Daemon den Inhalt des Archivs mit einer digitalen Signatur, auch Beglaubigung genannt. Diese digitale Signatur wird an das Archiv angehängt. Beim Importieren verifiziert der Daemon die Signatur und lehnt den Import ab, falls die Signatur ungültig oder der Signierschlüssel nicht autorisiert ist.

Die wichtigsten Befehlszeilenoptionen sind:

`--export` Exportiert die angegebenen Store-Dateien oder Pakete (siehe unten) und schreibt das resultierende Archiv auf die Standardausgabe.

Abhängigkeiten *fehlen* in der Ausgabe, außer wenn `--recursive` angegeben wurde.

`-r`
`--recursive`
Zusammen mit `--export` wird `guix archive` hiermit angewiesen, Abhängigkeiten der angegebenen Objekte auch ins Archiv aufzunehmen. Das resultierende Archiv ist somit eigenständig; es enthält den Abschluss der exportierten Store-Objekte.

`--import` Ein Archiv von der Standardeingabe lesen und darin enthaltende Dateien in den Store importieren. Der Import bricht ab, wenn das Archiv keine gültige digitale Signatur hat oder wenn es von einem öffentlichen Schlüssel signiert wurde, der keiner der autorisierten Schlüssel ist (siehe `--authorize` weiter unten).

`--missing`
Eine Liste der Store-Dateinamen von der Standardeingabe lesen, je ein Name pro Zeile, und auf die Standardausgabe die Teilmenge dieser Dateien schreiben, die noch nicht im Store vorliegt.

`--generate-key[=`*Parameter*`]`
Ein neues Schlüsselpaar für den Daemon erzeugen. Dies ist erforderlich, damit Archive mit `--export` exportiert werden können. Normalerweise wird diese Option sofort umgesetzt, jedoch kann sie, wenn erst der Entropie-Pool neu gefüllt werden muss, einige Zeit in Anspruch nehmen. Auf Guix System kümmert sich der `guix-service-type` darum, dass beim ersten Systemstart das Schlüsselpaar erzeugt wird.

Das erzeugte Schlüsselpaar wird typischerweise unter `/etc/guix` gespeichert, in den Dateien `signing-key.pub` (für den öffentlichen Schlüssel) und `signing-key.sec` (für den privaten Schlüssel, der geheim gehalten werden muss). Wurden keine *Parameters* angegeben, wird ein ECDSA-Schlüssel unter Verwendung der Kurve Ed25519 erzeugt, oder, falls die Libgcrypt-Version älter als 1.6.0 ist, ein 4096-Bit-RSA-Schlüssel. Sonst geben die *Parameter* für Libgcrypt geeignete Parameter für `genkey` an (siehe Abschnitt “General public-key related Functions” in *Referenzhandbuch von Libgcrypt*).

`--authorize`
Mit dem auf der Standardeingabe übergebenen öffentlichen Schlüssel signierte Importe autorisieren. Der öffentliche Schlüssel muss als „advanced“-formatierter

S-Ausdruck gespeichert sein, d.h. im selben Format wie die Datei `signing-key.pub`.

Die Liste autorisierter Schlüssel wird in der Datei `/etc/guix/acl` gespeichert, die auch von Hand bearbeitet werden kann. Die Datei enthält „advanced“-formatierte S-Ausdrücke (`https://people.csail.mit.edu/rivest/Sexp.txt`) und ist als eine Access Control List für die Simple Public-Key Infrastructure (SPKI) (`https://theworld.com/~cme/spki.txt`) aufgebaut.

`--extract=`*Verzeichnis*
`-x` *Verzeichnis*

Ein Archiv mit einem einzelnen Objekt lesen, wie es von Substitutservern geliefert wird (siehe Abschnitt 6.3 [Substitute], Seite 56), und ins *Verzeichnis* entpacken. Dies ist eine systemnahe Operation, die man nur selten direkt benutzt; siehe unten.

Zum Beispiel entpackt folgender Befehl das Substitut für Emacs, wie es von `ci.guix.gnu.org` geliefert wird, nach `/tmp/emacs`:

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

Archive mit nur einem einzelnen Objekt unterscheiden sich von Archiven für mehrere Dateien, wie sie `guix archive --export` erzeugt; sie enthalten nur ein einzelnes Store-Objekt und *keine* eingebettete Signatur. Beim Entpacken findet also *keine* Signaturprüfung statt und ihrer Ausgabe sollte so erst einmal nicht vertraut werden.

Der eigentliche Zweck dieser Operation ist, die Inspektion von Archivinhalten von Substitutservern möglich zu machen, auch wenn diesen unter Umständen nicht vertraut wird (siehe Abschnitt 10.12 [Aufruf von guix challenge], Seite 238).

`--list`
`-t`

Ein Archiv mit einem einzelnen Objekt lesen, wie es von Substitutservern geliefert wird (siehe Abschnitt 6.3 [Substitute], Seite 56), und die Dateien darin ausgeben, wie in diesem Beispiel:

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

7 Kanäle

Guix und die Sammlung darin verfügbarer Pakete können Sie durch Ausführen von `guix pull` aktualisieren (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65). Standardmäßig lädt `guix pull` Guix selbst vom offiziellen Repository von GNU Guix herunter und installiert es. Diesen Vorgang können Sie anpassen, indem Sie *Kanäle* in der Datei `~/.config/guix/channels.scm` angeben. Ein Kanal enthält eine Angabe einer URL und eines Branches eines zu installierenden Git-Repositorys. Sie können `guix pull` veranlassen, die Aktualisierungen von einem oder mehreren Kanälen zu beziehen. Mit anderen Worten können Kanäle benutzt werden, um Guix *anzupassen* und zu *erweitern*, wie wir im Folgenden sehen werden. Guix ist in der Lage, dabei Sicherheitsbedenken zu berücksichtigen und Aktualisierungen zu authentifizieren.

7.1 Weitere Kanäle angeben

Sie können auch *weitere Kanäle* als Bezugsquelle angeben. Um einen Kanal zu benutzen, tragen Sie ihn in `~/.config/guix/channels.scm` ein, damit `guix pull` diesen Kanal *zusätzlich* zu den standardmäßigen Guix-Kanälen als Paketquelle verwendet:

```
;; Paketvarianten zu denen von Guix dazunehmen.
(cons (channel
        (name 'paketvarianten)
        (url "https://example.org/variant-packages.git"))
      %default-channels)
```

Beachten Sie, dass der obige Schnipsel (wie immer!) Scheme-Code ist; mit `cons` fügen wir einen Kanal zur Liste der Kanäle hinzu, an die die Variable `%default-channels` gebunden ist (siehe Abschnitt "Pairs" in *Referenzhandbuch zu GNU Guile*). Mit diesem Dateiinhalt wird `guix pull` nun nicht mehr nur Guix, sondern auch die Paketmodule aus Ihrem Repository erstellen. Das Ergebnis in `~/.config/guix/current` ist so die Vereinigung von Guix und Ihren eigenen Paketmodulen.

```
$ guix describe
Generation 19 27. August 2018 16:20:48
  guix d894ab8
    Repository-URL: https://git.savannah.gnu.org/git/guix.git
    Branch: master
    Commit: d894ab8e9bfabcefa6c49d9ba2e834dd5a73a300
  paketvarianten dd3df5e
    Repository-URL: https://example.org/variant-packages.git
    Branch: master
    Commit: dd3df5e2c8818760a8fc0bd699e55d3b69fef2bb
```

Obige Ausgabe von `guix describe` zeigt an, dass jetzt Generation 19 läuft und diese sowohl Guix als auch Pakete aus dem Kanal `paketvarianten` enthält (siehe Abschnitt 6.9 [Aufruf von guix describe], Seite 72).

7.2 Eigenen Guix-Kanal benutzen

Der Kanal namens `guix` gibt an, wovon Guix selbst – seine Befehlszeilenwerkzeuge und seine Paketsammlung – heruntergeladen werden sollen. Wenn Sie zum Beispiel mit einer anderen

Kopie des Guix-Repositorys arbeiten möchten und diese auf `example.org` zu finden ist, und zwar im Branch namens `super-hacks`, dann schreiben Sie folgende Spezifikation in `~/.config/guix/channels.scm`:

```
;; 'guix pull' ein anderes Repository benutzen lassen.
(list (channel
        (name 'guix)
        (url "https://example.org/anderes-guix.git")
        (branch "super-hacks")))
```

Ab dann wird `guix pull` seinen Code vom Branch `super-hacks` des Repositorys auf `example.org` beziehen. Wie man dessen Autorisierung bewerkstelligt, können Sie im Folgenden lesen (siehe Abschnitt 7.4 [Kanalauthentifizierung], Seite 79).

7.3 Guix nachbilden

Der Befehl `guix describe` zeigt genau, aus welchen Commits die Guix-Instanz erstellt wurde, die Sie benutzen (siehe Abschnitt 6.9 [Aufruf von guix describe], Seite 72). Sie können diese Instanz auf einer anderen Maschine oder zu einem späteren Zeitpunkt nachbilden, indem Sie eine Kanalspezifikation angeben, die auf diese Commits „festgesetzt“ ist.

```
;; Ganz bestimmte Commits der relevanten Kanäle installieren.
(list (channel
       (name 'guix)
       (url "https://git.savannah.gnu.org/git/guix.git")
       (commit "6298c3ffd9654d3231a6f25390b056483e8f407c"))
      (channel
       (name 'paketvarianten)
       (url "https://example.org/variant-packages.git")
       (commit "dd3df5e2c8818760a8fc0bd699e55d3b69fef2bb")))
```

Um so eine festgesetzte Kanalspezifikation zu bekommen, ist es am einfachsten, `guix describe` auszuführen und seine Ausgabe in `channels`-Formatierung in einer Datei zu speichern. Das geht so:

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

Die erzeugte Datei `channels.scm` kann mit der Befehlszeilenoption `-C` von `guix pull` (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65) oder von `guix time-machine` (siehe Abschnitt 6.7 [Aufruf von time-machine], Seite 69) übergeben werden. Ein Beispiel:

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

Anhand der Datei `channels.scm` ist festgelegt, dass der obige Befehl immer *genau dieselbe Guix-Instanz* lädt und dann mit dieser Instanz genau dasselbe Python startet (siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86). So werden auf jeder beliebigen Maschine zu jedem beliebigen Zeitpunkt genau dieselben Binärdateien ausgeführt, Bit für Bit.

Das Problem, das solche festgesetzten Kanäle lösen, ist ähnlich zu dem, wofür andere Werkzeuge zur Softwareauslieferung „Lockfiles“ einsetzen – ein Satz von Paketen wird reproduzierbar auf eine bestimmte Version festgesetzt. Im Fall von Guix werden allerdings sämtliche mit Guix ausgelieferte Pakete auf die angegebenen Commits fixiert; tatsächlich sogar der ganze Rest von Guix mitsamt seinen Kernmodulen und Programmen für die Befehlszeile. Dazu wird sicher garantiert, dass Sie wirklich genau dieselbe Software vorfinden werden.

Das verleiht Ihnen Superkräfte, mit denen Sie die Provenienz binärer Artefakte sehr feinkörnig nachverfolgen können und Software-Umgebungen nach Belieben nachbilden können. Sie können es als eine Art Fähigkeit zur „Meta-Reproduzierbarkeit“ auffassen, wenn Sie möchten. Der Abschnitt Abschnitt 6.8 [Untergeordnete], Seite 70, beschreibt eine weitere Möglichkeit, diese Superkräfte zu nutzen.

7.4 Kanalauthentifizierung

Die Befehle `guix pull` und `guix time-machine` *authentifizieren* den von Kanälen bezogenen Code. Es wird geprüft, dass jeder geladene Commit von einem autorisierten Entwickler signiert wurde. Das Ziel ist, Sie vor unautorisierten Änderungen am Kanal, die Nutzer bösartigen Code ausführen lassen, zu schützen.

Als Nutzer müssen Sie eine *Kanaleinführung* („Channel Introduction“) in Ihrer Kanaldatei angeben, damit Guix weiß, wie der erste Commit authentifiziert werden kann. Eine Kanalspezifikation, zusammen mit seiner Einführung, sieht in etwa so aus:

```
(channel
  (name 'ein-kanal)
  (url "https://example.org/ein-kanal.git")
  (introduction
   (make-channel-introduction
    "6f0d8cc0d88abb59c324b2990bfee2876016bb86"
    (openpgp-fingerprint
     "CABB A931 C0FF EEC6 900D  0CFB 090B 1199 3D9A EBB5"))))
```

Obige Spezifikation zeigt den Namen und die URL des Kanals. Der Aufruf von `make-channel-introduction`, den Sie oben sehen, gibt an, dass die Authentifizierung dieses Kanals bei Commit `6f0d8cc...` beginnt, welcher mit dem OpenPGP-Schlüssel mit Fingerabdruck `CABB A931...` signiert ist.

Für den Hauptkanal mit Namen `guix` bekommen Sie diese Informationen automatisch mit Ihrer Guix-Installation. Für andere Kanäle tragen Sie die Kanaleinführung, die Ihnen die Kanalautoren mitteilen, in die Datei `channels.scm` ein. Achten Sie dabei darauf, die Kanaleinführung von einer vertrauenswürdigen Quelle zu bekommen, denn sie stellt die Wurzel all Ihren Vertrauens dar.

Wenn Sie neugierig sind, wie die Authentifizierung funktioniert, lesen Sie weiter!

7.5 Kanäle mit Substituten

Wenn Sie `guix pull` ausführen, wird Guix als Erstes die Definition jedes verfügbaren Pakets kompilieren. Das ist eine teure Operation, für die es aber Substitute geben könnte (siehe Abschnitt 6.3 [Substitute], Seite 56). Mit dem folgenden Schnipsel in `channels.scm` wird sichergestellt, dass `guix pull` den neuesten Commit benutzt, für den bereits Substitute für die Paketdefinitionen vorliegen. Dazu wird der Server für Kontinuierliche Integration auf `https://ci.guix.gnu.org` angefragt.

```
(use-modules (guix ci))

(list (channel-with-substitutes-available
       %default-guix-channel
```

```
"https://ci.guix.gnu.org"))
```

Beachten Sie: Das heißt *nicht*, dass für alle Pakete, die Sie installieren werden, nachdem Sie `guix pull` durchgeführt haben, bereits Substitute vorliegen. Es wird nur sichergestellt, dass `guix pull` keine Paketdefinitionen zu kompilieren versucht. Das hilft besonders auf Maschinen mit eingeschränkten Rechenressourcen.

7.6 Einen Kanal erstellen

Sagen wir, Sie haben ein paar eigene Paketvarianten oder persönliche Pakete, von denen Sie meinen, dass sie *nicht* geeignet sind, ins Guix-Projekt selbst aufgenommen zu werden, die Ihnen aber dennoch wie andere Pakete auf der Befehlszeile zur Verfügung stehen sollen. Dann würden Sie zunächst Module mit diesen Paketdefinitionen schreiben (siehe Abschnitt 9.1 [Paketmodule], Seite 108) und diese dann in einem Git-Repository verwalten, welches Sie selbst oder jeder andere dann als zusätzlichen Kanal eintragen können, von dem Pakete geladen werden. Klingt gut, oder?

> **Warnung:** Bevor Sie, verehrter Nutzer, ausrufen: „Wow, das ist *soooo cooool*!“, und Ihren eigenen Kanal der Welt zur Verfügung stellen, möchten wir Ihnen auch ein paar Worte der Warnung mit auf den Weg geben:
>
> - Bevor Sie einen Kanal veröffentlichen, überlegen Sie sich bitte erst, ob Sie die Pakete nicht besser zum eigentlichen Guix-Projekt beisteuern (siehe Kapitel 22 [Mitwirken], Seite 708). Das Guix-Projekt ist gegenüber allen Arten freier Software offen und zum eigentlichen Guix gehörende Pakete stehen allen Guix-Nutzern zur Verfügung, außerdem profitieren sie von Guix' Qualitätssicherungsprozess.
> - Wenn Sie Paketdefinitionen außerhalb von Guix betreuen, sehen wir Guix-Entwickler es als *Ihre Aufgabe an, deren Kompatibilität sicherzstellen.* Bedenken Sie, dass Paketmodule und Paketdefinitionen nur Scheme-Code sind, der verschiedene Programmierschnittstellen (APIs) benutzt. Wir nehmen uns das Recht heraus, diese APIs jederzeit zu ändern, damit wir Guix besser machen können, womöglich auf eine Art, wodurch Ihr Kanal nicht mehr funktioniert. Wir ändern APIs nie einfach so, werden aber auch *nicht* versprechen, APIs nicht zu verändern.
> - Das bedeutet auch, dass Sie, wenn Sie einen externen Kanal verwenden und dieser kaputt geht, Sie dies bitte *den Autoren des Kanals* und nicht dem Guix-Projekt melden.
>
> Wir haben Sie gewarnt! Allerdings denken wir auch, dass externe Kanäle eine praktische Möglichkeit sind, die Paketsammlung von Guix zu ergänzen und Ihre Verbesserungen mit anderen zu teilen, wie es dem Grundgedanken freier Software (`https://www.gnu.org/philosophy/free-sw.html`) entspricht. Bitte schicken Sie eine E-Mail an `guix-devel@gnu.org`, wenn Sie dies diskutieren möchten.

Um einen Kanal zu erzeugen, müssen Sie ein Git-Repository mit Ihren eigenen Paketmodulen erzeugen und den Zugriff darauf ermöglichen. Das Repository kann beliebigen Inhalt haben, aber wenn es ein nützlicher Kanal sein soll, muss es Guile-Module enthalten, die Pakete exportieren. Sobald Sie anfangen, einen Kanal zu benutzen, verhält sich Guix, als

wäre das Wurzelverzeichnis des Git-Repositorys des Kanals in Guiles Ladepfad enthalten (siehe Abschnitt “Load Paths” in *Referenzhandbuch zu GNU Guile*). Wenn Ihr Kanal also zum Beispiel eine Datei als `my-packages/my-tools.scm` enthält, die ein Guile-Modul definiert, dann wird das Modul unter dem Namen `(my-packages my-tools)` verfügbar sein und Sie werden es wie jedes andere Modul benutzen können (siehe Abschnitt “Modules” in *Referenzhandbuch zu GNU Guile*).

Als Kanalautor möchten Sie vielleicht Materialien mitliefern, damit dessen Nutzer ihn authentifizieren können. Siehe Abschnitt 7.4 [Kanalauthentifizierung], Seite 79, und Abschnitt 7.9 [Weitere Kanalautorisierungen angeben], Seite 82, für Informationen, wie das geht.

7.7 Paketmodule in einem Unterverzeichnis

Als Kanalautor möchten Sie vielleicht Ihre Kanalmodule in einem Unterverzeichnis anlegen. Wenn sich Ihre Module im Unterverzeichnis `guix` befinden, müssen Sie eine Datei `.guix-channel` mit Metadaten einfügen:

```
(channel
  (version 0)
  (directory "guix"))
```

7.8 Kanalabhängigkeiten deklarieren

Kanalautoren können auch beschließen, die Paketsammlung von anderen Kanälen zu erweitern. Dazu können sie in einer Metadatendatei `.guix-channel` deklarieren, dass ihr Kanal von anderen Kanälen abhängt. Diese Datei muss im Wurzelverzeichnis des Kanal-Repositorys platziert werden.

Die Metadatendatei sollte einen einfachen S-Ausdruck wie diesen enthalten:

```
(channel
 (version 0)
 (dependencies
  (channel
   (name irgendeine-sammlung)
   (url "https://example.org/erste-sammlung.git")

   ;; Der Teil mit der 'introduction' hier ist optional. Sie geben hier
   ;; die Kanaleinführung an, wenn diese Abhängigkeiten authentifiziert
   ;; werden können.
   (introduction
    (channel-introduction
      (version 0)
      (commit "a8883b58dc82e167c96506cf05095f37c2c2c6cd")
      (signer "CABB A931 C0FF EEC6 900D  0CFB 090B 1199 3D9A EBB5"))))
  (channel
   (name eine-andere-sammlung)
   (url "https://example.org/zweite-sammlung.git")
   (branch "testing"))))
```

Im Beispiel oben wird deklariert, dass dieser Kanal von zwei anderen Kanälen abhängt, die beide automatisch geladen werden. Die vom Kanal angebotenen Module werden in einer Umgebung kompiliert, in der die Module all dieser deklarierten Kanäle verfügbar sind.

Um Verlässlichkeit und Wartbarkeit zu gewährleisten, sollten Sie darauf verzichten, eine Abhängigkeit von Kanälen herzustellen, die Sie nicht kontrollieren, außerdem sollten Sie sich auf eine möglichst kleine Anzahl von Abhängigkeiten beschränken.

7.9 Weitere Kanalautorisierungen angeben

Wie wir oben gesehen haben, wird durch Guix sichergestellt, dass von Kanälen geladener Code von autorisierten Entwicklern stammt. Als Kanalautor müssen Sie die Liste der autorisierten Entwickler in der Datei `.guix-authorizations` im Git-Repository des Kanals festlegen. Die Regeln für die Authentifizierung sind einfach: Jeder Commit muss mit einem Schlüssel signiert sein, der in der Datei `.guix-authorizations` seines bzw. seiner Elterncommits steht[1] Die Datei `.guix-authorizations` sieht so aus:

```
;; Beispiel für eine '.guix-authorizations'-Datei.

(authorizations
 (version 0)               ;aktuelle Version des Dateiformats

 (("AD17 A21E F8AE D8F1 CC02  DBD9 F8AE D8F1 765C 61E3"
   (name "alice"))
  ("2A39 3FFF 68F4 EF7A 3D29  12AF 68F4 EF7A 22FB B2D5"
   (name "bob"))
  ("CABB A931 C0FF EEC6 900D  0CFB 090B 1199 3D9A EBB5"
   (name "charlie"))))
```

Auf jeden Fingerabdruck folgen optional Schlüssel-/Wert-Paare wie im obigen Beispiel. Derzeit werden diese Schlüssel-/Wert-Paare ignoriert.

Diese Authentifizierungsregel hat ein Henne-Ei-Problem zur Folge: Wie authentifizieren wir den ersten Commit? Und damit zusammenhängend: Was tun wir, wenn die Historie eines Kanal-Repositorys *nicht* signierte Commits enthält und eine `.guix-authorizations`-Datei fehlt? Und wie legen wir eine Abspaltung („Fork“) existierender Kanäle an?

Kanaleinführungen beantworten diese Fragen, indem sie den ersten Commit eines Kanals angeben, der authentifiziert werden soll. Das erste Mal, wenn ein Kanal durch `guix pull` oder `guix time-machine` geladen wird, wird nachgeschaut, was der einführende Commit ist und ob er mit dem angegebenen OpenPGP-Schlüssel signiert wurde. Von da an werden Commits gemäß obiger Regel authentifiziert. Damit die Authentifizierung erfolgreich ist, muss der Ziel-Commit entweder Nachkomme oder Vorgänger des einführenden Commits sein.

Außerdem muss Ihr Kanal alle OpenPGP-Schlüssel zur Verfügung stellen, die jemals in `.guix-authorizations` erwähnt wurden, gespeichert in Form von `.key`-Dateien, entweder als Binärdateien oder mit „ASCII-Hülle“. Nach Vorgabe wird nach diesen `.key`-Dateien im

[1] Git-Commits bilden einen *gerichteten azyklischen Graphen* (englisch „directed acyclic graph“, kurz DAG). Jeder Commit kann null oder mehr Eltern haben; in der Regel haben Commits einen Elterncommit, Mergecommits haben zwei. Lesen Sie *Git for Computer Scientists* (`https://eagain.net/articles/git-for-computer-scientists/`) für eine gelungene Übersicht.

Branch namens `keyring` gesucht, aber Sie können wie hier einen anderen Branchnamen in `.guix-channel` angeben:

```
(channel
  (version 0)
  (keyring-reference "my-keyring-branch"))
```

Zusammenfassen haben Kanalautoren drei Dinge zu tun, damit Nutzer deren Code authentifizieren können:

1. Exportieren Sie die OpenPGP-Schlüssel früherer und aktueller Committer mittels `gpg --export` und speichern Sie sie in `.key`-Dateien, nach Vorgabe gespeichert in einem Branch namens `keyring` (wir empfehlen, dass Sie diesen als einen *verwaisten Branch*, d.h. einen „Orphan Branch“, anlegen).
2. Sie müssen eine anfängliche `.guix-authorizations`-Datei im Kanalrepository platzieren. Der Commit dazu muss signiert sein (siehe Abschnitt 22.8 [Commit-Zugriff], Seite 733, für Informationen, wie Sie Git-Commits signieren).
3. Sie müssen veröffentlichen, wie die Kanaleinführung aussieht, zum Beispiel auf der Webseite des Kanals. Die Kanaleinführung besteht, wie wir oben gesehen haben, aus einem Paar aus Commit und Schlüssel – für denjenigen Commit, der `.guix-authorizations` hinzugefügt hat, mit dem Fingerabdruck des OpenPGP-Schlüssels, mit dem er signiert wurde.

Bevor Sie auf Ihr öffentliches Git-Repository pushen, können Sie `guix git-authenticate` ausführen, um zu überprüfen, dass Sie alle Commits, die Sie pushen würden, mit einem autorisierten Schlüssel signiert haben:

```
guix git authenticate Commit Unterzeichner
```

Dabei sind *Commit* und *Unterzeichner* Ihre Kanaleinführung. Siehe Abschnitt 8.5 [Aufruf von guix git authenticate], Seite 106, für die Details.

Um einen signierten Kanal anzubieten, ist Disziplin vonnöten: Jeder Fehler, wie z.B. ein unsignierter Commit oder ein mit einem unautorisierten Schlüssel signierter Commit, verhindert, das Nutzer den Kanal benutzen können – naja, das ist ja auch der Zweck der Authentifizierung! Passen Sie besonders bei Merges auf: Merge-Commits werden dann und nur dann als authentisch angesehen, wenn sie mit einem Schlüssel aus der `.guix-authorizations`-Datei *beider* Branches signiert wurden.

7.10 Primäre URL

Kanalautoren können die primäre URL des Git-Repositorys ihres Kanals in der Datei `.guix-channel` hinterlegen, etwa so:

```
(channel
  (version 0)
  (url "https://example.org/guix.git"))
```

Dadurch kann der Befehl `guix pull` feststellen, ob er Code von einem Spiegelserver des Kanals lädt. In diesem Fall wird der Benutzer davor gewarnt, dass Spiegelserver veraltete Versionen anbieten könnten, und die eigentliche, primäre URL anzeigen. Auf diese Weise wird verhindert, dass Nutzer manipuliert werden, Code von einem veralteten Spiegelserver zu benutzen, der keine Sicherheitsaktualisierungen bekommt.

Diese Funktionalität ergibt nur bei authentifizierbaren Repositorys Sinn, wie zum Beispiel dem offiziellen `guix`-Kanal, für den `guix pull` sicherstellt, dass geladener Code authentisch ist.

7.11 Kanalneuigkeiten verfassen

Kanalautoren möchten ihren Nutzern vielleicht manchmal Informationen über wichtige Änderungen im Kanal zukommen lassen. Man könnte allen eine E-Mail schicken, aber das wäre unbequem.

Stattdessen können Kanäle eine Datei mit Neuigkeiten („News File“) anbieten: Wenn die Kanalnutzer `guix pull` ausführen, wird diese Datei automatisch ausgelesen. Mit `guix pull --news` kann man sich die Ankündigungen anzeigen lassen, die den neu gepullten Commits entsprechen, falls es welche gibt.

Dazu müssen Kanalautoren zunächst den Namen der Datei mit Neuigkeiten in der Datei `.guix-channel` nennen:

```
(channel
  (version 0)
  (news-file "etc/news.txt"))
```

Die Datei mit Neuigkeiten, `etc/news.txt` in diesem Beispiel, muss selbst etwa so aussehen:

```
(channel-news
  (version 0)
  (entry (tag "the-bug-fix")
         (title (en "Fixed terrible bug")
                (fr "Oh la la"))
         (body (en "@emph{Good news}!  It's fixed!")
               (eo "Certe ĝi pli bone funkcias nun!")))
  (entry (commit "bdcabe815cd28144a2d2b4bc3c5057b051fa9906")
         (title (en "Added a great package")
                (ca "Què vol dir guix?"))
         (body (en "Don't miss the @code{hello} package!"))))
```

Obwohl die Datei für Neuigkeiten Scheme-Syntax verwendet, sollten Sie ihr keinen Dateinamen mit der Endung `.scm` geben, sonst wird sie bei der Erstellung des Kanals miteinbezogen und dann zu einem Fehler führen, weil es sich bei ihr um kein gültiges Modul handelt. Alternativ können Sie auch das Kanalmodul in einem Unterverzeichnis platzieren und die Datei für Neuigkeiten in einem Verzeichnis außerhalb platzieren.

Die Datei setzt sich aus einer Liste von Einträgen mit Neuigkeiten („News Entries“) zusammen. Jeder Eintrag ist mit einem Commit oder einem Git-Tag assoziiert und beschreibt die Änderungen, die in diesem oder auch vorangehenden Commits gemacht wurden. Benutzer sehen die Einträge nur beim erstmaligen Übernehmen des Commits, auf den sich der jeweilige Eintrag bezieht.

Das `title`-Feld sollte eine einzeilige Zusammenfassung sein, während `body` beliebig lang sein kann. Beide können Texinfo-Auszeichnungen enthalten (siehe Abschnitt “Overview” in *GNU Texinfo*). Sowohl `title` als auch `body` sind dabei eine Liste aus Tupeln mit jeweils Sprachcode und Mitteilung, wodurch `guix pull` Neuigkeiten in derjenigen Sprache anzeigen kann, die der vom Nutzer eingestellten Locale entspricht.

Wenn Sie Neuigkeiten mit einem gettext-basierten Arbeitsablauf übersetzen möchten, können Sie übersetzbare Zeichenketten mit `xgettext` extrahieren (siehe Abschnitt “xgettext Invocation” in *GNU Gettext Utilities*). Unter der Annahme, dass Sie Einträge zu Neuigkeiten zunächst auf Englisch verfassen, können Sie mit diesem Befehl eine PO-Datei erzeugen, die die zu übersetzenden Zeichenketten enthält:

```
xgettext -o news.po -l scheme -ken etc/news.txt
```

Kurz gesagt, ja, Sie können Ihren Kanal sogar als Blog missbrauchen. Aber das ist *nicht ganz*, was Ihre Nutzer erwarten dürften.

8 Entwicklung

Wenn Sie ein Software-Entwickler sind, gibt Ihnen Guix Werkzeuge an die Hand, die Sie für hilfreich erachten dürften – ganz unabhängig davon, in welcher Sprache Sie entwickeln. Darum soll es in diesem Kapitel gehen.

Der Befehl `guix shell` stellt eine bequeme Möglichkeit dar, Umgebungen für einmalige Nutzungen zu erschaffen, etwa zur Entwicklung an einem Programm oder um einen Befehl auszuführen, den Sie nicht in Ihr Profil installieren möchten. Der Befehl `guix pack` macht es Ihnen möglich, *Anwendungsbündel* zu erstellen, die leicht an Nutzer verteilt werden können, die kein Guix benutzen.

8.1 `guix shell` aufrufen

Der Zweck von `guix shell` ist, dass Sie Software-Umgebungen für außergewöhnliche Fälle einfach aufsetzen können, ohne dass Sie Ihr Profil ändern müssen. Normalerweise braucht man so etwas für Entwicklungsumgebungen, aber auch wenn Sie Anwendungen ausführen wollen, ohne Ihr Profil mit ihnen zu verunreinigen.

> **Anmerkung:** Der Befehl `guix shell` wurde erst kürzlich eingeführt als Neuauflage von `guix environment` (siehe Abschnitt 8.2 [Aufruf von guix environment], Seite 93). Wenn Sie mit `guix environment` vertraut sind, werden Sie die Ähnlichkeit bemerken, aber wir hoffen, der neue Befehl erweist sich als praktischer.

Die allgemeine Syntax lautet:

```
guix shell [Optionen] [Pakete...]
```

Folgendes Beispiel zeigt, wie Sie eine Umgebung erzeugen lassen, die Python und NumPy enthält, – jedes dazu fehlende Paket wird erstellt oder heruntergeladen –, und in der Umgebung dann `python3` ausführen.

```
guix shell python python-numpy -- python3
```

Entwicklungsumgebungen erzeugen Sie wie im folgenden Beispiel, wo eine interaktive Shell gestartet wird, in der alle Abhängigkeiten und Variable vorliegen, die zur Arbeit an Inkscape nötig sind:

```
guix shell --development inkscape
```

Sobald er die Shell beendet, findet sich der Benutzer in der ursprünglichen Umgebung wieder, in der er sich vor dem Aufruf von `guix shell` befand. Beim nächsten Durchlauf des Müllsammlers (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61) dürfen Pakete von der Platte gelöscht werden, die innerhalb der Umgebung installiert worden sind und außerhalb nicht länger benutzt werden.

Um die Nutzung zu vereinfachen, wird `guix shell`, wenn Sie es interaktiv ohne Befehlszeilenargumente aufrufen, das bewirken, was es dann vermutlich soll. Also:

```
guix shell
```

Wenn im aktuellen Arbeitsverzeichnis oder irgendeinem übergeordneten Verzeichnis eine Datei `manifest.scm` vorkommt, wird sie so behandelt, als hätten Sie sie mit `--manifest` übergeben. Ebenso wird mit einer Datei `guix.scm`, wenn sie in selbigen Verzeichnissen enthalten ist, eine Entwicklungsumgebung hergestellt, als wären `--development` und `--file`

beide angegeben worden. So oder so wird die jeweilige Datei nur dann geladen, wenn das Verzeichnis, in dem sie steckt, in `~/.config/guix/shell-authorized-directories` aufgeführt ist. Damit lassen sich Entwicklungsumgebungen leicht definieren, teilen und betreten.

Vorgegeben ist, dass für die Shell-Sitzung bzw. den Befehl eine *ergänzte* Umgebung aufgebaut wird, in der die neuen Pakete zu Suchpfad-Umgebungsvariablen wie `PATH` hinzugefügt werden. Sie können stattdessen auch beschließen, eine *isolierte* Umgebung anzufordern mit nichts als den Paketen, die Sie angegeben haben. Wenn Sie die Befehlszeilenoption `--pure` angeben, werden alle Definitionen von Umgebungsvariablen aus der Elternumgebung gelöscht[1]. Wenn Sie `--container` angeben, wird die Shell darüber hinaus in einem *Container* vom restlichen System isoliert:

```
guix shell --container emacs gcc-toolchain
```

Dieser Befehl startet eine interaktive Shell in einem Container, der den Zugriff auf alles außer `emacs`, `gcc-toolchain` und deren Abhängigkeiten unterbindet. Im Container haben Sie keinen Netzwerkzugriff und es werden außer dem aktuellen Arbeitsverzeichnis keine Dateien mit der äußeren Umgebung geteilt. Das dient dazu, den Zugriff auf systemweite Ressourcen zu verhindern, wie `/usr/bin` auf Fremddistributionen.

Diese `--container`-Befehlszeilenoption kann sich aber auch als nützlich erweisen, um Anwendungen mit hohem Sicherheitsrisiko wie z.B. Webbrowser in eine isolierte Umgebung eingesperrt auszuführen. Folgender Befehl startet zum Beispiel Ungoogled-Chromium in einer isolierten Umgebung. Dabei wird der Netzwerkzugriff des Wirtssystems geteilt und die Umgebungsvariable `DISPLAY` bleibt erhalten, aber *nicht* einmal das aktuelle Arbeitsverzeichnis wird geteilt zugänglich gemacht.

```
guix shell --container --network --no-cwd ungoogled-chromium \
  --preserve='^DISPLAY$' -- chromium
```

`guix shell` definiert die Variable `GUIX_ENVIRONMENT` in der neu erzeugten Shell. Ihr Wert ist der Dateiname des Profils dieser neuen Umgebung. Das könnten Nutzer verwenden, um zum Beispiel eine besondere Prompt als Eingabeaufforderung für Entwicklungsumgebungen in ihrer `.bashrc` festzulegen (siehe Abschnitt “Bash Startup Files” in *Referenzhandbuch von GNU Bash*):

```
if [ -n "$GUIX_ENVIRONMENT" ]
then
    export PS1="\u@\h \w [dev]\$ "
fi
```

. . . oder um ihr Profil durchzusehen:

```
$ ls "$GUIX_ENVIRONMENT/bin"
```

Im Folgenden werden die verfügbaren Befehlszeilenoptionen zusammengefasst.

`--check` Hiermit wird die Umgebung angelegt und gemeldet, ob die Shell Umgebungsvariable überschreiben würde. Es ist eine gute Idee, diese Befehlszeilenoption anzugeben, wenn Sie zum ersten Mal `guix shell` zum Starten einer interaktiven Sitzung einsetzen, um sicherzugehen, dass Ihre Shell richtig eingestellt ist.

[1] Denken Sie daran, wenn Sie das erste Mal `guix shell` interaktiv benutzen, mit der Befehlszeilenoption `--check` zu prüfen, dass die Shell-Einstellungen die Wirkung von `--pure` nicht wieder zurücknehmen.

Wenn die Shell zum Beispiel den Wert der Umgebungsvariablen `PATH` ändert, meldet dies `--check`, weil Sie dadurch nicht die Umgebung bekämen, die Sie angefordert haben.

In der Regel sind solche Probleme ein Zeichen dafür, dass Dateien, die beim Start der Shell geladen werden, die Umgebungsvariablen unerwartet verändern. Wenn Sie zum Beispiel Bash benutzen, vergewissern Sie sich, dass Umgebungsvariable in `~/.bash_profile` festgelegt oder geändert werden, *nicht* in `~/.bashrc` – erstere Datei wird nur von Login-Shells mit „source“ geladen. Siehe Abschnitt “Bash Startup Files” in *Referenzhandbuch zu GNU Bash* für Details über beim Starten von Bash gelesene Dateien.

`--development`
`-D` Lässt `guix shell` die Abhängigkeiten des danach angegebenen Pakets anstelle des Pakets in die Umgebung aufnehmen. Es kann mit anderen Paketen kombiniert werden. Zum Beispiel wird mit folgendem Befehl eine interaktive Shell gestartet, die die zum Erstellen nötigen Abhängigkeiten von GNU Guile sowie Autoconf, Automake und Libtool enthält:

```
guix shell -D guile autoconf automake libtool
```

`--expression=`*Ausdruck*
`-e` *Ausdruck*
Eine Umgebung für das Paket oder die Liste von Paketen erzeugen, zu der der *Ausdruck* ausgewertet wird.

Zum Beispiel startet dies:

```
guix shell -D -e '(@ (gnu packages maths) petsc-openmpi)'
```

eine Shell mit der Umgebung für eben diese bestimmte Variante des Pakets PETSc.

Wenn man dies ausführt:

```
guix shell -e '(@ (gnu) %base-packages)'
```

bekommt man eine Shell, in der alle Basis-Pakete verfügbar sind.

Die obigen Befehle benutzen nur die Standard-Ausgabe des jeweiligen Pakets. Um andere Ausgaben auszuwählen, können zweielementige Tupel spezifiziert werden:

```
guix shell -e '(list (@ (gnu packages bash) bash) "include")'
```

Siehe [package-development-manifest], Seite 129, für Informationen, wie Sie die Entwicklungsumgebung für ein Paket in einem Manifest wiedergeben.

`--file=`*Datei*
`-f` *Datei* Eine Umgebung erstellen mit dem Paket oder der Liste von Paketen, zu der der Code in der *Datei* ausgewertet wird.

Zum Beispiel könnte die *Datei* eine Definition wie diese enthalten (siehe Abschnitt 9.2 [Pakete definieren], Seite 109):

```
(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-2.64 automake texinfo))))
```

Mit so einer Datei können Sie eine Entwicklungsumgebung für GDB betreten, indem Sie dies ausführen:

```
guix shell -D -f gdb-devel.scm
```

`--manifest=`*Datei*
`-m` *Datei* Eine Umgebung für die Pakete erzeugen, die im Manifest-Objekt enthalten sind, das vom Scheme-Code in der *Datei* geliefert wird. Wenn diese Befehlszeilenoption mehrmals wiederholt angegeben wird, werden die Manifeste aneinandergehängt.

Dies verhält sich ähnlich wie die gleichnamige Option des Befehls `guix package` (siehe [profile-manifest], Seite 50) und benutzt auch dieselben Manifestdateien.

Siehe Abschnitt 9.4 [Manifeste verfassen], Seite 125, für Informationen dazu, wie man ein Manifest schreibt. Siehe `--export-manifest` im folgenden Text, um zu erfahren, wie Sie ein erstes Manifest erhalten.

`--export-manifest`
Auf die Standardausgabe ein Manifest ausgeben, das mit `--manifest` genutzt werden kann und den angegebenen Befehlszeilenoptionen entspricht.

Auf diese Weise können Sie die Argumente der Befehlszeile in eine Manifestdatei verwandeln. Nehmen wir an, Sie wären es leid, lange Befehlszeilen abzutippen, und hätten stattdessen lieber ein gleichbedeutendes Manifest:

```
guix shell -D guile git emacs emacs-geiser emacs-geiser-guile
```

Fügen Sie einfach `--export-manifest` zu der obigen Befehlszeile hinzu:

```
guix shell --export-manifest \
  -D guile git emacs emacs-geiser emacs-geiser-guile
```

... und schon sehen Sie ein Manifest, was so ähnlich aussieht:

```
(concatenate-manifests
  (list (specifications->manifest
          (list "git"
                "emacs"
                "emacs-geiser"
                "emacs-geiser-guile"))
        (package->development-manifest
          (specification->package "guile"))))
```

Diese Ausgabe können Sie in einer Datei speichern, nennen wir sie `manifest.scm`, die Sie dann an `guix shell` oder so ziemlich jeden beliebigen `guix`-Befehl übergeben:

```
guix shell -m manifest.scm
```

Na bitte. Aus der langen Befehlszeile ist ein Manifest geworden! Beim Verwandlungsvorgang werden Paketumwandlungsoptionen berücksichtigt (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192), d.h. er sollte verlustfrei sein.

`--profile=`*Profil*
`-p` *Profil* Eine Umgebung mit den Paketen erzeugen, die im *Profil* installiert sind. Benutzen Sie `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45), um Profile anzulegen und zu verwalten.

`--pure` Bestehende Umgebungsvariable deaktivieren, wenn die neue Umgebung erzeugt wird, mit Ausnahme der mit `--preserve` angegebenen Variablen (siehe unten). Dies bewirkt, dass eine Umgebung erzeugt wird, in der die Suchpfade nur Paketeingaben nennen und sonst nichts.

`--preserve=`*Regexp*
`-E` *Regexp* Wenn das hier zusammen mit `--pure` angegeben wird, bleiben die zum regulären Ausdruck *Regexp* passenden Umgebungsvariablen erhalten – mit anderen Worten werden sie auf eine „weiße Liste“ von Umgebungsvariablen gesetzt, die erhalten bleiben müssen. Diese Befehlszeilenoption kann mehrmals wiederholt werden.

```
guix shell --pure --preserve=^SLURM openmpi ... \
  -- mpirun ...
```

In diesem Beispiel wird `mpirun` in einem Kontext ausgeführt, in dem die einzig definierten Umgebungsvariablen `PATH` und solche sind, deren Name mit '`SLURM`' beginnt, sowie die üblichen besonders „kostbaren“ Variablen (`HOME`, `USER`, etc.).

`--search-paths`
Die Umgebungsvariablendefinitionen anzeigen, aus denen die Umgebung besteht.

`--system=`*System*
`-s` *System* Versuchen, für das angegebene *System* zu erstellen – z.B. `i686-linux`.

`--container`
`-C` Den *Befehl* in einer isolierten Umgebung (einem sogenannten „Container“) ausführen. Das aktuelle Arbeitsverzeichnis außerhalb des Containers wird in den Container zugeordnet. Zusätzlich wird, wenn es mit der Befehlszeilenoption `--user` nicht anders spezifiziert wurde, ein stellvertretendes persönliches Verzeichnis erzeugt, dessen Inhalt der des wirklichen persönlichen Verzeichnisses ist, sowie eine passend konfigurierte Datei `/etc/passwd`.

Der erzeugte Prozess läuft außerhalb des Containers als der momentane Nutzer. Innerhalb des Containers hat er dieselbe UID und GID wie der momentane Nutzer, außer die Befehlszeilenoption `--user` wird übergeben (siehe unten).

`--network`
`-N` Bei isolierten Umgebungen („Containern“) wird hiermit der Netzwerk-Namensraum mit dem des Wirtssystems geteilt. Container, die ohne diese Befehlszeilenoption erzeugt wurden, haben nur Zugriff auf das Loopback-Gerät.

`--link-profile`
`-P`
: Bei isolierten Umgebungen („Containern“) wird das Umgebungsprofil im Container als `~/.guix-profile` verknüpft und `~/.guix-profile` dann in `GUIX_ENVIRONMENT` gespeichert. Das ist äquivalent dazu, `~/.guix-profile` im Container zu einer symbolischen Verknüpfung auf das tatsächliche Profil zu machen. Wenn das Verzeichnis bereits existiert, schlägt das Verknüpfen fehl und die Umgebung wird nicht hergestellt. Dieser Fehler wird immer eintreten, wenn `guix shell` im persönlichen Verzeichnis des Benutzers aufgerufen wurde.

  Bestimmte Pakete sind so eingerichtet, dass sie in `~/.guix-profile` nach Konfigurationsdateien und Daten suchen,[2] weshalb `--link-profile` benutzt werden kann, damit sich diese Programme auch in der isolierten Umgebung wie erwartet verhalten.

`--user=`*Benutzer*
`-u` *Benutzer*
: Bei isolierten Umgebungen („Containern“) wird der Benutzername *Benutzer* anstelle des aktuellen Benutzers benutzt. Der erzeugte Eintrag in `/etc/passwd` im Container wird also den Namen *Benutzer* enthalten und das persönliche Verzeichnis wird den Namen `/home/BENUTZER` tragen; keine GECOS-Daten über den Nutzer werden in die Umgebung übernommen. Des Weiteren sind UID und GID innerhalb der isolierten Umgebung auf 1000 gesetzt. *Benutzer* muss auf dem System nicht existieren.

  Zusätzlich werden alle geteilten oder exponierten Pfade (siehe jeweils `--share` und `--expose`), deren Ziel innerhalb des persönlichen Verzeichnisses des aktuellen Benutzers liegt, relativ zu `/home/BENUTZER` erscheinen, einschließlich der automatischen Zuordnung des aktuellen Arbeitsverzeichnisses.

```
# wird Pfade als /home/foo/wd, /home/foo/test und /home/foo/target exponiere
cd $HOME/wd
guix shell --container --user=foo \
     --expose=$HOME/test \
     --expose=/tmp/target=$HOME/target
```

  Obwohl dies das Datenleck von Nutzerdaten durch Pfade im persönlichen Verzeichnis und die Benutzereinträge begrenzt, kann dies nur als Teil einer größeren Lösung für Datenschutz und Anonymität sinnvoll eingesetzt werden. Es sollte nicht für sich allein dazu eingesetzt werden.

`--no-cwd`
: In isolierten Umgebungen („Containern“) ist das vorgegebene Verhalten, das aktuelle Arbeitsverzeichnis mit dem isolierten Container zu teilen und in der Umgebung sofort in dieses Verzeichnis zu wechseln. Wenn das nicht gewünscht wird, kann das Angeben von `--no-cwd` dafür sorgen, dass das Arbeitsverzeichnis *nicht* automatisch geteilt wird und stattdessen in das Persönliche Verzeichnis („Home“-Verzeichnis) gewechselt wird. Siehe auch `--user`.

[2] Zum Beispiel inspiziert das Paket `fontconfig` das Verzeichnis `~/.guix-profile/share/fonts`, um zusätzliche Schriftarten zu finden.

`--expose=`*Quelle*`[=`*Ziel*`]`
`--share=`*Quelle*`[=`*Ziel*`]`
: Bei isolierten Umgebungen („Containern“) wird das Dateisystem unter *Quelle* vom Wirtssystem als Nur-Lese-Dateisystem *Ziel* (bzw. für `--share` auch mit Schreibrechten) im Container zugänglich gemacht. Wenn kein *Ziel* angegeben wurde, wird die *Quelle* auch als Ziel-Einhängepunkt in der isolierten Umgebung benutzt.

  Im folgenden Beispiel wird eine Guile-REPL in einer isolierten Umgebung gestartet, in der das persönliche Verzeichnis des Benutzers als Verzeichnis `/austausch` nur für Lesezugriffe zugänglich gemacht wurde:

```
guix shell --container --expose=$HOME=/austausch guile -- guile
```

`--symlink=`*Spezifikation*
`-S` *Spezifikation*
: Für Container werden hiermit die in der *Spezifikation* angegebenen symbolischen Verknüpfungen hergestellt. Siehe die Dokumentation in [pack-symlink-option], Seite 104.

`--emulate-fhs`
`-F`
: Wenn Sie dies zusammen mit `--container` angeben, wird im Container eine Konfiguration emuliert, die dem Filesystem Hierarchy Standard (FHS) (`https://refspecs.linuxfoundation.org/fhs.shtml`) folgt, so dass es `/bin`, `/lib` und weitere Verzeichnisse gibt, wie es der FHS vorschreibt.

  Obwohl Guix von der FHS-Spezifikation abweicht, kann das System in Ihrem Container mit dieser Befehlszeilenoption zu anderen GNU/Linux-Distributionen ähnlicher werden. Dadurch lassen sich dortige Entwicklungsumgebungen reproduzieren und Sie können Programme testen und benutzen, die das Befolgen der FHS-Spezifikation voraussetzen. Wenn die Option angegeben wird, enthält der Container eine Version von glibc, die die im Container befindliche `/etc/ld.so.cache` als Zwischenspeicher gemeinsamer Bibliotheken ausliest (anders als glibc im normalen Gebrauch von Guix), und die im FHS verlangten Verzeichnisse `/bin`, `/etc`, `/lib` und `/usr` werden aus dem Profil des Containers übernommen.

`--rebuild-cache`
: In der Regel wird `guix shell` die Umgebung zwischenspeichern, damit weitere Aufrufe sie ohne Verzögerung verwenden können. Am längsten *nicht* verwendete („least recently used“) Einträge im Zwischenspeicher werden bei `guix shell` regelmäßig entfernt. Auch wenn Sie `--file` oder `--manifest` nutzen, wird der Zwischenspeicher ungültig, sobald die entsprechende Datei geändert wurde.

  Mit der Befehlszeilenoption `--rebuild-cache` wird die zwischengespeicherte Umgebung erneuert. Sie können `--rebuild-cache` verwenden, wenn Sie `--file` oder `--manifest` nutzen und die Datei `guix.scm` oder `manifest.scm` auf externe Abhängigkeiten verweist oder eine andere Umgebung ergeben sollte, wenn sich zum Beispiel Umgebungsvariable geändert haben.

`--root=`*Datei*
`-r` *Datei*
: Die *Datei* zu einer symbolischen Verknüpfung auf das Profil dieser Umgebung machen und als eine Müllsammlerwurzel registrieren.

Das ist nützlich, um seine Umgebung vor dem Müllsammler zu schützen und sie „persistent“ zu machen.

Wenn diese Option weggelassen wird, wird das Profil von `guix shell` zwischengespeichert, damit weitere Aufrufe es ohne Verzögerung verwenden können. Das ist vergleichbar damit, wenn Sie `--root` angeben, jedoch wird `guix shell` die am längsten *nicht* verwendeten Müllsammlerwurzeln („least recently used“) regelmäßig entfernt.

In manchen Fällen wird `guix shell` keinen Zwischenspeicher für die Profile anlegen, z.B. wenn Umwandlungsoptionen wie `--with-latest` angegeben wurde. Das bedeutet, die Umgebung ist nur, solange die Sitzung von `guix shell` besteht, vor dem Müllsammler sicher. Dann müssen Sie, wenn Sie das nächste Mal dieselbe Umgebung neu erzeugen, vielleicht Pakete neu erstellen oder neu herunterladen.

Abschnitt 6.5 [Aufruf von guix gc], Seite 61, hat mehr Informationen über Müllsammlerwurzeln.

`guix shell` unterstützt auch alle gemeinsamen Erstellungsoptionen, die von `guix build` unterstützt werden (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189), und die Paketumwandlungsoptionen (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192).

8.2 `guix environment` aufrufen

Der Zweck von `guix environment` ist es, dass Sie Entwicklungsumgebungen leicht anlegen können.

Diese Funktionalität wird demnächst verschwinden: Der Befehl `guix environment` gilt als veraltet. Stattdessen sollten Sie `guix shell` benutzen, was einen ähnlichen Zweck erfüllt, aber leichter zu benutzen ist. Siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86.

Veraltet heißt, `guix environment` wird letztendlich entfernt, aber das Guix-Projekt wird gewährleisten, dass `guix environment` bis zum 1. Mai 2023 verfügbar bleibt. Reden Sie mit uns unter `guix-devel@gnu.org`, wenn Sie dabei mitdiskutieren möchten!

Die allgemeine Syntax lautet:

```
guix environment Optionen Paket...
```

Folgendes Beispiel zeigt, wie eine neue Shell gestartet wird, auf der alles für die Entwicklung von GNU Guile eingerichtet ist:

```
guix environment guile
```

Wenn benötigte Abhängigkeiten noch nicht erstellt worden sind, wird `guix environment` sie automatisch erstellen lassen. Die Umgebung der neuen Shell ist eine ergänzte Version der Umgebung, in der `guix environment` ausgeführt wurde. Sie enthält neben den existierenden Umgebungsvariablen auch die nötigen Suchpfade, um das angegebene Paket erstellen zu können. Um eine „reine“ Umgebung zu erstellen, in der die ursprünglichen Umgebungsvariablen nicht mehr vorkommen, kann die Befehlszeilenoption `--pure` benutzt werden[3].

[3] Manchmal ergänzen Nutzer fälschlicherweise Umgebungsvariable wie `PATH` in ihrer `~/.bashrc`-Datei. Das hat zur Folge, dass wenn `guix environment` Bash startet, selbige `~/.bashrc` von Bash gelesen wird und

Eine Guix-Umgebung zu verlassen ist gleichbedeutend mit dem Verlassen der Shell; danach findet sich der Benutzer in der alten Umgebung wieder, in der er sich vor dem Aufruf von `guix environment` befand. Beim nächsten Durchlauf des Müllsammlers (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61) werden Pakete von der Platte gelöscht, die innerhalb der Umgebung installiert worden sind und außerhalb nicht länger benutzt werden.

`guix environment` definiert die Variable `GUIX_ENVIRONMENT` in der neu erzeugten Shell. Ihr Wert ist der Dateiname des Profils dieser neuen Umgebung. Das könnten Nutzer verwenden, um zum Beispiel eine besondere Prompt als Eingabeaufforderung für Entwicklungsumgebungen in ihrer `.bashrc` festzulegen (siehe Abschnitt "Bash Startup Files" in *Referenzhandbuch von GNU Bash*):

```
if [ -n "$GUIX_ENVIRONMENT" ]
then
    export PS1="\u@\h \w [dev]\$ "
fi
```

... oder um ihr Profil durchzusehen:

```
$ ls "$GUIX_ENVIRONMENT/bin"
```

Des Weiteren kann mehr als ein Paket angegeben werden. In diesem Fall wird die Vereinigung der Eingaben der jeweiligen Pakete zugänglich gemacht. Zum Beispiel erzeugt der folgende Befehl eine Shell, in der alle Abhängigkeiten von sowohl Guile als auch Emacs verfügbar sind:

```
guix environment guile emacs
```

Manchmal will man keine interaktive Shell-Sitzung. Ein beliebiger Befehl kann aufgerufen werden, indem Sie nach Angabe der Pakete noch `--` vor den gewünschten Befehl schreiben, um ihn von den übrigen Argumenten abzutrennen:

```
guix environment guile -- make -j4
```

In anderen Situationen ist es bequemer, aufzulisten, welche Pakete in der Umgebung benötigt werden. Zum Beispiel führt der folgende Befehl `python` aus einer Umgebung heraus aus, in der Python 3 und NumPy enthalten sind:

```
guix environment --ad-hoc python-numpy python -- python3
```

Man kann auch sowohl die Abhängigkeiten eines Pakets haben wollen als auch ein paar zusätzliche Pakete, die nicht Erstellungs- oder Laufzeitabhängigkeiten davon sind, aber trotzdem bei der Entwicklung nützlich sind. Deshalb hängt die Wirkung von der Position der Befehlszeilenoption `--ad-hoc` ab. Pakete, die links von `--ad-hoc` stehen, werden als Pakete interpretiert, deren Abhängigkeiten zur Umgebung hinzugefügt werden. Pakete, die rechts stehen, werden selbst zur Umgebung hinzugefügt. Zum Beispiel erzeugt der folgende Befehl eine Guix-Entwicklungsumgebung, die zusätzlich Git und strace umfasst:

```
guix environment --pure guix --ad-hoc git strace
```

Manchmal ist es wünschenswert, die Umgebung so viel wie möglich zu isolieren, um maximale Reinheit und Reproduzierbarkeit zu bekommen. Insbesondere ist es wünschenswert, den Zugriff auf `/usr/bin` und andere Systemressourcen aus der Entwicklungsumgebung

---

die neuen Umgebungen somit „verunreinigt“. Es ist ein Fehler, solche Umgebungsvariable in `.bashrc` zu definieren, stattdessen sollten sie in `.bash_profile` geschrieben werden, was nur von Login-Shells mit „source“ geladen wird. Siehe Abschnitt "Bash Startup Files" in *Referenzhandbuch zu GNU Bash* für Details über beim Starten von Bash gelesene Dateien

heraus zu verhindern, wenn man Guix auf einer fremden Wirtsdistribution benutzt, die nicht Guix System ist. Zum Beispiel startet der folgende Befehl eine Guile-REPL in einer isolierten Umgebung, einem sogenannten „Container“, in der nur der Store und das aktuelle Arbeitsverzeichnis eingebunden sind:

```
guix environment --ad-hoc --container guile -- guile
```

Anmerkung: Die Befehlszeilenoption `--container` funktioniert nur mit Linux-libre 3.19 oder neuer.

Ein weiterer typischer Anwendungsfall für Container ist, gegenüber Sicherheitslücken anfällige Anwendungen auszuführen, z.B. Webbrowser. Um Eolie auszuführen, müssen wir den Zugriff auf manche Dateien und Verzeichnisse über `--expose` und `--share` gewähren. Wir lassen `nss-certs` bereitstellen und machen `/etc/ssl/certs/` für die HTTPS-Authentifizierung sichtbar. Zu guter Letzt behalten wir die `DISPLAY`-Umgebungsvariable bei, weil isolierte grafische Anwendungen ohne sie nicht angezeigt werden können.

```
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
```

Im Folgenden werden die verfügbaren Befehlszeilenoptionen zusammengefasst.

`--check` Hiermit wird die Umgebung angelegt und gemeldet, ob die Shell Umgebungsvariable überschreiben würde. Siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86, für weitere Informationen.

`--root=`*Datei*
`-r` *Datei* Die *Datei* zu einer symbolischen Verknüpfung auf das Profil dieser Umgebung machen und als eine Müllsammlerwurzel registrieren.

Das ist nützlich, um seine Umgebung vor dem Müllsammler zu schützen und sie „persistent“ zu machen.

Wird diese Option weggelassen, ist die Umgebung nur, solange die Sitzung von `guix environment` besteht, vor dem Müllsammler sicher. Das bedeutet, wenn Sie das nächste Mal dieselbe Umgebung neu erzeugen, müssen Sie vielleicht Pakete neu erstellen oder neu herunterladen. Abschnitt 6.5 [Aufruf von guix gc], Seite 61, hat mehr Informationen über Müllsammlerwurzeln.

`--expression=`*Ausdruck*
`-e` *Ausdruck*
Eine Umgebung für das Paket oder die Liste von Paketen erzeugen, zu der der *Ausdruck* ausgewertet wird.

Zum Beispiel startet dies:

```
guix environment -e '(@ (gnu packages maths) petsc-openmpi)'
```

eine Shell mit der Umgebung für eben diese bestimmte Variante des Pakets PETSc.

Wenn man dies ausführt:

```
guix environment --ad-hoc -e '(@ (gnu) %base-packages)'
```

bekommt man eine Shell, in der alle Basis-Pakete verfügbar sind.

Die obigen Befehle benutzen nur die Standard-Ausgabe des jeweiligen Pakets. Um andere Ausgaben auszuwählen, können zweielementige Tupel spezifiziert werden:

```
guix environment --ad-hoc -e '(list (@ (gnu packages bash) bash) "include")'
```

`--load=`*Datei*

`-l` *Datei* Eine Umgebung erstellen für das Paket oder die Liste von Paketen, zu der der Code in der *Datei* ausgewertet wird.

Zum Beispiel könnte die *Datei* eine Definition wie diese enthalten (siehe Abschnitt 9.2 [Pakete definieren], Seite 109):

```
(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-2.64 automake texinfo))))
```

`--manifest=`*Datei*

`-m` *Datei* Eine Umgebung für die Pakete erzeugen, die im Manifest-Objekt enthalten sind, das vom Scheme-Code in der *Datei* geliefert wird. Wenn diese Befehlszeilenoption mehrmals wiederholt angegeben wird, werden die Manifeste aneinandergehängt.

Dies verhält sich ähnlich wie die gleichnamige Option des Befehls `guix package` (siehe [profile-manifest], Seite 50) und benutzt auch dieselben Manifestdateien.

Siehe [shell-export-manifest], Seite 89, für Informationen, wie Sie Befehlszeilenoptionen in ein Manifest „verwandeln“ können.

`--ad-hoc` Alle angegebenen Pakete in der resultierenden Umgebung einschließen, als wären sie Eingaben eines *ad hoc* definierten Pakets. Diese Befehlszeilenoption ist nützlich, um schnell Umgebungen aufzusetzen, ohne dafür einen Paketausdruck schreiben zu müssen, der die gewünschten Eingaben enthält.

Zum Beispiel wird mit diesem Befehl:

```
guix environment --ad-hoc guile guile-sdl -- guile
```

`guile` in einer Umgebung ausgeführt, in der sowohl Guile als auch Guile-SDL zur Verfügung stehen.

Beachten Sie, dass in diesem Beispiel implizit die vorgegebene Ausgabe von `guile` und `guile-sdl` verwendet wird, es aber auch möglich ist, eine bestimmte Ausgabe auszuwählen – z.B. wird mit `glib:bin` die Ausgabe `bin` von `glib` gewählt (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60).

Diese Befehlszeilenoption kann mit dem standardmäßigen Verhalten von `guix environment` verbunden werden. Pakete, die vor `--ad-hoc` aufgeführt werden,

werden als Pakete interpretiert, deren Abhängigkeiten zur Umgebung hinzugefügt werden, was dem standardmäßigen Verhalten entspricht. Pakete, die danach aufgeführt werden, werden selbst zur Umgebung hinzugefügt.

`--profile=`*Profil*
`-p` *Profil* Eine Umgebung mit den Paketen erzeugen, die im *Profil* installiert sind. Benutzen Sie `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45), um Profile anzulegen und zu verwalten.

`--pure` Bestehende Umgebungsvariable deaktivieren, wenn die neue Umgebung erzeugt wird, mit Ausnahme der mit `--preserve` angegebenen Variablen (siehe unten). Dies bewirkt, dass eine Umgebung erzeugt wird, in der die Suchpfade nur Paketeingaben nennen und sonst nichts.

`--preserve=`*Regexp*
`-E` *Regexp* Wenn das hier zusammen mit `--pure` angegeben wird, bleiben die zum regulären Ausdruck *Regexp* passenden Umgebungsvariablen erhalten – mit anderen Worten werden sie auf eine „weiße Liste“ von Umgebungsvariablen gesetzt, die erhalten bleiben müssen. Diese Befehlszeilenoption kann mehrmals wiederholt werden.

```
guix environment --pure --preserve=^SLURM --ad-hoc openmpi ... \
  -- mpirun ...
```

In diesem Beispiel wird `mpirun` in einem Kontext ausgeführt, in dem die einzig definierten Umgebungsvariablen `PATH` und solche sind, deren Name mit ‘`SLURM`’ beginnt, sowie die üblichen besonders „kostbaren“ Variablen (`HOME`, `USER`, etc.).

`--search-paths`
Die Umgebungsvariablendefinitionen anzeigen, aus denen die Umgebung besteht.

`--system=`*System*
`-s` *System* Versuchen, für das angegebene *System* zu erstellen – z.B. `i686-linux`.

`--container`
`-C` Den *Befehl* in einer isolierten Umgebung (einem sogenannten „Container“) ausführen. Das aktuelle Arbeitsverzeichnis außerhalb des Containers wird in den Container zugeordnet. Zusätzlich wird, wenn es mit der Befehlszeilenoption `--user` nicht anders spezifiziert wurde, ein stellvertretendes persönliches Verzeichnis erzeugt, dessen Inhalt der des wirklichen persönlichen Verzeichnisses ist, sowie eine passend konfigurierte Datei `/etc/passwd`.

Der erzeugte Prozess läuft außerhalb des Containers als der momentane Nutzer. Innerhalb des Containers hat er dieselbe UID und GID wie der momentane Nutzer, außer die Befehlszeilenoption `--user` wird übergeben (siehe unten).

`--network`
`-N` Bei isolierten Umgebungen („Containern“) wird hiermit der Netzwerk-Namensraum mit dem des Wirtssystems geteilt. Container, die ohne diese Befehlszeilenoption erzeugt wurden, haben nur Zugriff auf das Loopback-Gerät.

`--link-profile`
`-P` Bei isolierten Umgebungen („Containern“) wird das Umgebungsprofil im Container als `~/.guix-profile` verknüpft und `~/.guix-profile` dann in `GUIX_`

`ENVIRONMENT` gespeichert. Das ist äquivalent dazu, `~/.guix-profile` im Container zu einer symbolischen Verknüpfung auf das tatsächliche Profil zu machen. Wenn das Verzeichnis bereits existiert, schlägt das Verknüpfen fehl und die Umgebung wird nicht hergestellt. Dieser Fehler wird immer eintreten, wenn `guix environment` im persönlichen Verzeichnis des Benutzers aufgerufen wurde.

Bestimmte Pakete sind so eingerichtet, dass sie in `~/.guix-profile` nach Konfigurationsdateien und Daten suchen,[4] weshalb `--link-profile` benutzt werden kann, damit sich diese Programme auch in der isolierten Umgebung wie erwartet verhalten.

`--user=`*Benutzer*
`-u` *Benutzer*

Bei isolierten Umgebungen („Containern“) wird der Benutzername *Benutzer* anstelle des aktuellen Benutzers benutzt. Der erzeugte Eintrag in `/etc/passwd` im Container wird also den Namen *Benutzer* enthalten und das persönliche Verzeichnis wird den Namen `/home/BENUTZER` tragen; keine GECOS-Daten über den Nutzer werden in die Umgebung übernommen. Des Weiteren sind UID und GID innerhalb der isolierten Umgebung auf 1000 gesetzt. *Benutzer* muss auf dem System nicht existieren.

Zusätzlich werden alle geteilten oder exponierten Pfade (siehe jeweils `--share` und `--expose`), deren Ziel innerhalb des persönlichen Verzeichnisses des aktuellen Benutzers liegt, relativ zu `/home/BENUTZER` erscheinen, einschließlich der automatischen Zuordnung des aktuellen Arbeitsverzeichnisses.

```
# wird Pfade als /home/foo/wd, /home/foo/test und /home/foo/target exponiere
cd $HOME/wd
guix environment --container --user=foo \
     --expose=$HOME/test \
     --expose=/tmp/target=$HOME/target
```

Obwohl dies das Datenleck von Nutzerdaten durch Pfade im persönlichen Verzeichnis und die Benutzereinträge begrenzt, kann dies nur als Teil einer größeren Lösung für Datenschutz und Anonymität sinnvoll eingesetzt werden. Es sollte nicht für sich allein dazu eingesetzt werden.

`--no-cwd` In isolierten Umgebungen („Containern“) ist das vorgegebene Verhalten, das aktuelle Arbeitsverzeichnis mit dem isolierten Container zu teilen und in der Umgebung sofort in dieses Verzeichnis zu wechseln. Wenn das nicht gewünscht wird, kann das Angeben von `--no-cwd` dafür sorgen, dass das Arbeitsverzeichnis *nicht* automatisch geteilt wird und stattdessen in das Persönliche Verzeichnis („Home“-Verzeichnis) gewechselt wird. Siehe auch `--user`.

`--expose=`*Quelle*`[=`*Ziel*`]`
`--share=`*Quelle*`[=`*Ziel*`]`

Bei isolierten Umgebungen („Containern“) wird das Dateisystem unter *Quelle* vom Wirtssystem als Nur-Lese-Dateisystem *Ziel* (bzw. für `--share` auch mit Schreibrechten) im Container zugänglich gemacht. Wenn kein *Ziel* angegeben

[4] Zum Beispiel inspiziert das Paket `fontconfig` das Verzeichnis `~/.guix-profile/share/fonts`, um zusätzliche Schriftarten zu finden.

wurde, wird die *Quelle* auch als Ziel-Einhängepunkt in der isolierten Umgebung benutzt.

Im folgenden Beispiel wird eine Guile-REPL in einer isolierten Umgebung gestartet, in der das persönliche Verzeichnis des Benutzers als Verzeichnis `/austausch` nur für Lesezugriffe zugänglich gemacht wurde:

```
guix environment --container --expose=$HOME=/austausch --ad-hoc guile -- gui
```

`--emulate-fhs`
`-F` In Containern wird eine Konfiguration emuliert, die dem Filesystem Hierarchy Standard (FHS) folgt, siehe dessen offizielle Spezifikation (`https://refspecs.linuxfoundation.org/fhs.shtml`). Obwohl Guix von der FHS-Spezifikation abweicht, kann das System in Ihrem Container mit dieser Befehlszeilenoption zu anderen GNU/Linux-Distributionen ähnlicher werden. Dadurch lassen sich dortige Entwicklungsumgebungen reproduzieren und Sie können Programme testen und benutzen, die das Befolgen der FHS-Spezifikation voraussetzen. Wenn die Option angegeben wird, enthält der Container eine Version von glibc, die die im Container befindliche `/etc/ld.so.cache` als Zwischenspeicher gemeinsamer Bibliotheken ausliest (anders als glibc im normalen Gebrauch von Guix), und die im FHS verlangten Verzeichnisse `/bin`, `/etc`, `/lib` und `/usr` werden aus dem Profil des Containers übernommen.

`guix environment` unterstützt auch alle gemeinsamen Erstellungsoptionen, die von `guix build` unterstützt werden (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189), und die Paketumwandlungsoptionen (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192).

8.3 `guix pack` aufrufen

Manchmal möchten Sie Software an Leute weitergeben, die (noch!) nicht das Glück haben, Guix zu benutzen. Mit Guix würden sie nur `guix package -i` *`irgendetwas`* einzutippen brauchen, aber wenn sie kein Guix haben, muss es anders gehen. Hier kommt `guix pack` ins Spiel.

> **Anmerkung:** Wenn Sie aber nach einer Möglichkeit suchen, Binärdateien unter Maschinen auszutauschen, auf denen Guix bereits läuft, sollten Sie einen Blick auf die Abschnitte Abschnitt 10.13 [Aufruf von guix copy], Seite 241, Abschnitt 10.11 [Aufruf von guix publish], Seite 233, und Abschnitt 6.10 [Aufruf von guix archive], Seite 74, werfen.

Der Befehl `guix pack` erzeugt ein gut verpacktes *Software-Bündel*: Konkret wird dadurch ein Tarball oder eine andere Art von Archiv mit den Binärdateien der Software erzeugt, die Sie sich gewünscht haben, zusammen mit all ihren Abhängigkeiten. Der resultierende Archiv kann auch auf jeder Maschine genutzt werden, die kein Guix hat, und jeder kann damit genau dieselben Binärdateien benutzen, die Ihnen unter Guix zur Verfügung stehen. Das Bündel wird dabei auf eine Bit für Bit reproduzierbare Art erzeugt, damit auch jeder nachprüfen kann, dass darin wirklich diejenigen Binärdateien enthalten sind, von denen Sie es behaupten.

Um zum Beispiel ein Bündel mit Guile, Emacs, Geiser und all ihren Abhängigkeiten zu erzeugen, führen Sie diesen Befehl aus:

```
$ guix pack guile emacs emacs-geiser
...
/gnu/store/...-pack.tar.gz
```

Als Ergebnis erhalten Sie einen Tarball mit einem Verzeichnis `/gnu/store`, worin sich alles relevanten Pakete befinden. Der resultierende Tarball enthält auch ein *Profil* mit den drei angegebenen Paketen; es ist dieselbe Art von Profil, die auch `guix package -i` erzeugen würde. Mit diesem Mechanismus wird auch der binäre Tarball zur Installation von Guix erzeugt (siehe Abschnitt 2.1 [Aus Binärdatei installieren], Seite 5).

Benutzer des Bündels müssten dann aber zum Beispiel `/gnu/store/...-profile/bin/guile` eintippen, um Guile auszuführen, was Ihnen zu unbequem sein könnte. Ein Ausweg wäre, dass Sie etwa eine symbolische Verknüpfung `/opt/gnu/bin` auf das Profil anlegen:

```
guix pack -S /opt/gnu/bin=bin guile emacs emacs-geiser
```

Benutzer müssten dann nur noch `/opt/gnu/bin/guile` eintippen, um Guile zu genießen.

Doch was ist, wenn die Empfängerin Ihres Bündels keine Administratorrechte auf ihrer Maschine hat, das Bündel also nicht ins Wurzelverzeichnis ihres Dateisystems entpacken kann? Dann möchten Sie vielleicht die Befehlszeilenoption `--relocatable` benutzen (siehe weiter unten). Mit dieser Option werden *verschiebliche Binärdateien* erzeugt, die auch in einem beliebigen anderen Verzeichnis in der Dateisystemhierarchie abgelegt und von dort ausgeführt werden können. Man könnte sagen, sie sind pfad-agnostisch. In obigem Beispiel würden Benutzer Ihren Tarball in ihr Persönliches Verzeichnis (das „Home“-Verzeichnis) entpacken und von dort den Befehl `./opt/gnu/bin/guile` ausführen.

Eine weitere Möglichkeit ist, das Bündel im Format eines Docker-Abbilds (englisch Docker-Image) zu erzeugen. Das geht mit dem folgenden Befehl:

```
guix pack -f docker -S /bin=bin guile guile-readline
```

Das Ergebnis ist ein Tarball, der dem Befehl `docker load` übergeben werden kann, gefolgt von `docker run`:

```
docker load < Datei
docker run -ti guile-guile-readline /bin/guile
```

Dabei steht *Datei* für das durch *guix pack* gelieferte Abbild und `guile-guile-readline` für den „Image-Tag“, der diesem zugewiesen wurde. In der Dokumentation von Docker (`https://docs.docker.com/engine/reference/commandline/load/`) finden Sie nähere Informationen.

Und noch eine weitere Möglichkeit ist, dass Sie ein SquashFS-Abbild mit folgendem Befehl erzeugen:

```
guix pack -f squashfs bash guile emacs emacs-geiser
```

Das Ergebnis ist ein SquashFS-Dateisystemabbild, dass entweder als Dateisystem eingebunden oder mit Hilfe der Singularity-Container-Ausführungsumgebung (`https://www.sylabs.io/docs/`) als Dateisystemcontainer benutzt werden kann, mit Befehlen wie `singularity shell` oder `singularity exec`.

Es gibt mehrere Befehlszeilenoptionen, mit denen Sie Ihr Bündel anpassen können:

`--format=`*Format*
`-f` *Format* Generiert ein Bündel im angegebenen *Format*.

Die verfügbaren Formate sind:

`tarball` Das standardmäßig benutzte Format. Damit wird ein Tarball generiert, der alle angegebenen Binärdateien und symbolischen Verknüpfungen enthält.

`docker` Generiert einen Tarball gemäß der Spezifikation für Docker-Abbilder (`https://github.com/docker/docker/blob/master/image/spec/v1.2.md`). Der „Repository-Name“, wie er in der Ausgabe des Befehls `docker images` erscheint, wird anhand der Paketnamen berechnet, die auf der Befehlszeile oder in der Manifest-Datei angegeben wurden.

`squashfs` Generiert ein SquashFS-Abbild, das alle angegebenen Binärdateien und symbolischen Verknüpfungen enthält, sowie leere Einhängepunkte für virtuelle Dateisysteme wie procfs.

> **Anmerkung:** Für Singularity *müssen* Sie eine `/bin/sh` in das Abbild aufnehmen. Aus diesem Grund gilt für `guix pack -f squashfs` implizit immer auch `-S /bin=bin`. Daher muss Ihr Aufruf von `guix pack` immer ungefähr so beginnen:
>
> ```
> guix pack -f squashfs bash ...
> ```
>
> Wenn Sie vergessen, das `bash`-Paket (oder etwas Ähnliches) zu bündeln, werden `singularity run` und `singularity exec` mit der wenig hilfreichen Meldung „Datei oder Verzeichnis nicht gefunden“ scheitern.

`deb` Hiermit wird ein Debian-Archiv erzeugt (ein Paket mit der Dateinamenserweiterung „`.deb`“), in dem alle angegebenen Binärdateien und symbolischen Verknüpfungen enthalten sind. Es kann auf jeder dpkg-basierten GNU(/Linux)-Distribution installiert werden. Fortgeschrittene Optionen werden Ihnen angezeigt, wenn Sie die Befehlszeilenoption `--help-deb-format` angeben. Mit jenen können Control-Dateien hinzugefügt werden für eine genaue Steuerung z.B. welche bestimmten Trigger aktiviert werden oder um mit einem Betreuerskript beliebigen Konfigurations-Code einzufügen, der bei Installation ausgeführt werden soll.

```
guix pack -f deb -C xz -S /usr/bin/hello=bin/hello hello
```

> **Anmerkung:** Weil in den mit `guix pack` erzeugten Archiven eine Ansammlung von Store-Objekten enthalten ist und weil in jedem `dpkg`-Paket keine im Konflikt stehenden Dateien enthalten sein dürfen, können Sie *de facto* wahrscheinlich nur ein einziges mit `guix pack` erzeugtes „`.deb`“-Archiv je System installieren.
>
> **Warnung:** `dpkg` übernimmt die Zuständigkeit für alle Dateien, die im Bündel enthalten sind, auch wenn es die Dateien *nicht* kennt. Es ist unklug, mit Guix erzeugte „`.deb`“-Dateien auf einem System zu installie-

ren, auf dem `/gnu/store` bereits von anderer Software verwendet wird, also wenn z.B. Guix bereits installiert ist oder andere Nicht-`deb`-Bündel installiert sind.

`--relocatable`
`-R` Erzeugt *verschiebliche Binärdateien* – also pfad-agnostische, „portable“ Binärdateien, die an einer beliebigen Stelle in der Dateisystemhierarchie platziert und von dort ausgeführt werden können.

Wenn diese Befehlszeilenoption einmal übergeben wird, funktionieren die erzeugten Binärdateien nur dann, wenn *Benutzernamensräume* des Linux-Kernels unterstützt werden. Wenn sie *zweimal*[5] übergeben wird, laufen die Binärdateien notfalls mit anderen Methoden, wenn keine Benutzernamensräume zur Verfügung stehen, funktionieren also ziemlich überall – siehe unten für die Auswirkungen.

Zum Beispiel können Sie ein Bash enthalltendes Bündel erzeugen mit:

```
guix pack -RR -S /meine-bin=bin bash
```

. . . Sie können dieses dann auf eine Maschine ohne Guix kopieren und als normaler Nutzer aus Ihrem Persönlichen Verzeichnis (auch „Home“-Verzeichnis genannt) dann ausführen mit:

```
tar xf pack.tar.gz
./meine-bin/sh
```

Wenn Sie in der so gestarteten Shell dann `ls /gnu/store` eintippen, sehen Sie, dass Ihnen angezeigt wird, in `/gnu/store` befänden sich alle Abhängigkeiten von `bash`, obwohl auf der Maschine überhaupt kein Verzeichnis `/gnu/store` existiert! Dies ist vermutlich die einfachste Art, mit Guix erstellte Software für eine Maschine ohne Guix auszuliefern.

Anmerkung: Wenn die Voreinstellung verwendet wird, funktionieren verschiebliche Binärdateien nur mit *Benutzernamensräumen* (englisch *User namespaces*), einer Funktionalität des Linux-Kernels, mit der Benutzer ohne besondere Berechtigungen Dateisysteme einbinden (englisch „mount“) oder die Wurzel des Dateisystems wechseln können („change root“, kurz „chroot“). Alte Versionen von Linux haben diese Funktionalität noch nicht unterstützt und manche Distributionen von GNU/Linux schalten sie ab.

Um verschiebliche Binärdateien zu erzeugen, die auch ohne Benutzernamensräume funktionieren, können Sie die Befehlszeilenoption `--relocatable` oder `-R` *zweimal* angeben. In diesem Fall werden die Binärdateien zuerst überprüfen, ob Benutzernamensräume unterstützt werden, und sonst notfalls einen anderen *Ausführungstreiber* benutzen, um das Programm auszuführen, wenn Benutzernamensräume nicht unterstützt werden. Folgende Ausführungstreiber werden unterstützt:

[5] Es gibt einen Trick, wie Sie sich das merken können: `-RR`, womit PRoot-Unterstützung hinzugefügt wird, kann man sich als Abkürzung für „Rundum Relocatable“ oder englisch „Really Relocatable“ vorstellen. Ist das nicht prima?

`default` Es wird versucht, Benutzernamensräume zu verwenden. Sind Benutzernamensräume nicht verfügbar (siehe unten), wird auf PRoot zurückgegriffen.

`performance`
Es wird versucht, Benutzernamensräume zu verwenden. Sind Benutzernamensräume nicht verfügbar (siehe unten), wird auf Fakechroot zurückgegriffen.

`userns` Das Programm wird mit Hilfe von Benutzernamensräumen ausgeführt. Wenn sie nicht unterstützt werden, bricht das Programm ab.

`proot` Durch PRoot ausführen. Das Programm PRoot (`https://proot-me.github.io/`) bietet auch Unterstützung für Dateisystemvirtualisierung, indem der Systemaufruf `ptrace` auf das laufende Programm angewendet wird. Dieser Ansatz funktioniert auch ohne besondere Kernel-Unterstützung, aber das Programm braucht mehr Zeit, um selbst Systemaufrufe durchzuführen.

`fakechroot`
Durch Fakechroot laufen lassen. Fakechroot (`https://github.com/dex4er/fakechroot/`) virtualisiert Dateisystemzugriffe, indem Aufrufe von Funktionen der C-Bibliothek wie `open`, `stat`, `exec` und so weiter abgefangen werden. Anders als bei PRoot entsteht dabei kaum Mehraufwand. Jedoch funktioniert das nicht immer, zum Beispiel werden manche Dateisystemzugriffe aus der C-Bibliothek heraus *nicht* abgefangen, ebenso wenig wie Dateisystemaufrufe über direkte Systemaufrufe, was zu fehlerbehaftetem Verhalten führt.

Wenn Sie ein verpacktes Programm ausführen, können Sie einen der oben angeführten Ausführungstreiber ausdrücklich anfordern, indem Sie die Umgebungsvariable `GUIX_EXECUTION_ENGINE` entsprechend festlegen.

`--entry-point=`*`Befehl`*
Den *Befehl* als den *Einsprungpunkt* des erzeugten Bündels verwenden, wenn das Bündelformat einen solchen unterstützt – derzeit tun das `docker` und `squashfs` (Singularity). Der *Befehl* wird relativ zum Profil ausgeführt, das sich im Bündel befindet.

Der Einsprungpunkt gibt den Befehl an, der mit `docker run` oder `singularity run` beim Start nach Voreinstellung automatisch ausgeführt wird. Zum Beispiel können Sie das hier benutzen:

```
guix pack -f docker --entry-point=bin/guile guile
```

Dann kann das erzeugte Bündel mit z.B. `docker run` ohne weitere Befehlszeilenargumente einfach geladen und ausgeführt werden, um `bin/guile` zu starten:

```
docker load -i pack.tar.gz
docker run Abbild-ID
```

`--expression=`*Ausdruck*
`-e` *Ausdruck*

Als Paket benutzen, wozu der *Ausdruck* ausgewertet wird.

Der Zweck hiervon ist derselbe wie bei der gleichnamigen Befehlszeilenoption in `guix build` (siehe Abschnitt 10.1.3 [Zusätzliche Erstellungsoptionen], Seite 198).

`--manifest=`*Datei*
`-m` *Datei*

Die Pakete benutzen, die im Manifest-Objekt aufgeführt sind, das vom Scheme-Code in der angegebenen *Datei* geliefert wird. Wenn diese Befehlszeilenoption mehrmals wiederholt angegeben wird, werden die Manifeste aneinandergehängt.

Dies hat einen ähnlichen Zweck wie die gleichnamige Befehlszeilenoption in `guix package` (siehe [profile-manifest], Seite 50) und benutzt dieselben Regeln für Manifest-Dateien. Damit können Sie eine Reihe von Paketen einmal definieren und dann sowohl zum Erzeugen von Profilesn als auch zum Erzeugen von Archiven benutzen, letztere für Maschinen, auf denen Guix nicht installiert ist. Beachten Sie, dass Sie *entweder* eine Manifest-Datei *oder* eine Liste von Paketen angeben können, aber nicht beides.

Siehe Abschnitt 9.4 [Manifeste verfassen], Seite 125, für Informationen dazu, wie man ein Manifest schreibt. Siehe [shell-export-manifest], Seite 89, für Informationen, wie Sie Befehlszeilenoptionen in ein Manifest „verwandeln“ können.

`--system=`*System*
`-s` *System*

Versuchen, für die angegebene Art von *System* geeignete Binärdateien zu erstellen – z.B. `i686-linux` – statt für die Art von System, das die Erstellung durchführt.

`--target=`*Tripel*

Lässt für das angegebene *Tripel* cross-erstellen, dieses muss ein gültiges GNU-Tripel wie z.B. `"aarch64-linux-gnu"` sein (siehe Abschnitt “Specifying target triplets” in *Autoconf*).

`--compression=`*Werkzeug*
`-C` *Werkzeug*

Komprimiert den resultierenden Tarball mit dem angegebenen *Werkzeug* – dieses kann `gzip`, `zstd`, `bzip2`, `xz`, `lzip` oder `none` für keine Kompression sein.

`--symlink=`*Spezifikation*
`-S` *Spezifikation*

Fügt die in der *Spezifikation* festgelegten symbolischen Verknüpfungen zum Bündel hinzu. Diese Befehlszeilenoption darf mehrmals vorkommen.

Die *Spezifikation* muss von der Form `Quellort=Zielort` sein, wobei der *Quellort* der Ort der symbolischen Verknüpfung, die erstellt wird, und *Zielort* das Ziel der symbolischen Verknüpfung ist.

Zum Beispiel wird mit `-S /opt/gnu/bin=bin` eine symbolische Verknüpfung `/opt/gnu/bin` auf das Unterverzeichnis `bin` im Profil erzeugt.

`--save-provenance`
: Provenienzinformationen für die auf der Befehlszeile übergebenen Pakete speichern. Zu den Provenienzinformationen gehören die URL und der Commit jedes benutzten Kanals (siehe Kapitel 7 [Kanäle], Seite 77).

  Provenienzinformationen werden in der Datei `/gnu/store/...-profile/manifest` im Bündel zusammen mit den üblichen Paketmetadaten abgespeichert – also Name und Version jedes Pakets, welche Eingaben dabei propagiert werden und so weiter. Die Informationen nützen den Empfängern des Bündels, weil sie dann wissen, woraus das Bündel (angeblich) besteht.

  Der Vorgabe nach wird diese Befehlszeilenoption *nicht* verwendet, weil Provenienzinformationen genau wie Zeitstempel nichts zum Erstellungsprozess beitragen. Mit anderen Worten gibt es unendlich viele Kanal-URLs und Commit-IDs, aus denen dasselbe Bündel stammen könnte. Wenn solche „stillen“ Metadaten Teil des Ausgabe sind, dann wird also die bitweise Reproduzierbarkeit von Quellcode zu Binärdateien eingeschränkt.

`--root=`*Datei*
`-r` *Datei*
: Die *Datei* zu einer symbolischen Verknüpfung auf das erzeugte Bündel machen und als Müllsammlerwurzel registrieren.

`--localstatedir`
`--profile-name=`*Name*
: Das „lokale Zustandsverzeichnis“ `/var/guix` ins resultierende Bündel aufnehmen, speziell auch das Profil `/var/guix/profiles/per-user/root/`*Name* – der vorgegebene *Name* ist `guix-profile`, was `~root/.guix-profile` entspricht.

  `/var/guix` enthält die Store-Datenbank (siehe Abschnitt 9.9 [Der Store], Seite 165) sowie die Müllsammlerwurzeln (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61). Es ins Bündel aufzunehmen, bedeutet, dass der enthaltene Store „vollständig“ ist und von Guix verwaltet werden kann, andernfalls wäre der Store im Bündel „tot“ und nach dem Auspacken des Bündels könnte Guix keine Objekte mehr dort hinzufügen oder entfernen.

  Ein Anwendungsfall hierfür ist der eigenständige, alle Komponenten umfassende binäre Tarball von Guix (siehe Abschnitt 2.1 [Aus Binärdatei installieren], Seite 5).

`--derivation`
`-d`
: Den Namen der Ableitung ausgeben, die das Bündel erstellt.

`--bootstrap`
: Mit den Bootstrap-Binärdateien das Bündel erstellen. Diese Option ist nur für Guix-Entwickler nützlich.

Außerdem unterstützt `guix pack` alle gemeinsamen Erstellungsoptionen (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189) und alle Paketumwandlungsoptionen (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192).

8.4 GCC-Toolchain

Wenn Sie einen vollständigen Werkzeugsatz zum Kompilieren und Binden von Quellcode in C oder C++ brauchen, werden Sie das Paket `gcc-toolchain` haben wollen. Das Paket bietet eine vollständige GCC-Toolchain für die Entwicklung mit C/C++, einschließlich GCC selbst, der GNU-C-Bibliothek (Header-Dateien und Binärdateien samt Symbolen zur Fehlersuche/Debugging in der `debug`-Ausgabe), Binutils und einen Wrapper für den Binder/Linker.

Der Zweck des Wrappers ist, die an den Binder übergebenen Befehlszeilenoptionen mit `-L` und `-l` zu überprüfen und jeweils passende Argumente mit `-rpath` anzufügen, womit dann der echte Binder aufgerufen wird. Standardmäßig weigert sich der Binder-Wrapper, mit Bibliotheken außerhalb des Stores zu binden, um „Reinheit“ zu gewährleisten. Das kann aber stören, wenn man die Toolchain benutzt, um mit lokalen Bibliotheken zu binden. Um Referenzen auf Bibliotheken außerhalb des Stores zu erlauben, müssen Sie die Umgebungsvariable `GUIX_LD_WRAPPER_ALLOW_IMPURITIES` setzen.

Das Paket `gfortran-toolchain` stellt eine vollständige GCC-Toolchain für die Entwicklung mit Fortran zur Verfügung. Pakete für die Entwicklung mit anderen Sprachen suchen Sie bitte mit ‘`guix search gcc toolchain`’ (siehe [Invoking guix package], Seite 52).

8.5 `guix git authenticate` aufrufen

Der Befehl `guix git authenticate` authentifiziert ein Git-Checkout nach derselben Regel wie für Kanäle (siehe [channel-authentication], Seite 79). Das bedeutet, dass angefangen beim angegebenen Commit sichergestellt wird, dass alle nachfolgenden Commits mit einem OpenPGP-Schlüssel signiert worden sind, dessen Fingerabdruck in der `.guix-authorizations`-Datei seines bzw. seiner jeweiligen Elterncommits aufgeführt ist.

Sie werden diesen Befehl zu schätzen wissen, wenn Sie einen Kanal betreuen. Tatsächlich ist dieser Authentifizierungsmechanismus aber auch bei weiteren Dingen nützlich; vielleicht möchten Sie ihn für Git-Repositorys einsetzen, die gar nichts mit Guix zu tun haben?

Die allgemeine Syntax lautet:

```
guix git authenticate Commit Unterzeichner [Optionen...]
```

Nach Vorgabe wird mit diesem Befehl das Git-Checkout im aktuellen Arbeitsverzeichnis authentifiziert. Es wird bei Erfolg nichts ausgegeben und der Exit-Status null zurückgeliefert; bei einem Fehler wird ein von null verschiedener Exit-Status zurückgeliefert. *Commit* gibt den ersten Commit an, der authentifiziert wird, und *Unterzeichner* ist der OpenPGP-Fingerabdruck des öffentlichen Schlüssels, mit dem der *Commit* signiert wurde. Zusammen bilden sie eine „Kanaleinführung“ (siehe [channel-authentication], Seite 79). Mit den unten aufgeführten Befehlszeilenoptionen können Sie Feineinstellungen am Prozess vornehmen.

`--repository=`*Verzeichnis*
`-r` *Verzeichnis*
: Das Git-Repository im *Verzeichnis* statt im aktuellen Verzeichnis öffnen.

`--keyring=`*Referenz*
`-k` *Referenz*
: Den OpenPGP-Schlüsselbund („Keyring“) von der angegebenen *Referenz* laden, einem Verweis auf einen Branch wie `origin/keyring` oder `my-keyring`.

Der Branch muss öffentliche Schlüssel im OpenPGP-Format in `.key`-Dateien enthalten, entweder als Binärdateien oder mit „ASCII-Hülle“. Nach Vorgabe wird der Schlüsselbund von einem Branch namens `keyring` geladen.

`--stats` Nach Abschluss Statistiken über die signierten Commits anzeigen.

`--cache-key=`*Schlüssel*
Bereits authentifizierte Commits werden in einer Datei unter `~/.cache/guix/authentication` zwischengespeichert. Diese Option erzwingt, dass der Speicher innerhalb dieses Verzeichnisses in der Datei *Schlüssel* angelegt wird.

`--historical-authorizations=`*Datei*
Nach Vorgabe wird jeder Commit, dessen Elterncommit(s) die Datei `.guix-authorizations` fehlt, als gefälscht angesehen. Mit dieser Option werden dagegen die Autorisierungen in der *Datei* für jeden Commit ohne `.guix-authorizations` verwendet. Das Format der *Datei* ist dasselbe wie bei `.guix-authorizations` (siehe [channel-authorizations], Seite 82).

9 Programmierschnittstelle

GNU Guix bietet mehrere Programmierschnittstellen (APIs) in der Programmiersprache Scheme an, mit denen Software-Pakete definiert, erstellt und gesucht werden können. Die erste Schnittstelle erlaubt es Nutzern, ihre eigenen Paketdefinitionen in einer Hochsprache zu schreiben. Diese Definitionen nehmen Bezug auf geläufige Konzepte der Paketverwaltung, wie den Namen und die Version eines Pakets, sein Erstellungssystem (Build System) und seine Abhängigkeiten (Dependencies). Diese Definitionen können dann in konkrete Erstellungsaktionen umgewandelt werden.

Erstellungsaktionen werden vom Guix-Daemon für dessen Nutzer durchgeführt. Bei einer normalen Konfiguration hat der Daemon Schreibzugriff auf den Store, also das Verzeichnis `/gnu/store`, Nutzer hingegen nicht. Die empfohlene Konfiguration lässt den Daemon die Erstellungen in chroot-Umgebungen durchführen, mit eigenen Benutzerkonten für „Erstellungsbenutzer“, um gegenseitige Beeinflussung der Erstellung und des übrigen Systems zu minimieren.

Systemnahe APIs stehen zur Verfügung, um mit dem Daemon und dem Store zu interagieren. Um den Daemon anzuweisen, eine Erstellungsaktion durchzuführen, versorgen ihn Nutzer jeweils mit einer *Ableitung*. Eine Ableitung ist, wie durchzuführende Erstellungsaktionen, sowie die Umgebungen, in denen sie durchzuführen sind, in Guix eigentlich intern dargestellt werden. Ableitungen verhalten sich zu Paketdefinitionen vergleichbar mit Assembler-Code zu C-Programmen. Der Begriff „Ableitung“ kommt daher, dass Erstellungsergebnisse daraus *abgeleitet* werden.

Dieses Kapitel beschreibt der Reihe nach all diese Programmierschnittstellen (APIs), angefangen mit hochsprachlichen Paketdefinitionen.

9.1 Paketmodule

Aus Programmierersicht werden die Paketdefinitionen der GNU-Distribution als Guile-Module in Namensräumen wie `(gnu packages ...)` sichtbar gemacht[1] (siehe Abschnitt “Modules” in *Referenzhandbuch zu GNU Guile*). Zum Beispiel exportiert das Modul `(gnu packages emacs)` eine Variable namens `emacs`, die an ein `<package>`-Objekt gebunden ist (siehe Abschnitt 9.2 [Pakete definieren], Seite 109).

Der Modulnamensraum `(gnu packages ...)` wird von Befehlszeilenwerkzeugen automatisch nach Paketen durchsucht. Wenn Sie zum Beispiel `guix install emacs` ausführen, werden alle `(gnu packages ...)`-Module durchlaufen, bis eines gefunden wird, das ein Paketobjekt mit dem Namen `emacs` exportiert. Diese Paketsuchfunktion ist im Modul `(gnu packages)` implementiert.

[1] Beachten Sie, dass Pakete unter dem Modulnamensraum `(gnu packages ...)` nicht notwendigerweise auch „GNU-Pakete“ sind. Dieses Schema für die Benennung von Modulen folgt lediglich den üblichen Guile-Konventionen: `gnu` bedeutet, dass die Module als Teil des GNU-Systems ausgeliefert werden, und `packages` gruppiert Module mit Paketdefinitionen.

Benutzer können Paketdefinitionen auch in Modulen mit anderen Namen unterbringen – z.B. `(my-packages emacs)`[2]. Es gibt zwei Arten, solche Paketdefinitionen für die Benutzungsschnittstelle sichtbar zu machen:

1. Eine Möglichkeit ist, das Verzeichnis, in dem Ihre Paketmodule stehen, mit der Befehlszeilenoption `-L` von `guix package` und anderen Befehlen (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189) oder durch Setzen der unten beschriebenen Umgebungsvariablen `GUIX_PACKAGE_PATH` zum Suchpfad hinzuzufügen.
2. Die andere Möglichkeit ist, einen *Kanal* zu definieren und `guix pull` so zu konfigurieren, dass es davon seine Module bezieht. Ein Kanal ist im Kern nur ein Git-Repository, in welchem Paketmodule liegen. Siehe Kapitel 7 [Kanäle], Seite 77, für mehr Informationen, wie Kanäle definiert und benutzt werden.

`GUIX_PACKAGE_PATH` funktioniert ähnlich wie andere Variable mit Suchpfaden:

`GUIX_PACKAGE_PATH` [Umgebungsvariable]
: Dies ist eine doppelpunktgetrennte Liste von Verzeichnissen, die nach zusätzlichen Paketmodulen durchsucht werden. In dieser Variablen aufgelistete Verzeichnisse haben Vorrang vor den Modulen, die zur Distribution gehören.

Die Distribution wird komplett von Grund auf initialisiert – man sagt zur Initialisierung auch *Bootstrapping* – und sie ist *eigenständig* („self-contained“): Jedes Paket wird nur auf Basis von anderen Paketen in der Distribution erstellt. Die Wurzel dieses Abhängigkeitsgraphen ist ein kleiner Satz von Initialisierungsbinärdateien, den *Bootstrap-Binärdateien*, die im Modul `(gnu packages bootstrap)` verfügbar gemacht werden. Für mehr Informationen über Bootstrapping, siehe Kapitel 20 [Bootstrapping], Seite 699.

9.2 Pakete definieren

Mit den Modulen `(guix packages)` und `(guix build-system)` können Paketdefinitionen auf einer hohen Abstraktionsebene geschrieben werden. Zum Beispiel sieht die Paketdefinition bzw. das *Rezept* für das Paket von GNU Hello so aus:

```
(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
```

[2] Beachten Sie, dass Dateiname und Modulname übereinstimmen müssen. Zum Beispiel muss das Modul `(my-packages emacs)` in einer Datei `my-packages/emacs.scm` relativ zum mit `--load-path` oder `GUIX_PACKAGE_PATH` angegebenen Ladepfad stehen. Siehe Abschnitt “Modules and the File System” in *Referenzhandbuch zu GNU Guile* für Details.

```
            (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+)))
```

Auch ohne ein Experte in Scheme zu sein, könnten Leser erraten haben, was die verschiedenen Felder dabei bedeuten. Dieser Ausdruck bindet die Variable `hello` an ein `<package>`-Objekt, was an sich nur ein Verbund (Record) ist (siehe Abschnitt "SRFI-9" in *Referenzhandbuch zu GNU Guile*). Die Felder dieses Paket-Objekts lassen sich mit den Prozeduren aus dem Modul `(guix packages)` auslesen, zum Beispiel liefert `(package-name hello)` – Überraschung! – `"hello"`.

Mit etwas Glück können Sie die Definition vielleicht teilweise oder sogar ganz aus einer anderen Paketsammlung importieren, indem Sie den Befehl `guix import` verwenden (siehe Abschnitt 10.5 [Aufruf von guix import], Seite 206).

In obigem Beispiel wurde `hello` in einem eigenen Modul ganz für sich alleine definiert, und zwar `(gnu packages hello)`. Technisch gesehen muss es nicht unbedingt in einem solchen Modul definiert werden, aber es ist bequem, denn alle Module unter `(gnu packages ...)` werden automatisch von den Befehlszeilenwerkzeugen gefunden (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

Ein paar Dinge sind noch erwähnenswert in der obigen Paketdefinition:

- Das `source`-Feld für die Quelle des Pakets ist ein `<origin>`-Objekt, was den Paketursprung angibt (siehe Abschnitt 9.2.2 [„origin"-Referenz], Seite 118, für eine vollständige Referenz). Hier wird dafür die Methode `url-fetch` aus dem Modul `(guix download)` benutzt, d.h. die Quelle ist eine Datei, die über FTP oder HTTP heruntergeladen werden soll.

  Das Präfix `mirror://gnu` lässt `url-fetch` einen der GNU-Spiegelserver benutzen, die in `(guix download)` definiert sind.

  Das Feld `sha256` legt den erwarteten SHA256-Hashwert der herunterzuladenden Datei fest. Ihn anzugeben ist Pflicht und er ermöglicht es Guix, die Integrität der Datei zu überprüfen. Die Form `(base32 ...)` geht der base32-Darstellung des Hash-Wertes voraus. Sie finden die base32-Darstellung mit Hilfe der Befehle `guix download` (siehe Abschnitt 10.3 [Aufruf von guix download], Seite 204) und `guix hash` (siehe Abschnitt 10.4 [Aufruf von guix hash], Seite 205).

  Wenn nötig kann in der `origin`-Form auch ein `patches`-Feld stehen, wo anzuwendende Patches aufgeführt werden, sowie ein `snippet`-Feld mit einem Scheme-Ausdruck mit den Anweisungen, wie der Quellcode zu modifizieren ist.

- Das Feld `build-system` legt fest, mit welcher Prozedur das Paket erstellt werden soll (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). In diesem Beispiel steht `gnu-build-system` für das wohlbekannte GNU-Erstellungssystem, wo Pakete mit der üblichen Befehlsfolge `./configure && make && make check && make install` konfiguriert, erstellt und installiert werden.

  Sobald Sie anfangen, Pakete für nichttriviale Software zu schreiben, könnten Sie Werkzeuge benötigen, um jene Erstellungsphasen abzuändern, Dateien zu verändern oder Ähnliches. Siehe Abschnitt 9.7 [Werkzeuge zur Erstellung], Seite 154, für mehr Informationen dazu.

- Das Feld `arguments` gibt an, welche Optionen dem Erstellungssystem mitgegeben werden sollen (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). In diesem Fall interpretiert `gnu-build-system` diese als Auftrag, `configure` mit der Befehlszeilenoption `--enable-silent-rules` auszuführen.

  Was hat es mit diesen einfachen Anführungszeichen (`'`) auf sich? Sie gehören zur Syntax von Scheme und führen eine wörtlich zu interpretierende Datenliste ein; dies nennt sich Maskierung oder Quotierung. `'` ist synonym mit `quote`. Ihnen könnte auch `` ` `` begegnen (ein Backquote, stattdessen kann man auch das längere Synonym `quasiquote` schreiben); damit können wir eine wörtlich als Daten interpretierte Liste einführen, aber bei dieser „Quasimaskierung“ kann `,` (ein Komma, oder dessen Synonym `unquote`) benutzt werden, um den ausgewerteten Wert eines Ausdrucks in diese Liste einzufügen. Abschnitt “Expression Syntax” in *Referenzhandbuch zu GNU Guile* enthält weitere Details. Hierbei ist also der Wert des `arguments`-Feldes eine Liste von Argumenten, die an das Erstellungssystem weitergereicht werden, wie bei `apply` (siehe Abschnitt “Fly Evaluation” in *Referenzhandbuch zu GNU Guile*).

  Ein Doppelkreuz gefolgt von einem Doppelpunkt (`#:`) definiert ein Scheme-*Schlüsselwort* (siehe Abschnitt “Keywords” in *Referenzhandbuch zu GNU Guile*) und `#:configure-flags` ist ein Schlüsselwort, um eine Befehlszeilenoption an das Erstellungssystem mitzugeben (siehe Abschnitt “Coding With Keywords” in *Referenzhandbuch zu GNU Guile*).

- Das Feld `inputs` legt Eingaben an den Erstellungsprozess fest – d.h. Abhängigkeiten des Pakets zur Erstellungs- oder Laufzeit. Hier fügen wir eine Eingabe hinzu, eine Referenz auf die Variable `gawk`; *gawk* ist auch selbst wiederum an ein `<package>`-Objekt als Variablenwert gebunden.

  Beachten Sie, dass GCC, Coreutils, Bash und andere essenzielle Werkzeuge hier nicht als Eingaben aufgeführt werden müssen. Stattdessen sorgt schon `gnu-build-system` dafür, dass diese vorhanden sein müssen (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130).

  Sämtliche anderen Abhängigkeiten müssen aber im `inputs`-Feld aufgezählt werden. Jede hier nicht angegebene Abhängigkeit wird während des Erstellungsprozesses schlicht nicht verfügbar sein, woraus ein Erstellungsfehler resultieren kann.

Siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113, für eine umfassende Beschreibung aller erlaubten Felder.

Vertiefung:

Fühlen Sie sich eingeschüchtert von der Scheme-Sprache oder wurde Ihre Neugier geweckt? Im Kochbuch gibt es einen kurzen Abschnitt, wie man anfängt.

> Dort werden einige der oben gezeigten Dinge wiederholt und die Grundlagen erklärt. Siehe Abschnitt “Ein Schnellkurs in Scheme” in *GNU-Guix-Kochbuch* für weitere Informationen.

Sobald eine Paketdefinition eingesetzt wurde, können Sie das Paket mit Hilfe des Befehlszeilenwerkzeugs `guix build` dann auch tatsächlich erstellen (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189) und dabei jegliche Erstellungsfehler, auf die Sie stoßen, beseitigen (siehe Abschnitt 10.1.4 [Fehlschläge beim Erstellen untersuchen], Seite 203). Sie können den Befehl `guix edit` benutzen, um leicht zur Paketdefinition zurückzuspringen (siehe Abschnitt 10.2 [Aufruf von guix edit], Seite 204). Unter Abschnitt 22.4 [Paketrichtlinien], Seite 713, finden Sie mehr Informationen darüber, wie Sie Paketdefinitionen testen, und unter Abschnitt 10.8 [Aufruf von guix lint], Seite 223, finden Sie Informationen, wie Sie prüfen, ob eine Definition alle Stilkonventionen einhält.

Zuletzt finden Sie unter Kapitel 7 [Kanäle], Seite 77, Informationen, wie Sie die Distribution um Ihre eigenen Pakete in einem „Kanal“ erweitern.

Zu all dem sei auch erwähnt, dass Sie das Aktualisieren einer Paketdefinition auf eine vom Anbieter neu veröffentlichte Version mit dem Befehl `guix refresh` teilweise automatisieren können (siehe Abschnitt 10.6 [Aufruf von guix refresh], Seite 214).

Hinter den Kulissen wird die einem `<package>`-Objekt entsprechende Ableitung zuerst durch `package-derivation` berechnet. Diese Ableitung wird in der `.drv`-Datei unter `/gnu/store` gespeichert. Die von ihr vorgeschriebenen Erstellungsaktionen können dann durch die Prozedur `build-derivations` umgesetzt werden (siehe Abschnitt 9.9 [Der Store], Seite 165).

`package-derivation` *`Store Paket`* [*`System`*] [Scheme-Prozedur]

> Das `<derivation>`-Objekt zum *Paket* für das angegebene *System* liefern (siehe Abschnitt 9.10 [Ableitungen], Seite 167).
>
> Als *Paket* muss ein gültiges `<package>`-Objekt angegeben werden und das *System* muss eine Zeichenkette sein, die das Zielsystem angibt – z.B. `"x86_64-linux"` für ein auf x86_64 laufendes, Linux-basiertes GNU-System. *Store* muss eine Verbindung zum Daemon sein, der die Operationen auf dem Store durchführt (siehe Abschnitt 9.9 [Der Store], Seite 165).

Auf ähnliche Weise kann eine Ableitung berechnet werden, die ein Paket für ein anderes System cross-erstellt.

`package-cross-derivation` *`Store Paket Ziel`* [*`System`*] [Scheme-Prozedur]

> *Liefert das*
> `<derivation>`-Objekt, um das *Paket* zu cross-erstellen vom *System* aus für das *Ziel*-System.
>
> Als *Ziel* muss ein gültiges GNU-Tripel angegeben werden, was die Ziel-Hardware und das zugehörige Betriebssystem beschreibt, wie z.B. `"aarch64-linux-gnu"` (siehe Abschnitt “Specifying Target Triplets” in *Autoconf*).

Wenn Sie einmal Paketdefinitionen fertig verfasst haben, können Sie leicht *Varianten* derselben Pakete definieren. Siehe Abschnitt 9.3 [Paketvarianten definieren], Seite 121, für mehr Informationen dazu.

9.2.1 `package`-Referenz

Dieser Abschnitt fasst alle in `package`-Deklarationen zur Verfügung stehenden Optionen zusammen (siehe Abschnitt 9.2 [Pakete definieren], Seite 109).

`package` [Datentyp]

Dieser Datentyp steht für ein Paketrezept.

`name` Der Name des Pakets als Zeichenkette.

`version` Die Version des Pakets als Zeichenkette. Siehe Abschnitt 22.4.3 [Versionsnummern], Seite 715, wo erklärt wird, worauf Sie achten müssen.

`source` Ein Objekt, das beschreibt, wie der Quellcode des Pakets bezogen werden soll. Meistens ist es ein `origin`-Objekt, welches für eine aus dem Internet heruntergeladene Datei steht (siehe Abschnitt 9.2.2 [„origin“-Referenz], Seite 118). Es kann aber auch ein beliebiges anderes „dateiähnliches“ Objekt sein, wie z.B. ein `local-file`, was eine Datei im lokalen Dateisystem bezeichnet (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

`build-system`
Das Erstellungssystem, mit dem das Paket erstellt werden soll (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130).

`arguments` (Vorgabe: `'()`)
Die Argumente, die an das Erstellungssystem übergeben werden sollen (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). Dies ist eine Liste, typischerweise eine Reihe von Schlüssel-Wert-Paaren wie in diesem Beispiel:

```
(package
  (name "beispiel")
  ;; hier würden noch ein paar Felder stehen
  (arguments
    (list #:tests? #f                     ;Tests überspringen
          #:make-flags #~'("VERBOSE=1")   ;Optionen für 'make'
          #:configure-flags #~'("--enable-frobbing"))))
```

Welche Schlüsselwortargumente genau unterstützt werden, hängt vom jeweiligen Erstellungssystem ab (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130), doch werden Sie feststellen, dass fast alle `#:configure-flags`, `#:make-flags`, `#:tests?` und `#:phases` berücksichtigen. Insbesondere können Sie mit dem Schlüsselwort `#:phases` den Satz von Erstellungsphasen, der für Ihr Paket zum Einsatz kommt, ändern (siehe Abschnitt 9.6 [Erstellungsphasen], Seite 150).

`inputs` (Vorgabe: `'()`)
`native-inputs` (Vorgabe: `'()`)
`propagated-inputs` (Vorgabe: `'()`)
In diesen Feldern wird eine Liste der Abhängigkeiten des Pakets aufgeführt. Dabei ist jedes Listenelement ein „package“-, „origin“- oder sonstiges dateiartiges Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

Um die zu benutzende Ausgabe zu benennen, übergeben Sie eine zweielementige Liste mit der Ausgabe als zweitem Element (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60, für mehr Informationen zu Paketausgaben). Im folgenden Beispiel etwa werden drei Eingaben festgelegt:

```
(list libffi libunistring
      `(,glib "bin"))      ;Ausgabe "bin" von GLib
```

Im obigen Beispiel wird die Ausgabe `"out"` für `libffi` und `libunistring` benutzt.

> **Anmerkung zur Kompatibilität:** Bis Version 1.3.0 waren Eingabelisten noch Listen von Tupeln, wobei jedes Tupel eine Bezeichnung für die Eingabe (als Zeichenkette) als erstes Element, dann ein „package"-, „origin"- oder „derivation"-Objekt (Paket, Ursprung oder Ableitung) als zweites Element und optional die Benennung der davon zu benutzenden Ausgabe umfasste; letztere hatte als Vorgabewert `"out"`. Im folgenden Beispiel wird dasselbe wie oben angegeben, aber im *alten Stil für Eingaben*:
>
> ```
> ;; Alter Eingabenstil (veraltet).
> `(("libffi" ,libffi)
>   ("libunistring" ,libunistring)
>   ("glib:bin" ,glib "bin"))  ;Ausgabe "bin" von GLib
> ```
>
> Dieser Stil gilt als veraltet. Er wird bislang unterstützt, aber er wird in einer zukünftigen Version entfernt werden. Man sollte ihn nicht für neue Paketdefinitionen gebrauchen. Siehe Abschnitt 10.7 [Aufruf von guix style], Seite 221, für eine Erklärung, wie Sie zum neuen Stil migrieren.

Die Unterscheidung zwischen `native-inputs` und `inputs` ist wichtig, damit Cross-Kompilieren möglich ist. Beim Cross-Kompilieren werden als `inputs` aufgeführte Abhängigkeiten für die Ziel-Prozessorarchitektur (*target*) erstellt, andersherum werden als `native-inputs` aufgeführte Abhängigkeiten für die Prozessorarchitektur der erstellenden Maschine (*build*) erstellt.

`native-inputs` listet typischerweise die Werkzeuge auf, die während der Erstellung gebraucht werden, aber nicht zur Laufzeit des Programms gebraucht werden. Beispiele sind Autoconf, Automake, pkg-config, Gettext oder Bison. `guix lint` kann melden, ob wahrscheinlich Fehler in der Auflistung sind (siehe Abschnitt 10.8 [Aufruf von guix lint], Seite 223).

Schließlich ist `propagated-inputs` ähnlich wie `inputs`, aber die angegebenen Pakete werden automatisch mit ins Profil installiert (siehe Abschnitt 6.1 [Funktionalitäten], Seite 44), wenn das Paket installiert wird, zu dem sie gehören (siehe [package-cmd-propagated-inputs], Seite 47, für Informationen darüber, wie `guix package` mit propagierten Eingaben umgeht).

Dies ist zum Beispiel nötig, wenn Sie ein Paket für eine C-/C++-Bibliothek schreiben, die Header-Dateien einer anderen Bibliothek braucht, um mit ihr kompilieren zu können, oder wenn sich eine pkg-config-Datei auf eine andere über ihren `Requires`-Eintrag bezieht.

Noch ein Beispiel, wo `propagated-inputs` nützlich ist, sind Sprachen, die den Laufzeit-Suchpfad *nicht* zusammen mit dem Programm abspeichern (*nicht* wie etwa im `RUNPATH` bei ELF-Dateien), also Sprachen wie Guile, Python, Perl und weitere. Wenn Sie ein Paket für eine in solchen Sprachen geschriebene Bibliothek schreiben, dann sorgen Sie dafür, dass es zur Laufzeit den von ihr benötigten Code finden kann, indem Sie ihre Laufzeit-Abhängigkeiten in `propagated-inputs` statt in `inputs` aufführen.

`outputs` (Vorgabe: `'("out")`)
: Die Liste der Benennungen der Ausgaben des Pakets. Der Abschnitt Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60, beschreibt übliche Nutzungen zusätzlicher Ausgaben.

`native-search-paths` (Vorgabe: `'()`)
`search-paths` (Vorgabe: `'()`)
: Eine Liste von `search-path-specification`-Objekten, die Umgebungsvariable für von diesem Paket beachtete Suchpfade („search paths“) beschreiben. Siehe Abschnitt 9.8 [Suchpfade], Seite 161, wenn Sie mehr über Suchpfadspezifikationen erfahren möchten.

  Wie auch bei Eingaben wird zwischen `native-search-paths` und `search-paths` nur dann ein Unterschied gemacht, wenn Sie crosskompilieren. Beim Cross-Kompilieren bezieht sich `native-search-paths` ausschließlich auf native Eingaben, wohingegen sich die `search-paths` ausschließlich auf Eingaben für den „Host“ genannten Rechner beziehen.

  Für Pakete wie z.B. Cross-Compiler sind die Ziel-Eingaben wichtig, z.B. hat unser (angepasster) GCC-Cross-Compiler einen Eintrag für `CROSS_C_INCLUDE_PATH` in `search-paths`, damit er `.h`-Dateien für das Zielsystem auswählt und *nicht* die aus nativen Eingaben. Für die meisten Pakete ist aber einzig `native-search-paths` von Bedeutung.

`replacement` (Vorgabe: `#f`)
: Dies muss entweder `#f` oder ein package-Objekt sein, das als Ersatz (*replacement*) dieses Pakets benutzt werden soll. Im Abschnitt zu Kapitel 19 [Sicherheitsaktualisierungen], Seite 697, wird dies erklärt.

`synopsis` Eine einzeilige Beschreibung des Pakets.

`description`
: Eine ausführlichere Beschreibung des Pakets, als eine Zeichenkette in Texinfo-Syntax.

`license` Die Lizenz des Pakets; benutzt werden kann ein Wert aus dem Modul `(guix licenses)` oder eine Liste solcher Werte.

`home-page`
: Die URL, die die Homepage des Pakets angibt, als Zeichenkette.

`supported-systems` (Vorgabe: `%supported-systems`)
: Die Liste der vom Paket unterstützten Systeme als Zeichenketten der Form `Architektur-Kernel`, zum Beispiel `"x86_64-linux"`.

`location` (Vorgabe: die Stelle im Quellcode, wo die `package`-Form steht)
: Wo im Quellcode das Paket definiert wurde. Es ist sinnvoll, dieses Feld manuell zuzuweisen, wenn das Paket von einem anderen Paket erbt, weil dann dieses Feld nicht automatisch berichtigt wird.

`this-package` [Scheme-Syntax]
: Wenn dies im *lexikalischen Geltungsbereich* der Definition eines Feldes im Paket steht, bezieht sich dieser Bezeichner auf das Paket, das gerade definiert wird.

  Das folgende Beispiel zeigt, wie man ein Paket als native Eingabe von sich selbst beim Cross-Kompilieren deklariert:

```
(package
  (name "guile")
  ;; ...

  ;; Wenn es cross-kompiliert wird, hängt zum Beispiel
  ;; Guile von einer nativen Version von sich selbst ab.
  ;; Wir fügen sie hier hinzu.
  (native-inputs (if (%current-target-system)
                     (list this-package)
                     '())))
```

  Es ist ein Fehler, außerhalb einer Paketdefinition auf `this-package` zu verweisen.

Die folgenden Hilfsprozeduren sind für den Umgang mit Paketeingaben gedacht.

`lookup-package-input` *Paket Name* [Scheme-Prozedur]
`lookup-package-native-input` *Paket Name* [Scheme-Prozedur]
`lookup-package-propagated-input` *Paket Name* [Scheme-Prozedur]
`lookup-package-direct-input` *Paket Name* [Scheme-Prozedur]
: *Name* unter den (nativen, propagierten, direkten) Eingaben von *Paket* suchen und die Eingabe zurückliefern, wenn sie gefunden wird, ansonsten `#f`.

  Name ist der Name eines Pakets, von dem eine Abhängigkeit besteht. So benutzen Sie die Prozeduren:

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

(lookup-package-direct-input coreutils "gmp")
⇒ #<package gmp@6.2.1 ...>
```

  In diesem Beispiel erhalten wir das `gmp`-Paket, das zu den direkten Eingaben von `coreutils` gehört.

Manchmal werden Sie die Liste der zur Entwicklung an einem Paket nötigen Eingaben brauchen, d.h. alle Eingaben, die beim Kompilieren des Pakets sichtbar gemacht werden. Diese Liste wird von der Prozedur `package-development-inputs` geliefert.

`package-development-inputs` *Paket* [*System*] [*#:target #f*] [Scheme-Prozedur]
Liefert die Liste derjenigen Eingaben, die das *Paket* zu Entwicklungszwecken für *System* braucht. Wenn *target* wahr ist, muss dafür ein Ziel wie das GNU-Tripel `"aarch64-linux-gnu"` übergeben werden, damit die Eingaben zum Cross-Kompilieren von *Paket* zurückgeliefert werden.

Beachten Sie, dass dazu sowohl explizite als auch implizite Eingaben gehören. Mit impliziten Eingaben meinen wir solche, die vom Erstellungssystem automatisch hinzugefügt werden (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). Nehmen wir das Paket `hello` zur Illustration:

```
(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 ...>) ...)
```

Für dieses Beispiel liefert `package-direct-inputs` die leere Liste zurück, weil `hello` keinerlei explizite Abhängigkeiten hat. Zu `package-development-inputs` gehören auch durch `gnu-build-system` implizit hinzugefügte Eingaben, die für die Erstellung von `hello` gebraucht werden, nämlich tar, gzip, GCC, libc, Bash und weitere. Wenn Sie sich das anschauen wollen, zeigt Ihnen `guix graph hello` die expliziten Eingaben, dagegen zeigt `guix graph -t bag hello` auch die impliziten Eingaben (siehe Abschnitt 10.10 [Aufruf von guix graph], Seite 228).

Weil Pakete herkömmliche Scheme-Objekte sind, die einen vollständigen Abhängigkeitsgraphen und die zugehörigen Erstellungsprozeduren umfassen, bietet es sich oftmals an, Prozeduren zu schreiben, die ein Paket entgegennehmen und in Abhängigkeit bestimmter Parameter eine abgeänderte Fassung desselben zurückliefern. Es folgen einige Beispiele.

`package-with-c-toolchain` *Paket Toolchain* [Scheme-Prozedur]
Liefert eine Variante des *Pakets*, die die angegebene *Toolchain* anstelle der vorgegebenen GNU-C/C++-Toolchain benutzt. Als *Toolchain* muss eine Liste von Eingaben (als Tupel aus Bezeichnung und bezeichnetem Paket) angegeben werden, die eine gleichartige Funktion erfüllen, wie zum Beispiel das Paket `gcc-toolchain`.

Das folgende Beispiel liefert eine Variante des Pakets `hello`, die mit GCC 10.x und den übrigen Komponenten der GNU-Toolchain (Binutils und GNU-C-Bibliothek) erstellt wurde statt mit der vorgegebenen Toolchain:

```
(let ((toolchain (specification->package "gcc-toolchain@10")))
  (package-with-c-toolchain hello `(("toolchain" ,toolchain))))
```

Die Erstellungs-Toolchain gehört zu den *impliziten Eingaben* von Paketen – sie wird normalerweise nicht ausdrücklich unter den verschiedenen „inputs“-Feldern mit verschiedenen Arten von Eingaben aufgeführt, stattdessen kommt sie über das Erstel-

lungssystem dazu. Daher funktioniert diese Prozedur intern so, dass sie das Erstellungssystem des *Pakets* verändert, damit es die ausgewählte *Toolchain* statt der vorgegebenen benutzt. Siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130, für weitere Informationen zu Erstellungssystemen.

9.2.2 `origin`-Referenz

In diesem Abschnitt werden Paketursprünge – englisch *Origins* – beschrieben. Eine `origin`-Deklaration legt Daten fest, die „produziert“ werden müssen – normalerweise heißt das heruntergeladen. Die Hash-Prüfsumme von deren Inhalt muss dabei im Voraus bekannt sein. Ursprünge werden in erster Linie benutzt, um den Quellcode von Paketen zu repräsentieren (siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Aus diesem Grund können Sie mit der `origin`-Form Patches angeben, die auf den ursprünglichen Quellcode angewandt werden sollen, oder auch Schnipsel von Code, der Veränderungen daran vornimmt.

`origin` [Datentyp]

Mit diesem Datentyp wird ein Ursprung, von dem Quellcode geladen werden kann, beschrieben.

`uri` Ein Objekt, was die URI des Quellcodes enthält. Der Objekttyp hängt von der `Methode` ab (siehe unten). Zum Beispiel sind, wenn die *url-fetch*-Methode aus `(guix download)` benutzt wird, die gültigen Werte für `uri`: eine URL dargestellt als Zeichenkette oder eine Liste solcher URLs.

`method` Eine monadische Prozedur, um die angegebene URL zu benutzen. Die Prozedur muss mindestens drei Argumente akzeptieren: den Wert des `uri`-Feldes, den Hash-Algorithmus und den Hash-Wert, der im `hash`-Feld angegeben wird. Sie muss ein Store-Objekt oder eine Ableitung in der Store-Monade liefern (siehe Abschnitt 9.11 [Die Store-Monade], Seite 170). Die meisten Methoden liefern eine Ableitung mit fester Ausgabe (siehe Abschnitt 9.10 [Ableitungen], Seite 167).

Zu den häufig benutzten Methoden gehören `url-fetch`, das Daten von einer URL lädt, und `git-fetch`, das Daten aus einem Git-Repository lädt (siehe unten).

`sha256` Ein Byte-Vektor mit dem SHA-256-Hash des Quellcodes. Seine Funktion ist dieselbe wie das Angeben eines `content-hash`-SHA256-Objekts im weiter unten beschriebenen `hash`-Feld.

`hash` Das `content-hash`-Objekt des Quellcodes. Siehe unten für eine Erklärung, wie Sie `content-hash` benutzen.

Diese Informationen liefert Ihnen der Befehl `guix download` (siehe Abschnitt 10.3 [Aufruf von guix download], Seite 204) oder `guix hash` (siehe Abschnitt 10.4 [Aufruf von guix hash], Seite 205).

`file-name` (Vorgabe: `#f`)
Der Dateiname, unter dem der Quellcode abgespeichert werden soll. Wenn er auf `#f` steht, wird ein vernünftiger Name automatisch gewählt. Falls der Quellcode von einer URL geladen wird, wird der Dateiname aus der URL

genommen. Wenn der Quellcode von einem Versionskontrollsystem bezogen wird, empfiehlt es sich, den Dateinamen ausdrücklich anzugeben, weil dann keine sprechende Benennung automatisch gefunden werden kann.

`patches` (Vorgabe: `'()`)
: Eine Liste von Dateinamen, Ursprüngen oder dateiähnlichen Objekten (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) mit Patches, welche auf den Quellcode anzuwenden sind.

  Die Liste von Patches kann nicht von Parametern der Erstellung abhängen. Insbesondere kann sie nicht vom Wert von `%current-system` oder `%current-target-system` abhängen.

`snippet` (Vorgabe: `#f`)
: Ein im Quellcode-Verzeichnis auszuführender G-Ausdruck (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) oder S-Ausdruck. Hiermit kann der Quellcode bequem modifiziert werden, manchmal ist dies bequemer als mit einem Patch.

`patch-flags` (Vorgabe: `'("-p1")`)
: Eine Liste der Befehlszeilenoptionen, die dem `patch`-Befehl übergeben werden sollen.

`patch-inputs` (Vorgabe: `#f`)
: Eingabepakete oder -ableitungen für den Patch-Prozess. Bei `#f` werden die üblichen Patcheingaben wie GNU Patch bereitgestellt.

`modules` (Vorgabe: `'()`)
: Eine Liste von Guile-Modulen, die während des Patch-Prozesses und während der Ausführung des `snippet`-Felds geladen sein sollen.

`patch-guile` (Vorgabe: `#f`)
: Welches Guile-Paket für den Patch-Prozess benutzt werden soll. Bei `#f` wird ein vernünftiger Vorgabewert angenommen.

`content-hash` *Wert* [*Algorithmus*] [Datentyp]
: Erzeugt ein Inhaltshash-Objekt für den gegebenen *Algorithmus* und benutzt dabei den *Wert* als dessen Hashwert. Wenn kein *Algorithmus* angegeben wird, wird `sha256` angenommen.

  Als *Wert* kann ein Zeichenketten-Literal, was base32-dekodiert wird, oder ein Byte-Vektor angegeben werden.

  Folgende Formen sind äquivalent:

```
(content-hash "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj")
(content-hash "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj"
              sha256)
(content-hash (base32
               "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj"))
(content-hash (base64 "kkb+RPaP7uyMZmu4eXPVkM4BN8yhRd8BTHLslb6f/Rc=")
              sha256)
```

  Als interne Implementierung wird für `content-hash` derzeit ein Makro benutzt. Es überprüft, wenn möglich, zum Zeitpunkt der Makroumschreibung, ob die Angaben in

Ordnung sind, z.B. ob der *Wert* die richtige Größe für den angegebenen *Algorithmus* hat.

Wie wir oben gesehen haben, hängt es von der im `method`-Feld angegebenen Methode ab, wie die in einem Paketursprung verwiesenen Daten geladen werden. Das Modul `(guix download)` stellt die am häufigsten benutzte Methode zur Verfügung, nämlich `url-fetch`, die im Folgenden beschrieben wird.

`url-fetch` *URL Hash-Algo Hash* [*name*] [*#:executable? #f*] [Scheme-Prozedur]
Liefert eine Ableitung mit fester Ausgabe, die

Daten von der *URL* lädt (einer Zeichenkette oder Liste von Zeichenketten für alternativ mögliche URLs). Es wird erwartet, dass die Daten *Hash* als Prüfsumme haben, gemäß dem Algorithmus, der in *Hash-Algo* (einem Symbol) angegebenen wurde. Nach Vorgabe ergibt sich der Dateiname aus dem Basisnamen der URL; optional kann in *name* ein anderslautender Name festgelegt werden. Wenn *executable?* wahr ist, wird die heruntergeladene Datei als ausführbar markiert.

Wenn eine der URL mit `mirror://` beginnt, wird der „Host Part“ an deren Anfang als Name eines Spiegelserver-Schemas aufgefasst, wie es in `%mirror-file` steht.

Alternativ wird, wenn die URL mit `file://` beginnt, der zugehörige Dateiname in den Store eingefügt und zurückgeliefert.

Ebenso ist im Modul `(guix git-download)` die `git-fetch`-Methode für Paketursprünge definiert. Sie lädt Daten aus einem Repository des Versionskontrollsystems Git. Der Datentyp `git-reference` beschreibt dabei das Repository und den daraus zu ladenden Commit.

`git-fetch` *Ref Hash-Algo Hash* [Scheme-Prozedur]
Liefert eine Ableitung mit fester Ausgabe, die *Ref* lädt, ein `<git-reference>`-Objekt. Es wird erwartet, dass die Ausgabe rekursiv die Prüfsumme *Hash* aufweist (ein „rekursiver Hash“) gemäß dem Typ *Hash-Algo* (einem Symbol). Als Dateiname wird *name* verwendet, oder ein generischer Name, falls *name* `#f` ist.

`git-reference` [Datentyp]
Dieser Datentyp steht für eine Git-Referenz, die `git-fetch` laden soll.

`url` Die URL des zu klonenden Git-Repositorys.

`commit` Diese Zeichenkette gibt entweder den zu ladenden Commit an (als Zeichenkette aus Hexadezimalzeichen) oder sie entspricht dem zu ladenden Tag. Sie können auch eine „kurze“ Commit-Zeichenkette oder einen Bezeichner wie von `git describe`, z.B. `v1.0.1-10-g58d7909c97`, verwenden.

`recursive?` (Vorgabe: `#f`)
Dieser boolesche Wert gibt an, ob Git-Submodule rekursiv geladen werden sollen.

Im folgenden Beispiel wird der Tag `v2.10` des Repositorys für GNU Hello bezeichnet:

```
(git-reference
  (url "https://git.savannah.gnu.org/git/hello.git")
  (commit "v2.10"))
```

Das ist äquivalent zu folgender Referenz, wo der Commit ausdrücklich benannt wird:

```
(git-reference
  (url "https://git.savannah.gnu.org/git/hello.git")
  (commit "dc7dc56a00e48fe6f231a58f6537139fe2908fb9"))
```

Bei Mercurial-Repositorys finden Sie im Modul `(guix hg-download)` Definitionen für die Methode `hg-fetch` für Paketursprünge sowie den Datentyp `hg-reference`. Mit ihnen wird das Versionskontrollsystem Mercurial unterstützt.

`hg-fetch` *`Ref Hash-Algo Hash`* [*Name*] *Liefert eine Ableitung mit fester Ausgabe, die* `Ref` *lädt, ein* [Scheme-Prozedur]

`<hg-reference>`-Objekt. Es wird erwartet, dass die Ausgabe rekursiv die Prüfsumme *Hash* aufweist (ein „rekursiver Hash“) gemäß dem Typ *Hash-Algo* (einem Symbol). Als Dateiname wird *Name* verwendet, oder ein generischer Name, falls *Name* `#false` ist.

9.3 Paketvarianten definieren

Eine der schönen Sachen an Guix ist, dass Sie aus einer Paketdefinition leicht Varianten desselben Pakets *ableiten* können – solche, die vom Anbieter eine andere Paketversion nehmen, als im Guix-Repository angegeben, solche mit anderen Abhängigkeiten, anders gearteten Compiler-Optionen und mehr. Manche dieser eigenen Pakete lassen sich direkt aus der Befehlszeile definieren (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192). Dieser Abschnitt beschreibt, wie man Paketvarianten mit Code definiert. Das kann in „Manifesten“ nützlich sein (siehe Abschnitt 9.4 [Manifeste verfassen], Seite 125) und in Ihrer eigenen Paketsammlung (siehe Abschnitt 7.6 [Einen Kanal erstellen], Seite 80), unter anderem!

Wie zuvor erörtert, sind Pakete Objekte erster Klasse in der Scheme-Sprache. Das Modul `(guix packages)` stellt das `package`-Konstrukt zur Verfügung, mit dem neue Paketobjekte definiert werden können (siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113). Am einfachsten ist es, eine Paketvariante zu definieren, indem Sie das `inherit`-Schlüsselwort mit einem `package`-Objekt verwenden. Dadurch können Sie die Felder einer Paketdefinition erben lassen, aber die Felder selbst festlegen, die Sie festlegen möchten.

Betrachten wir zum Beispiel die Variable `hello`, die eine Definition für die aktuelle Version von GNU Hello enthält. So können Sie eine Variante für Version 2.2 definieren (welche 2006 veröffentlicht wurde – ein guter Jahrgang!):

```
(use-modules (gnu packages base))    ;für ,,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"))))))
```

Das obige Beispiel entspricht dem, was Sie mit der Paketumwandlungsoption `--with-source` erreichen können. Im Kern erhält `hello-2.2` alle Felder von `hello` mit Ausnahme von `version` und `source`, die ersetzt werden (die beiden unterliegen einem „Override“). Beachten Sie, dass es die ursprüngliche `hello`-Variable weiterhin gibt, sie bleibt unverändert in dem Modul `(gnu packages base)`. Wenn Sie auf diese Weise ein eigenes Paket definieren, fügen Sie tatsächlich eine neue Paketdefinition hinzu; das Original bleibt erhalten.

Genauso gut können Sie Varianten mit einer anderen Menge von Abhängigkeiten als im ursprünglichen Paket definieren. Zum Beispiel hängt das vorgegebene `gdb`-Paket von `guile` ab, aber weil es eine optionale Abhängigkeit ist, können Sie eine Variante definieren, die jene Abhängigkeit entfernt, etwa so:

```
(use-modules (gnu packages gdb))   ;für „gdb“

(define gdb-sans-guile
  (package
    (inherit gdb)
    (inputs (modify-inputs (package-inputs gdb)
              (delete "guile")))))
```

Mit obiger `modify-inputs`-Form wird das `"guile"`-Paket aus den Eingaben im `inputs`-Feld von `gdb` entfernt. Das `modify-inputs`-Makro hilft Ihnen, wann immer Sie etwas aus den Paketeingaben entfernen, hinzufügen oder ersetzen möchten.

`modify-inputs` *Eingaben Klauseln* [Scheme-Syntax]

Ändert die übergebenen Paketeingaben, die `package-inputs` & Co. liefern können, entsprechend der angegebenen Klauseln. Jede Klausel muss eine der folgenden Formen aufweisen:

`(delete` *Name*`...)`
: Die Pakete mit den angegebenen *Name*n (als Zeichenketten) aus den Eingaben entfernen.

`(prepend` *Paket*`...)`
: Jedes *Paket* vorne an die Eingabenliste anstellen.

`(append` *Paket*`...)`
: Jedes *Paket* am Ende der Eingabenliste anhängen.

Mit folgendem Beispiel werden die Eingaben GMP und ACL unter denen von Coreutils weggelassen und libcap wird an dem Anfang hinzugefügt:

```
(modify-inputs (package-inputs coreutils)
  (delete "gmp" "acl")
  (prepend libcap))
```

Mit folgendem Beispiel wird das `guile`-Paket unter den Eingaben von `guile-redis` weggelassen und stattdessen wird `guile-2.2` verwendet:

```
(modify-inputs (package-inputs guile-redis)
  (replace "guile" guile-2.2))
```

Die letzte Art Klausel ist `append`, was bedeutet, dass Eingaben hinten an die Liste angehängt werden.

Manchmal bietet es sich an, Funktionen (also „Prozeduren“, wie Scheme-Anwender sagen) zu schreiben, die ein Paket abhängig von bestimmten Parametern zurückliefern. Als Beispiel betrachten wir die `luasocket`-Bibliothek für die Programmiersprache Lua. Wir möchten `luasocket`-Pakete für die hauptsächlichen Versionen von Lua verfügbar machen. Eine Möglichkeit, das zu erreichen, ist, eine Prozedur zu definieren, die ein Lua-Paket nimmt und ein von diesem abhängiges `luasocket`-Paket liefert.

```
(define (make-lua-socket name lua)
  ;; Liefert ein luasocket-Paket, das mit LUA erstellt wird.
  (package
    (name name)
    (version "3.0")
    ;; hier würden noch ein paar Felder stehen
    (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))
```

Damit haben wir Pakete `lua5.1-socket` und `lua5.2-socket` definiert, indem wir `make-lua-socket` mit verschiedenen Argumenten aufgerufen haben. Siehe Abschnitt “Procedures” in *Referenzhandbuch von GNU Guile* für mehr Informationen über Prozeduren. Weil wir mit `define-public` öffentlich sichtbare Definitionen auf oberster Ebene („top-level“) für diese beiden Pakete angegeben haben, kann man sie von der Befehlszeile aus benutzen (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

Bei diesen handelt es sich um sehr einfache Paketvarianten. Bequemer ist es dann, mit dem Modul `(guix transformations)` eine hochsprachliche Schnittstelle einzusetzen, die auf die komplexeren Paketumwandlungsoptionen direkt abbildet (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192):

`options->transformation` *`Optionen`* [Scheme-Prozedur]

Liefert eine Prozedur, die gegeben ein zu erstellendes Objekt (ein Paket, eine Ableitung oder Ähnliches) die durch *Optionen* festgelegten Umwandlungen daran umsetzt und die sich ergebenden Objekte zurückliefert. *Optionen* muss eine Liste von Paaren aus Symbol und Zeichenkette sein wie:

```
((with-branch . "guile-gcrypt=master")
 (without-tests . "libgcrypt"))
```

Jedes Symbol benennt eine Umwandlung. Die entsprechende Zeichenkette ist ein Argument an diese Umwandlung.

Zum Beispiel wäre ein gleichwertiges Manifest zu diesem Befehl:

```
guix build guix \
  --with-branch=guile-gcrypt=master \
  --with-debug-info=zlib
```

. . . dieses hier:

```
(use-modules (guix transformations))

(define transform
  ;; Die Prozedur zur Paketumwandlung.
  (options->transformation
   '((with-branch . "guile-gcrypt=master")
     (with-debug-info . "zlib"))))

(packages->manifest
 (list (transform (specification->package "guix"))))
```

Die Prozedur `options->transformation` lässt sich einfach benutzen, ist aber vielleicht nicht so flexibel, wie Sie es sich wünschen. Wie sieht ihre Implementierung aus? Aufmerksamen Lesern mag aufgefallen sein, dass die meisten Paketumwandlungen die oberflächlichen Änderungen aus den ersten Beispielen in diesem Abschnitt übersteigen: Sie *schreiben Eingaben um*, was im Abhängigkeitsgraphen bestimmte Eingaben durch andere ersetzt.

Das Umschreiben des Abhängigkeitsgraphen, damit Pakete im Graphen ausgetauscht werden, ist in der Prozedur `package-input-rewriting` aus `(guix packages)` implementiert.

`package-input-rewriting` *Ersetzungen* [Scheme-Prozedur]
[*umgeschriebener-Name*] [*#:deep? #t*] *Eine Prozedur liefern, die für ein*
ihr übergebenes Paket dessen direkte und indirekte Abhängigkeit gemäß den *Ersetzungen* umschreibt, einschließlich ihrer impliziten Eingaben, wenn *deep?* wahr ist. *Ersetzungen* ist eine Liste von Paketpaaren; das erste Element eines Paares ist das zu ersetzende Paket und das zweite ist, wodurch es ersetzt werden soll.

Optional kann als *umgeschriebener-Name* eine ein Argument nehmende Prozedur angegeben werden, die einen Paketnamen nimmt und den Namen nach dem Umschreiben zurückliefert.

Betrachten Sie dieses Beispiel:

```
(define libressl-statt-openssl
  ;; Dies ist eine Prozedur, mit der OPENSSL durch LIBRESSL
  ;; rekursiv ersetzt wird.
  (package-input-rewriting `((,openssl . ,libressl))))

(define git-mit-libressl
  (libressl-statt-openssl git))
```

Hier definieren wir zuerst eine Umschreibeprozedur, die *openssl* durch *libressl* ersetzt. Dann definieren wir damit eine *Variante* des *git*-Pakets, die *libressl* statt *openssl* benutzt. Das ist genau, was auch die Befehlszeilenoption `--with-input` tut (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192).

Die folgende Variante von `package-input-rewriting` kann für die Ersetzung passende Pakete anhand ihres Namens finden, statt zu prüfen, ob der Wert identisch ist.

`package-input-rewriting/spec` *Ersetzungen* [*#:deep? #t*] [Scheme-Prozedur]
: Liefert eine Prozedur, die für ein gegebenes Paket die angegebenen *Ersetzungen* auf dessen gesamten Paketgraphen anwendet (einschließlich impliziter Eingaben, außer wenn *deep?* falsch ist). *Ersetzungen* muss dabei eine Liste von Paaren aus je einer Spezifikation und Prozedur sein. Dabei ist jede Spezifikation eine Paketspezifikation wie `"gcc"` oder `"guile@2"` und jede Prozedur nimmt ein passendes Paket und liefert dafür einen Ersatz für das Paket.

Das obige Beispiel könnte auch so geschrieben werden:

```
(define libressl-statt-openssl
  ;; Rekursiv alle Pakete namens "openssl" durch LibreSSL ersetzen.
  (package-input-rewriting/spec `(("openssl" . ,(const libressl)))))
```

Der Hauptunterschied ist hier, dass diesmal Pakete zur Spezifikation passen müssen und nicht deren Wert identisch sein muss, damit sie ersetzt werden. Mit anderen Worten wird jedes Paket im Graphen ersetzt, das `openssl` heißt.

Eine allgemeiner anwendbare Prozedur, um den Abhängigkeitsgraphen eines Pakets umzuschreiben, ist `package-mapping`. Sie unterstützt beliebige Änderungen an den Knoten des Graphen.

`package-mapping` *Prozedur* [*Schnitt?*] [*#:deep? #f*] [Scheme-Prozedur]
: Liefert eine Prozedur, die, wenn ihr ein Paket übergeben wird, die an `package-mapping` übergebene *Prozedur* auf alle vom Paket abhängigen Pakete anwendet. Die Prozedur liefert das resultierende Paket. Wenn *Schnitt?* für ein Paket davon einen wahren Wert liefert, findet kein rekursiver Abstieg in dessen Abhängigkeiten statt. Steht *deep?* auf wahr, wird die *Prozedur* auch auf implizite Eingaben angewandt.

9.4 Manifeste verfassen

Sie können die Paketlisten für `guix`-Befehle auf der Befehlszeile angeben. Das ist bequem, bis die Paketlisten länger und weniger trivial werden. Dann nämlich wird es bald bequemer, die Paketliste in einer Form zu haben, die wir ein *Manifest* nennen. Neudeutsch kann man ein Manifest wie eine „Bill of Materials“, eine Stückliste oder Güterliste auffassen, womit ein Satz von Paketen festgelegt wird. Im Normalfall denken Sie sich ein Code-Schnipsel aus, mit dem das Manifest erstellt wird, bringen den Code in einer Datei unter, sagen wir `manifest.scm`, und übergeben diese Datei mit der Befehlszeilenoption `-m` (oder `--manifest`), die von vielen `guix`-Befehlen unterstützt wird. Zum Beispiel könnte so ein Manifest für einen einfachen Satz Pakete aussehen:

```
;; Manifest dreier Pakete.
(specifications->manifest '("gcc-toolchain" "make" "git"))
```

Sobald Sie das Manifest haben, können Sie es an z.B. `guix package` übergeben, was dann nur genau diese drei Pakete in Ihr Profil installiert (siehe [profile-manifest], Seite 50):

```
guix package -m manifest.scm
```

. . . oder Sie übergeben das Manifest an `guix shell` (siehe [shell-manifest], Seite 89) und richten sich so eine vergängliche Umgebung ein:

```
guix shell -m manifest.scm
```

. . . oder Sie übergeben das Manifest an `guix pack`, was ziemlich genauso geht (siehe [pack-manifest], Seite 104). Ein Manifest können Sie unter Versionskontrolle stellen oder es mit anderen Teilen, um deren System schnell auf den gleichen Stand zu bringen, und vieles mehr.

Doch wie schreibt man eigentlich sein erstes Manifest? Für den Anfang möchten Sie vielleicht ein Manifest, das dem nachempfunden ist, was Sie schon in einem Ihrer Profile haben. Statt bei null anzufangen, können Sie sich mit `guix package` ein Manifest generieren lassen (siehe [export-manifest], Seite 55):

```
# Wir schreiben in 'manifest.scm' ein Manifest, das dem
# Standardprofil ~/.guix-profile entspricht.
guix package --export-manifest > manifest.scm
```

Oder vielleicht möchten Sie die Argumente von der Befehlszeile in ein Manifest übertragen. Dabei kann `guix shell` helfen (siehe [shell-export-manifest], Seite 89):

```
# Wir schreiben ein Manifest für die auf der Befehlszeile
# angegebenen Pakete.
guix shell --export-manifest gcc-toolchain make git > manifest.scm
```

In beiden Fällen werden bei der Befehlszeilenoption `--export-manifest` etliche Feinheiten berücksichtigt, um ein originalgetreues Manifest zu erzeugen. Insbesondere werden Paketumwandlungsoptionen wiedergegeben (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192).

> **Anmerkung:** Manifeste sind *symbolisch*: Sie beziehen sich auf die Pakete, die in den *aktuell verwendeten* Kanälen enthalten sind (siehe Kapitel 7 [Kanäle], Seite 77). Im obigen Beispiel bezieht sich `gcc-toolchain` heute vielleicht auf Version 11, aber in zwei Jahren kann es Version 13 heißen.
>
> Wenn Sie wollen, dass immer dieselben Paketversionen und -varianten in Ihre Software-Umgebung aufgenommen werden, sind mehr Informationen nötig, nämlich welche Kanalversionen zur Zeit in Benutzung sind. Das sagt uns `guix describe`. Siehe Abschnitt 7.3 [Guix nachbilden], Seite 78, für weitere Informationen.

Sobald Sie sich Ihr erstes Manifest beschafft haben, möchten Sie vielleicht Anpassungen daran vornehmen. Da es sich beim Manifest um Code handelt, stehen Ihnen alle Guix-Programmierschnittstellen zur Verfügung!

Nehmen wir an, Sie hätten gerne ein Manifest, um eine angepasste Variante von GDB, dem GNU-Debugger, einzusetzen, die von Guile *nicht* abhängt, zusammen mit noch einem anderen Paket. Um auf dem Beispiel aus dem vorigen Abschnitt aufzubauen (siehe Abschnitt 9.3 [Paketvarianten definieren], Seite 121), können Sie das Manifest z.B. folgendermaßen schreiben:

```
(use-modules (guix packages)
             (gnu packages gdb)                ;für 'gdb'
             (gnu packages version-control))   ;für 'git'

;; Eine Variante von GDB ohne Abhängigkeit von Guile definieren.
(define gdb-sans-guile
  (package
    (inherit gdb)
```

```
    (inputs (modify-inputs (package-inputs gdb)
              (delete "guile")))))

;; Liefert ein Manifest mit diesem Paket und außerdem Git.
(packages->manifest (list gdb-sans-guile git))
```

Beachten Sie, in diesem Beispiel nimmt das Manifest direkt Bezug auf die Variablen `gdb` und `git`, die an je ein Paketobjekt vom Typ `package` gebunden sind (siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113), statt wie zuvor `specifications->manifest` aufzurufen und damit die Pakete anhand deren Namens zu finden. Die `use-modules`-Form am Dateianfang gibt uns Zugriff auf den Kern der Paketschnittstelle (siehe Abschnitt 9.2 [Pakete definieren], Seite 109) und die Module, in denen `gdb` und `git` definiert sind (siehe Abschnitt 9.1 [Paketmodule], Seite 108). Nahtlos können wir all dies miteinander verknüpfen – grenzenlose Möglichkeiten eröffnen sich; lassen Sie Ihrer Kreativität freien Lauf!

Der Datentyp für Manifeste sowie Prozeduren, um mit ihm umzugehen, sind definiert im Modul `(guix profiles)`. In Code, den Sie mit `-m` übergeben, wird es automatisch verfügbar gemacht. Nun folgt die Referenz dazu.

`manifest` [Datentyp]

Der Datentyp, der ein Manifest repräsentiert.

Zurzeit gibt es darin ein Feld:

`entries` Dies muss eine Liste von `manifest-entry`-Verbundsobjekten sein, wie hier beschrieben.

`manifest-entry` [Datentyp]

Der Datentyp steht für einen Eintrag im Manifest. Zu so einem Manifesteintrag gehören im Wesentlichen Metadaten: der Name und die Version als Zeichenkette, das Objekt selbst (meistens ein Paket), welche Ausgabe man davon haben will (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60) und noch ein paar optionale Informationen, wie wir im Folgenden näher ausführen.

Die meiste Zeit basteln Sie sich die Manifesteinträge nicht selber zusammen, sondern Sie übergeben ein Paket an `package->manifest-entry`, siehe unten. Aber es könnte Ihnen ein außergewöhnlicher Fall unterkommen, wo Sie einen Manifesteintrag für etwas erzeugen wollen, das *kein* Paket ist, wie in diesem Beispiel:

```
;; Für ein Nicht-Paketobjekt einen einzelnen Manifesteintrag von Hand schreiben.
(let ((hello (program-file "hello" #~(display "Hi!"))))
  (manifest-entry
    (name "foo")
    (version "42")
    (item
     (computed-file "verzeichnis-mit-hello"
                    #~(let ((bin (string-append #$output "/bin")))
                        (mkdir #$output) (mkdir bin)
                        (symlink #$hello
                                 (string-append bin "/hello")))))))
```

Diese Felder stehen zur Verfügung:

`name`
`version` Name und Version für diesen Eintrag als Zeichenkette.

`item` Ein Paket oder anderes dateiartiges Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

`output` (Vorgabe: `"out"`)
Welche der Ausgaben von `item` genommen werden soll, sofern `item` mehrere Ausgaben umfasst (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60).

`dependencies` (Vorgabe: `'()`)
Die Liste der Manifesteinträge, von denen dieser Eintrag abhängt. Wenn ein Profil erstellt wird, werden die aufgeführten Abhängigkeiten zum Profil hinzugefügt.

In der Regel landen die propagierten Eingaben eines Pakets (siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113) in je einem Manifesteintrag unter den Abhängigkeiten des Manifesteintrags des Pakets.

`search-paths` (Vorgabe: `'()`)
Die Liste der Suchpfadspezifikationen, die für diesen Eintrag beachtet werden (siehe Abschnitt 9.8 [Suchpfade], Seite 161).

`properties` (Vorgabe: `'()`)
Eine Liste von Paaren aus jeweils einem Symbol und dem Wert dazu. Beim Erstellen eines Profils werden die Eigenschaften serialisiert.

So kann man den Paketen zusätzliche Metadaten aufschnallen wie z.B. welche Umwandlungsoptionen darauf angewendet wurden (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192).

`parent` (Vorgabe: `(delay #f)`)
Ein Versprechen (unter englischsprechenden Schemern bekannt als „Promise“), das auf den übergeordneten Manifesteintrag zeigt.

Es wird benutzt, um in auf Manifesteinträge in `dependencies` bezogenen Fehlermeldungen Hinweise auf den Kontext zu geben.

`concatenate-manifests` *Liste* [Scheme-Prozedur]
Fasst die Manifeste in der *Liste* zu einem zusammen und liefert es zurück.

`package->manifest-entry` *Paket* [*Ausgabe*] [*#:properties*] [Scheme-Prozedur]
Liefert einen Manifesteintrag für die *Ausgabe* von *Paket*, wobei *Ausgabe* als Vorgabewert `"out"` hat. Als Eigenschaften werden die *properties* benutzt; vorgegeben ist die leere Liste oder, wenn eine oder mehrere Paketumwandlungen auf das *Paket* angewendet wurden, eine assoziative Liste mit diesen Umwandlungen, die als Argument an `options->transformation` gegeben werden kann (siehe Abschnitt 9.3 [Paketvarianten definieren], Seite 121).

Mit folgendem Code-Schnipsel wird ein Manifest mit einem Eintrag für die Standardausgabe sowie die Ausgabe namens `send-email` des Pakets `git` erstellt:

```
(use-modules (gnu packages version-control))
```

```
(manifest (list (package->manifest-entry git)
                (package->manifest-entry git "send-email")))
```

`packages->manifest` *Pakete* [Scheme-Prozedur]

Liefert eine Liste von Manifesteinträgen, jeweils einer für jedes Listenelement in *Pakete*. In *Pakete* können Paketobjekte oder Tupel aus Paket und Zeichenkette stehen, wobei die Zeichenkette die Paketausgabe angibt.

Mithilfe dieser Prozedur kann man das obige Manifest auch kürzer aufschreiben:

```
(use-modules (gnu packages version-control))

(packages->manifest (list git `(,git "send-email")))
```

`package->development-manifest` *Paket* [*System*] [*#:target*] [Scheme-Prozedur]

Liefert ein Manifest mit den Entwicklungseingaben des *Pakets* für *System*. Optional kann mit *target* das Zielsystem zum Cross-Kompilieren angegeben werden. Zu den Entwicklungseingaben gehören die expliziten und impliziten Eingaben vom *Paket*.

Genau wie bei der Option `-D` für `guix shell` (siehe [shell-development-option], Seite 88) beschreibt das sich daraus ergebende Manifest die Umgebung, um am *Paket* als Entwickler mitzuarbeiten. Wenn Sie zum Beispiel eine Entwicklungsumgebung für Inkscape aufsetzen wollen und darin auch Git für die Versionskontrolle zur Verfügung haben möchten, geben Sie diese Bestandteilliste mit folgendem Manifest wieder:

```
(use-modules (gnu packages inkscape)          ;für 'inkscape'
             (gnu packages version-control))  ;für 'git'

(concatenate-manifests
 (list (package->development-manifest inkscape)
       (packages->manifest (list git))))
```

Dieses Beispiel zeigt, wie `package->development-manifest` ein Entwicklungsmanifest mit einem Compiler (GCC), den vielen benutzten Bibliotheken (Boost, GLib, GTK, etc.) und noch ein paar zusätzlichen Entwicklungswerkzeugen liefert – das sagen uns die von `guix show inkscape` aufgeführten Abhängigkeiten.

Zum Schluss sind im Modul `(gnu packages)` noch Abstraktionen enthalten, um Manifeste anzufertigen. Dazu gehört, Pakete anhand ihres Namens zu finden – siehe unten.

`specifications->manifest` *Spezifikationen* [Scheme-Prozedur]

Liefert ein Manifest für die *Spezifikationen*, einer Liste von Spezifikationen wie `"emacs@25.2"` oder `"guile:debug"`. Das Format für die Spezifikationen ist dasselbe wie bei den Befehlszeilenwerkzeugen `guix install`, `guix package` und so weiter (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

Ein Beispiel wäre diese Möglichkeit, das zuvor gezeigte Git-Manifest anders zu formulieren wie hier:

```
(specifications->manifest '("git" "git:send-email"))
```

Man bemerke, dass wir uns nicht um `use-modules` kümmern müssen, um die richtige Auswahl von Modulen zu importieren und die richtigen Variablen zu referenzieren.

Stattdessen nehmen wir auf Pakete direkt auf die Weise Bezug, die wir von der Befehlszeile kennen. Wie praktisch!

9.5 Erstellungssysteme

Jede Paketdefinition legt ein *Erstellungssystem* („build system“) sowie dessen Argumente fest (siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Das `build-system`-Feld steht für die Erstellungsprozedur des Pakets sowie für weitere implizite Eingaben für die Erstellungsprozedur.

Erstellungssysteme sind `<build-system>`-Objekte. Die Schnittstelle, um solche zu erzeugen und zu verändern, ist im Modul `(guix build-system)` zu finden, und die eigentlichen Erstellungssysteme werden jeweils von ihren eigenen Modulen exportiert.

Intern funktionieren Erstellungssysteme, indem erst Paketobjekte zu *Bags* kompiliert werden. Eine Bag (deutsch: Beutel, Sack) ist wie ein Paket, aber mit weniger Zierrat – anders gesagt ist eine Bag eine systemnähere Darstellung eines Pakets, die sämtliche Eingaben des Pakets einschließlich vom Erstellungssystem hinzugefügter Eingaben enthält. Diese Zwischendarstellung wird dann zur eigentlichen Ableitung kompiliert (siehe Abschnitt 9.10 [Ableitungen], Seite 167). Die Prozedur `package-with-c-toolchain` ist zum Beispiel eine Möglichkeit, wie die durch das Erstellungssystem hinzugenommenen impliziten Eingaben abgeändert werden können (siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113).

Erstellungssysteme akzeptieren optional eine Liste von *Argumenten*. In Paketdefinitionen werden diese über das `arguments`-Feld übergeben (siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Sie sind in der Regel Schlüsselwort-Argumente (siehe Abschnitt “Optional Arguments” in *Referenzhandbuch zu GNU Guile*). Der Wert dieser Argumente wird normalerweise vom Erstellungssystem in der *Erstellungsschicht* ausgewertet, d.h. von einem durch den Daemon gestarteten Guile-Prozess (siehe Abschnitt 9.10 [Ableitungen], Seite 167).

Das häufigste Erstellungssystem ist `gnu-build-system`, was die übliche Erstellungsprozedur für GNU-Pakete und viele andere Pakete darstellt. Es wird vom Modul `(guix build-system gnu)` bereitgestellt.

`gnu-build-system` [Scheme-Variable]

`gnu-build-system` steht für das GNU-Erstellungssystem und Varianten desselben (siehe Abschnitt “Configuration” in *GNU Coding Standards*).

Kurz gefasst werden Pakete, die es benutzen, konfiguriert, erstellt und installiert mit der üblichen Befehlsfolge `./configure && make && make check && make install`. In der Praxis braucht man oft noch ein paar weitere Schritte. Alle Schritte sind in voneinander getrennte *Phasen* unterteilt. Siehe Abschnitt 9.6 [Erstellungsphasen], Seite 150, für mehr Informationen zu Erstellungsphasen und wie man sie anpassen kann.

Zusätzlich stellt dieses Erstellungssystem sicher, dass die „Standard“-Umgebung für GNU-Pakete zur Verfügung steht. Diese umfasst Werkzeuge wie GCC, libc, Coreutils, Bash, Make, Diffutils, grep und sed (siehe das Modul `(guix build-system gnu)` für eine vollständige Liste). Wir bezeichnen sie als *implizite Eingaben* eines Pakets, weil Paketdefinitionen sie nicht aufführen müssen.

Dieses Erstellungssystem unterstützt eine Reihe von Schlüsselwortargumenten, die über das `arguments`-Feld eines Pakets übergeben werden können. Hier sind einige der wichtigen Parameter:

`#:phases` Mit diesem Argument wird erstellungsseitiger Code angegeben, der zu einer assoziativen Liste von Erstellungsphasen ausgewertet wird. Siehe Abschnitt 9.6 [Erstellungsphasen], Seite 150, für nähere Informationen.

`#:configure-flags`
: Diese Liste von Befehlszeilenoptionen (als Zeichenketten) werden dem `configure`-Skript übergeben. Siehe Abschnitt 9.2 [Pakete definieren], Seite 109, für ein Beispiel.

`#:make-flags`
: Diese Zeichenkettenliste enthält Befehlszeilenoptionen, die als Argumente an `make`-Aufrufe in den Phasen `build`, `check` und `install` übergeben werden.

`#:out-of-source?`
: Dieser Boolesche Ausdruck, nach Vorgabe steht er auf `#f`, zeigt an, ob Erstellungen in einem gesonderten Erstellungsverzeichnis abseits des Quellbaums ausgeführt werden sollen.

  Wenn es auf wahr steht, wird in der `configure`-Phase eigens ein Erstellungsverzeichnis angelegt, dorthin gewechselt und das `configure`-Skript von dort ausgeführt. Das ist nützlich bei Paketen, die so etwas voraussetzen, wie `glibc`.

`#:tests?` Dieser Boolesche Ausdruck, nach Vorgabe steht er auf `#t`, zeigt an, ob in der `check`-Phase der Testkatalog des Pakets ausgeführt werden soll.

`#:test-target`
: In dieser Zeichenkette, nach Vorgabe `"check"`, wird der Name des Makefile-Ziels angegeben, das die `check`-Phase benutzt.

`#:parallel-build?`
`#:parallel-tests?`
: Mit diesen Booleschen Werten wird festgelegt, ob die Erstellung respektive der Testkatalog parallel ausgeführt werden soll, indem die Befehlszeilenoption `-j` an `make` übergeben wird. Wenn die Werte wahr sind, wird an `make` die Option `-j`*n* übergeben, wobei *n* die Zahl ist, die in der `--cores`-Befehlszeilenoption an `guix-daemon` oder an den `guix`-Clientbefehl übergeben wurde (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

`#:validate-runpath?`
: Dieser Boolesche Ausdruck, nach Vorgabe `#t`, bestimmt, ob der in ELF-Binärdateien, die in der `install`-Phase installiert worden sind, eingetragene `RUNPATH` „validiert“ werden soll. ELF-Binärdateien sind gemeinsame Bibliotheken („Shared Libraries“ mit Dateiendung `.so`) sowie ausführbare Dateien. Siehe [phase-validate-runpath], Seite 151, für die Details.

`#:substitutable?`
: Dieser Boolesche Ausdruck, nach Vorgabe `#t`, sagt aus, ob Paketausgaben substituierbar sein sollen, d.h. ob Benutzer Substitute dafür beziehen können sollen, statt sie lokal erstellen zu müssen (siehe Abschnitt 6.3 [Substitute], Seite 56).

`#:allowed-references`
`#:disallowed-references`

Wenn für diese Argumente ein wahrer Wert angegeben wird, muss er einer Liste von Abhängigkeiten entsprechen, die *nicht* unter den Referenzen der Erstellungsergebnisse vorkommen dürfen. Wenn nach dem Ende der Erstellung eine solche Referenz noch vorhanden ist, schlägt der Erstellungsvorgang fehl.

Sie eignen sich, um zu garantieren, dass ein Paket nicht fälschlich seine Abhängigkeiten aus der Erstellungszeit weiter referenziert, wenn das, zum Beispiel, die Größe unnötig in die Höhe treiben würde.

Auch die meisten anderen Erstellungssysteme unterstützen diese Schlüsselwortargumente.

Andere `<build-system>`-Objekte werden definiert, um andere Konventionen und Werkzeuge von Paketen für freie Software zu unterstützen. Die anderen Erstellungssysteme erben den Großteil vom `gnu-build-system` und unterscheiden sich hauptsächlich darin, welche Eingaben dem Erstellungsprozess implizit hinzugefügt werden und welche Liste von Phasen durchlaufen wird. Manche dieser Erstellungssysteme sind im Folgenden aufgeführt.

`ant-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system ant)` exportiert. Sie implementiert die Erstellungsprozedur für Java-Pakete, die mit dem Ant build tool (`https://ant.apache.org/`) erstellt werden können.

Sowohl `ant` als auch der *Java Development Kit* (JDK), wie er vom Paket `icedtea` bereitgestellt wird, werden zu den Eingaben hinzugefügt. Wenn andere Pakete dafür benutzt werden sollen, können sie jeweils mit den Parametern `#:ant` und `#:jdk` festgelegt werden.

Falls das ursprüngliche Paket über keine nutzbare Ant-Erstellungsdatei („Ant-Buildfile“) verfügt, kann aus der Angabe im Parameter `#:jar-name` eine minimale Ant-Erstellungsdatei `build.xml` erzeugt werden, in der die für die Erstellung durchzuführenden Aufgaben (Tasks) für die Erstellung des angegebenen Jar-Archivs stehen. In diesem Fall kann der Parameter `#:source-dir` benutzt werden, um das Unterverzeichnis mit dem Quellcode anzugeben; sein Vorgabewert ist „src“.

Der Parameter `#:main-class` kann mit einer minimalen Ant-Erstellungsdatei benutzt werden, um die Hauptklasse des resultierenden Jar-Archivs anzugeben. Dies ist nötig, wenn die Jar-Datei ausführbar sein soll. Mit dem Parameter `#:test-include` kann eine Liste angegeben werden, welche Junit-Tests auszuführen sind. Der Vorgabewert ist `(list "**/*Test.java")`. Mit `#:test-exclude` kann ein Teil der Testdateien ignoriert werden. Der Vorgabewert ist `(list "**/Abstract*.java")`, weil abstrakte Klassen keine ausführbaren Tests enthalten können.

Der Parameter `#:build-target` kann benutzt werden, um die Ant-Aufgabe (Task) anzugeben, die während der `build`-Phase ausgeführt werden soll. Vorgabe ist, dass die Aufgabe (Task) „jar“ ausgeführt wird.

`android-ndk-build-system` [Scheme-Variable]

Diese Variable wird von `(guix build-system android-ndk)` exportiert. Sie implementiert eine Erstellungsprozedur für das Android NDK (Native Development Kit) benutzende Pakete mit einem Guix-spezifischen Erstellungsprozess.

Für das Erstellungssystem wird angenommen, dass Pakete die zu ihrer öffentlichen Schnittstelle gehörenden Header-Dateien im Unterverzeichnis `include` der Ausgabe `out` und ihre Bibliotheken im Unterverzeichnis `lib` der Ausgabe `out` platzieren.

Ebenso wird angenommen, dass es keine im Konflikt stehenden Dateien unter der Vereinigung aller Abhängigkeiten gibt.

Derzeit wird Cross-Kompilieren hierfür nicht unterstützt, also wird dabei vorausgesetzt, dass Bibliotheken und Header-Dateien dieselben wie im Wirtssystem sind.

`asdf-build-system/source` [Scheme-Variable]
`asdf-build-system/sbcl` [Scheme-Variable]
`asdf-build-system/ecl` [Scheme-Variable]

Diese Variablen, die vom Modul `(guix build-system asdf)` exportiert werden, implementieren Erstellungsprozeduren für Common-Lisp-Pakete, welche „ASDF“ (`https://common-lisp.net/project/asdf/`) benutzen. ASDF dient der Systemdefinition für Common-Lisp-Programme und -Bibliotheken.

Das Erstellungssystem `asdf-build-system/source` installiert die Pakete in Quellcode-Form und kann *via* ASDF mit jeder Common-Lisp-Implementierung geladen werden. Die anderen Erstellungssysteme wie `asdf-build-system/sbcl` installieren binäre Systeme in dem Format, das von einer bestimmten Implementierung verstanden wird. Diese Erstellungssysteme können auch benutzt werden, um ausführbare Programme zu erzeugen oder um Lisp-Abbilder mit einem vorab geladenen Satz von Paketen zu erzeugen.

Das Erstellungssystem benutzt gewisse Namenskonventionen. Bei Binärpaketen sollte dem Paketnamen die Lispimplementierung als Präfix vorangehen, z.B. `sbcl-` für `asdf-build-system/sbcl`.

Zudem sollte das entsprechende Quellcode-Paket mit der Konvention wie bei Python-Paketen (siehe Abschnitt 22.4.7 [Python-Module], Seite 719) ein `cl-` als Präfix bekommen.

Um ausführbare Programme und Abbilder zu erzeugen, können die erstellungsseitigen Prozeduren `build-program` und `build-image` benutzt werden. Sie sollten in einer Erstellungsphase nach der `create-asdf-configuration`-Phase aufgerufen werden, damit das gerade erstellte System Teil des resultierenden Abbilds sein kann. An `build-program` muss eine Liste von Common-Lisp-Ausdrücken über das Argument `#:entry-program` übergeben werden.

Vorgegeben ist, alle `.asd`-Dateien im Quellverzeichnis zu lesen, um zu ermitteln, welche Systeme definiert sind. Mit dem Parameter `#:asd-files` kann die Liste zu lesender `.asd`-Dateien festgelegt werden. Außerdem wird bei Paketen, für deren Tests ein System in einer separaten Datei definiert wurde, dieses System geladen, bevor die Tests ablaufen, wenn es im Parameter `#:test-asd-file` steht. Ist dafür kein Wert gesetzt, werden die Dateien `<system>-tests.asd`, `<system>-test.asd`, `tests.asd` und `test.asd` durchsucht, wenn sie existieren.

Wenn aus irgendeinem Grund der Paketname nicht den Namenskonventionen folgen kann oder wenn mehrere Systeme kompiliert werden, kann der Parameter `#:asd-systems` benutzt werden, um die Liste der Systemnamen anzugeben.

`cargo-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system cargo)` exportiert. Damit können Pakete mit Cargo erstellt werden, dem Erstellungswerkzeug der Rust-Programmiersprache (`https://www.rust-lang.org`).

Das Erstellungssystem fügt `rustc` und `cargo` zu den Eingaben hinzu. Ein anderes Rust-Paket kann mit dem Parameter `#:rust` angegeben werden.

Normale cargo-Abhängigkeiten sollten so wie bei anderen Paketen in die Paketdefinition eingetragen werden; wenn sie nur zur Erstellungszeit gebraucht werden, gehören sie in `native-inputs`, sonst in `inputs`. Wenn die Abhängigkeiten Crates sind, die nur als Quellcode vorliegen, sollten sie zusätzlich über den Parameter `#:cargo-inputs` als eine Liste von Paaren aus Name und Spezifikation hinzugefügt, wobei als Spezifikation ein Paket oder eine Quellcode-Definition angegeben werden kann. Beachten Sie, dass die Spezifikation zu einem mit gzip komprimierten Tarball ausgewertet werden muss, der eine Datei `Cargo.toml` in seinem Wurzelverzeichnis enthält, ansonsten wird sie ignoriert. Analog sollten solche Abhängigkeiten, die in cargo als „dev-dependencies“ deklariert werden, zur Paketdefinition über den Parameter `#:cargo-development-inputs` hinzugefügt werden.

In seiner `configure`-Phase sorgt dieses Erstellungssystem dafür, dass cargo alle Quellcodeeingaben zur Verfügung stehen, die in den Parametern `#:cargo-inputs` und `#:cargo-development-inputs` angegeben wurden. Außerdem wird eine enthaltene `Cargo.lock`-Datei entfernt, damit `cargo` selbige während der `build`-Phase neu erzeugt. Die `package`-Phase führt `cargo package` aus, um eine Quellcode-Crate zur späteren Nutzung zu erzeugen. Die `install`-Phase installiert die in der Crate definierten Binärdateien. Wenn *nicht* `install-source? #f` definiert ist, werden auch ein Verzeichnis mit dem eigenen Quellcode in einer Crate und auch der unverpackte Quellcode installiert, damit es leichter ist, später an Rust-Paketen zu hacken.

`chicken-build-system` [Scheme-Variable]

Diese Variable wird von `(guix build-system chicken)` exportiert. Mit ihr werden Module von CHICKEN Scheme (`https://call-cc.org/`) kompiliert. Sie sind auch bekannt als „Eggs“ oder als „Erweiterungen“. CHICKEN erzeugt C-Quellcode, der dann von einem C-Compiler kompiliert wird; in diesem Fall vom GCC.

Dieses Erstellungssystem fügt `chicken` neben den Paketen des `gnu-build-system` zu den Paketeingaben hinzu.

Das Erstellungssystem kann den Namen des Eggs (noch) nicht automatisch herausfinden, also müssen Sie, ähnlich wie beim `#:import-path` des `go-build-system`, im `arguments`-Feld des Pakets den `#:egg-name` festlegen.

Zum Beispiel würden Sie so vorgehen, um ein Paket für das Egg `srfi-1` zu schreiben:

```
(arguments '(#:egg-name "srfi-1"))
```

Abhängigkeiten von Eggs müssen in `propagated-inputs` genannt werden und *nicht* in `inputs`, weil CHICKEN keine absoluten Referenzen in kompilierte Eggs einbettet. Abhängigkeiten für Tests sollten wie üblich in `native-inputs` stehen.

`copy-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system copy)` exportiert. Damit können einfache Pakete erstellt werden, für die nur wenig kompiliert werden muss, sondern in erster Linie Dateien kopiert werden.

Dadurch wird ein Großteil der `gnu-build-system` zur Menge der Paketeingaben hinzugefügt. Deswegen kann man bei Nutzung des `copy-build-system` auf große Teile des Codes verzichten, der beim `trivial-build-system` anfallen würde.

Um den Dateiinstallationsvorgang weiter zu vereinfachen, wird ein Argument `#:install-plan` zur Verfügung gestellt, mit dem der Paketautor angeben kann, welche Dateien wohin gehören. Der Installationsplan ist eine Liste aus (*Quelle Ziel* [*Filter*]). Die *Filter* sind optional.

- Wenn die *Quelle* einer Datei oder einem Verzeichnis ohne Schrägstrich am Ende entspricht, wird sie nach *Ziel* installiert.
  - Hat das *Ziel* einen Schrägstrich am Ende, wird mit dem Basisnamen der *Quelle* innerhalb von *Ziel* installiert.
  - Andernfalls wird die *Quelle* als *Ziel* installiert.
- Falls es sich bei der *Quelle* um ein Verzeichnis mit Schrägstrich am Ende handelt oder wenn *Filter* benutzt werden, so ist der Schrägstrich am Ende von *Ziel* mit der Bedeutung wie oben implizit.
  - Ohne Angabe von *Filter*n wird der gesamte *Inhalt* der *Quelle* nach *Ziel* installiert.
  - Werden *Filter* als `#:include`, `#:include-regexp`, `#:exclude` oder `#:exclude-regexp` aufgeführt, werden je nach Filter nur die ausgewählten Dateien installiert. Jeder Filter wird als Liste von Zeichenketten angegeben.
    - Bei `#:include` werden all die Dateien installiert, deren Pfad als Suffix zu mindestens einem der Elemente der angegebenen Liste passt.
    - Bei `#:include-regexp` werden all die Dateien installiert, deren Unterpfad zu mindestens einem der regulären Ausdrücke in der angegebenen Liste passt.
    - Die Filter `#:exclude` und `#:exclude-regexp` bewirken das Gegenteil ihrer Include-Entsprechungen. Ohne `#:include`-Angaben werden alle Dateien außer den zu den Exclude-Filtern passenden installiert. Werden sowohl `#:include` als auch `#:exclude` angegeben, werden zuerst die `#:include`-Angaben beachtet und danach wird durch `#:exclude` gefiltert.

  In jedem Fall bleiben die Pfade relativ zur *Quelle* innerhalb des *Ziel*s erhalten.

Beispiele:

- `("foo/bar" "share/my-app/")`: Installiert `bar` nach `share/my-app/bar`.
- `("foo/bar" "share/my-app/baz")`: Installiert `bar` nach `share/my-app/baz`.
- `("foo/" "share/my-app")`: Installiert den Inhalt von `foo` innerhalb von `share/my-app`. Zum Beispiel wird `foo/sub/datei` nach `share/my-app/sub/datei` installiert.

- `("foo/" "share/my-app" #:include ("sub/datei"))`: Installiert nur `foo/sub/datei` nach `share/my-app/sub/datei`.
- `("foo/sub" "share/my-app" #:include ("datei"))`: Installiert `foo/sub/datei` nach `share/my-app/datei`.

`clojure-build-system` [Scheme-Variable]

Diese Variable wird durch das Modul `(guix build-system clojure)` exportiert. Sie implementiert eine einfache Erstellungsprozedur für in Clojure (`https://clojure.org/`) geschriebene Pakete mit dem guten alten `compile` in Clojure. Cross-Kompilieren wird noch nicht unterstützt.

Das Erstellungssystem fügt `clojure`, `icedtea` und `zip` zu den Eingaben hinzu. Sollen stattdessen andere Pakete benutzt werden, können diese jeweils mit den Parametern `#:clojure`, `#:jdk` und `#:zip` spezifiziert werden.

Eine Liste der Quellcode-Verzeichnisse, Test-Verzeichnisse und Namen der Jar-Dateien können jeweils über die Parameter `#:source-dirs`, `#:test-dirs` und `#:jar-names` angegeben werden. Das Verzeichnis, in das kompiliert wird, sowie die Hauptklasse können jeweils mit den Parametern `#:compile-dir` und `#:main-class` angegeben werden. Andere Parameter sind im Folgenden dokumentiert.

Dieses Erstellungssystem ist eine Erweiterung des `ant-build-system`, bei der aber die folgenden Phasen geändert wurden:

`build` Diese Phase ruft `compile` in Clojure auf, um Quelldateien zu kompilieren, und führt `jar` aus, um Jar-Dateien aus sowohl Quelldateien als auch kompilierten Dateien zu erzeugen, entsprechend der jeweils in `#:aot-include` und `#:aot-exclude` festgelegten Listen aus in der Menge der Quelldateien eingeschlossenen und ausgeschlossenen Bibliotheken. Die Ausschlussliste hat Vorrang vor der Einschlussliste. Diese Listen setzen sich aus Symbolen zusammen, die für Clojure-Bibliotheken stehen oder dem Schlüsselwort `#:all` entsprechen, was für alle im Quellverzeichis gefundenen Clojure-Bibliotheken steht. Der Parameter `#:omit-source?` entscheidet, ob Quelldateien in die Jar-Archive aufgenommen werden sollen.

`check` In dieser Phase werden Tests auf die durch Einschluss- und Ausschlussliste `#:test-include` bzw. `#:test-exclude` angegebenen Dateien ausgeführt. Deren Bedeutung ist analog zu `#:aot-include` und `#:aot-exclude`, außer dass das besondere Schlüsselwort `#:all` jetzt für alle Clojure-Bibliotheken in den Test-Verzeichnissen steht. Der Parameter `#:tests?` entscheidet, ob Tests ausgeführt werden sollen.

`install` In dieser Phase werden alle zuvor erstellten Jar-Dateien installiert.

Zusätzlich zu den bereits angegebenen enthält dieses Erstellungssystem noch eine weitere Phase.

`install-doc`
Diese Phase installiert alle Dateien auf oberster Ebene, deren Basisnamen ohne Verzeichnisangabe zu `%doc-regex` passen. Ein anderer regulärer Ausdruck kann mit dem Parameter `#:doc-regex` verwendet werden. All die so gefundenen oder (rekursiv) in den mit `#:doc-dirs` angegebenen Dokumentationsverzeichnissen liegenden Dateien werden installiert.

`cmake-build-system` [Scheme-Variable]

Diese Variable wird von `(guix build-system cmake)` exportiert. Sie implementiert die Erstellungsprozedur für Pakete, die das CMake-Erstellungswerkzeug (`https://www.cmake.org`) benutzen.

Das Erstellungssystem fügt automatisch das Paket `cmake` zu den Eingaben hinzu. Welches Paket benutzt wird, kann mit dem Parameter `#:cmake` geändert werden.

Der Parameter `#:configure-flags` wird als Liste von Befehlszeilenoptionen aufgefasst, die an den Befehl `cmake` übergeben werden. Der Parameter `#:build-type` abstrahiert, welche Befehlszeilenoptionen dem Compiler übergeben werden; der Vorgabewert ist `"RelWithDebInfo"` (kurz für „release mode with debugging information“), d.h. kompiliert wird für eine Produktionsumgebung und Informationen zur Fehlerbehebung liegen bei, was ungefähr `-O2 -g` entspricht, wie bei der Vorgabe für Autoconf-basierte Pakete.

`dune-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system dune)` exportiert. Sie unterstützt es, Pakete mit Dune (`https://dune.build/`) zu erstellen, einem Erstellungswerkzeug für die Programmiersprache OCaml, und ist als Erweiterung des unten beschriebenen OCaml-Erstellungssystems `ocaml-build-system` implementiert. Als solche können auch die Parameter `#:ocaml` und `#:findlib` an dieses Erstellungssystem übergeben werden.

Das Erstellungssystem fügt automatisch das Paket `dune` zu den Eingaben hinzu. Welches Paket benutzt wird, kann mit dem Parameter `#:dune` geändert werden.

Es gibt keine `configure`-Phase, weil dune-Pakete typischerweise nicht konfiguriert werden müssen. Vom Parameter `#:build-flags` wird erwartet, dass es sich um eine Liste von Befehlszeilenoptionen handelt, die zur Erstellung an den `dune`-Befehl übergeben werden.

Der Parameter `#:jbuild?` kann übergeben werden, um den Befehl `jbuild` anstelle des neueren `dune`-Befehls aufzurufen, um das Paket zu erstellen. Der Vorgabewert ist `#f`.

Mit dem Parameter `#:package` kann ein Paketname angegeben werden, wenn im Paket mehrere Pakete enthalten sind und nur eines davon erstellt werden soll. Es ist äquivalent dazu, die Befehlszeilenoption `-p` an `dune` zu übergeben.

`elm-build-system` [Scheme-Variable]

Diese Variable wird von `(guix build-system elm)` exportiert. Sie implementiert eine Erstellungsprozedur für Elm-Pakete (`https://elm-lang.org`) ähnlich wie ‘`elm install`’.

Mit dem Erstellungssystem wird ein Elm-Compiler-Paket zu der Menge der Eingaben hinzugefügt. Anstelle des vorgegebenen Compiler-Pakets (derzeit ist es `elm-sans-reactor`) kann das stattdessen zu verwendende Compiler-Paket im Argument `#:elm` angegeben werden. Zudem werden Elm-Pakete, die vom Erstellungssystem selbst vorausgesetzt werden, als implizite Eingaben hinzugefügt, wenn sie noch keine sind; wenn Sie das verhindern möchten, übergeben Sie das Argument `#:implicit-elm-package-inputs?`, was in erster Linie zum Bootstrapping gebraucht wird.

Die Einträge für `"dependencies"` und `"test-dependencies"` in der Datei `elm.json` eines Elm-Pakets werden jeweils mit `propagated-inputs` bzw. `inputs` wiedergegeben.

In Elm wird von Paketnamen eine bestimmte Struktur verlangt. Siehe Abschnitt 22.4.11 [Elm-Pakete], Seite 721, für mehr Details auch zu den Werkzeugen in `(guix build-system elm)`, um damit umzugehen.

Derzeit gelten für `elm-build-system` ein paar nennenswerte Einschränkungen:

- Der Fokus für das Erstellungssystem liegt auf dem, was in Elm *Pakete* genannt wird, also auf *Projekten in Elm*, für die `{ "type": "package" }` in deren `elm.json`-Dateien deklariert wurde. Wenn jemand mit `elm-build-system` *Elm-Anwendungen* erstellen möchte (für die `{ "type": "application" }` deklariert wurde), ist das möglich, aber es müssen eigens Änderungen an den Erstellungsphasen vorgenommen werden. Beispiele sind in den Definitionen der Beispielanwendung `elm-todomvc` und im `elm`-Paket selbst zu finden (weil die Web-Oberfläche des Befehls '`elm reactor`' eine Elm-Anwendung ist).
- In Elm können mehrere Versionen desselben Pakets nebeneinander unter `ELM_HOME` vorkommen, aber mit `elm-build-system` klappt das noch nicht so gut. Diese Einschränkung gilt vor allem für Elm-Anwendungen, weil dort die Versionen ihrer Abhängigkeiten genau festgelegt werden, während Elm-Pakete mit einem Bereich von Versionen umgehen können. Ein Ausweg wird in der oben genannten Beispielanwendung genommen, nämlich benutzt sie die Prozedur `patch-application-dependencies` aus `(guix build elm-build-system)`, um die `elm.json`-Dateien umzuschreiben, damit sie sich stattdessen auf die Paketversionen beziehen, die es in der Erstellungsumgebung tatsächlich gibt. Alternativ könnte man auch auf Guix-Paketumwandlungen zurückgreifen (siehe Abschnitt 9.3 [Paketvarianten definieren], Seite 121), um den gesamten Abhängigkeitsgraphen einer Anwendung umzuschreiben.
- Wir sind noch nicht so weit, dass wir den Testkatalog für Elm-Projekte durchlaufen lassen könnten, weil es in Guix weder ein Paket für `elm-test-rs` (`https://github.com/mpizenberg/elm-test-rs`) noch für den Node.js-basierten `elm-test` (`https://github.com/rtfeldman/node-test-runner`) gibt, um Testläufe durchzuführen.

`go-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system go)` exportiert. Mit ihr ist eine Erstellungsprozedur für Go-Pakete implementiert, die dem normalen Go-Erstellungsmechanismus (`https://golang.org/cmd/go/#hdr-Compile_packages_and_dependencies`) entspricht.

Beim Aufruf wird ein Wert für den Schlüssel `#:import-path` und manchmal auch für `#:unpack-path` erwartet. Der „import path“ (`https://golang.org/doc/code.html#ImportPaths`) entspricht dem Dateisystempfad, den die Erstellungsskripts des Pakets und darauf Bezug nehmende Pakete erwarten; durch ihn wird ein Go-Paket eindeutig bezeichnet. Typischerweise setzt er sich aus einer Kombination der entfernten URI des Paketquellcodes und der Dateisystemhierarchie zusammen. Manchmal ist es nötig, den Paketquellcode in ein anderes als das vom „import path“ bezeichnete

Verzeichnis zu entpacken; diese andere Verzeichnisstruktur sollte dann als `#:unpack-path` angegeben werden.

Pakete, die Go-Bibliotheken zur Verfügung stellen, sollten ihren Quellcode auch in die Erstellungsausgabe installieren. Der Schlüssel `#:install-source?`, sein Vorgabewert ist `#t`, steuert, ob Quellcode installiert wird. Bei Paketen, die nur ausführbare Dateien liefern, kann der Wert auf `#f` gesetzt werden.

Cross-Erstellungen von Paketen sind möglich. Wenn für eine bestimmte Architektur oder ein bestimmtes Betriebssystem erstellt werden muss, kann man mit den Schlüsselwörtern `#:goarch` und `#:goos` erzwingen, dass das Paket für diese Architektur und dieses Betriebssystem erstellt wird. Die für Go nutzbaren Kombinationen sind in der Go-Dokumentation (`"https://golang.org/doc/install/source#environment"`) nachlesbar.

`glib-or-gtk-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system glib-or-gtk)` exportiert. Sie ist für Pakete gedacht, die GLib oder GTK benutzen.

Dieses Erstellungssystem fügt die folgenden zwei Phasen zu denen von `gnu-build-system` hinzu:

`glib-or-gtk-wrap`

Die Phase `glib-or-gtk-wrap` stellt sicher, dass Programme in `bin/` in der Lage sind, GLib-„Schemata“ und GTK-Module (`https://developer.gnome.org/gtk3/stable/gtk-running.html`) zu finden. Dazu wird für das Programm ein Wrapper-Skript erzeugt, dass das eigentliche Programm mit den richtigen Werten für die Umgebungsvariablen `XDG_DATA_DIRS` und `GTK_PATH` aufruft.

Es ist möglich, bestimmte Paketausgaben von diesem Wrapping-Prozess auszunehmen, indem Sie eine Liste ihrer Namen im Parameter `#:glib-or-gtk-wrap-excluded-outputs` angeben. Das ist nützlich, wenn man von einer Ausgabe weiß, dass sie keine Binärdateien enthält, die GLib oder GTK benutzen, und diese Ausgabe durch das Wrappen ohne Not eine weitere Abhängigkeit von GLib und GTK bekäme.

`glib-or-gtk-compile-schemas`

Mit der Phase `glib-or-gtk-compile-schemas` wird sichergestellt, dass alle GSettings-Schemata (`https://developer.gnome.org/gio/stable/glib-compile-schemas.html`) für GLib kompiliert werden. Dazu wird das Programm `glib-compile-schemas` ausgeführt. Es kommt aus dem Paket `glib:bin`, was automatisch vom Erstellungssystem importiert wird. Welches `glib`-Paket dieses `glib-compile-schemas` bereitstellt, kann mit dem Parameter `#:glib` spezifiziert werden.

Beide Phasen finden nach der `install`-Phase statt.

`guile-build-system` [Scheme-Variable]

Dieses Erstellungssystem ist für Guile-Pakete gedacht, die nur aus Scheme-Code bestehen und so schlicht sind, dass sie nicht einmal ein Makefile und erst recht keinen `configure`-Skript enthalten. Hierzu wird Scheme-Code mit `guild compile` kompiliert (siehe Abschnitt “Compilation” in *Referenzhandbuch zu GNU Guile*) und die

`.scm`- und `.go`-Dateien an den richtigen Pfad installiert. Auch Dokumentation wird installiert.

Das Erstellungssystem unterstützt Cross-Kompilieren durch die Befehlszeilenoption `--target` für '`guild compile`'.

Mit `guile-build-system` erstellte Pakete müssen ein Guile-Paket in ihrem `native-inputs`-Feld aufführen.

`julia-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system julia)` exportiert. Sie entspricht einer Implementierung der durch Julia-Pakete (`https://julialang.org/`) genutzten Erstellungsprozedur und verhält sich im Prinzip so, wie wenn man '`julia -e 'using Pkg; Pkg.add(paket)'`' in einer Umgebung ausführt, in der die Umgebungsvariable `JULIA_LOAD_PATH` die Pfade aller Julia-Pakete unter den Paketeingaben enthält. Tests werden durch Aufruf von `/test/runtests.jl` ausgeführt.

Der Name des Julia-Pakets und seine UUID werden aus der Datei `Project.toml` ausgelesen. Durch Angabe des Arguments `#:julia-package-name` (die Groß-/Kleinschreibung muss stimmen) bzw. durch `#:julia-package-uuid` können andere Werte verwendet werden.

Julia-Pakete verwalten ihre Abhängigkeiten zu Binärdateien für gewöhnlich mittels `JLLWrappers.jl`, einem Julia-Paket, das ein Modul erzeugt (benannt nach der Bibliothek, die zugänglich gemacht wird, gefolgt von `_jll.jl`).

Um die `_jll.jl`-Pakete mit den Pfaden zu Binärdateien hinzuzufügen, müssen Sie die Dateien in `src/wrappers/` patchen, um dem Aufruf an das Makro `JLLWrappers.@generate_wrapper_header` noch ein zweites Argument mit dem Store-Pfad der Binärdatei mitzugeben.

Zum Beispiel fügen wir für das MbedTLS-Julia-Paket eine Erstellungsphase hinzu (siehe Abschnitt 9.6 [Erstellungsphasen], Seite 150), in der der absolute Dateiname des zugänglich gemachten MbedTLS-Pakets hinzugefügt wird:

```
(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-apache") "\")\n"))))
              ;; Es gibt eine Julia-Datei für jede Plattform,
              ;; wir ändern sie alle.
              (find-files "src/wrappers/" "\\.jl$"))))
```

Für manche ältere Pakete, die noch keine `Project.toml` benutzen, muss auch diese Datei erstellt werden. Das geht intern vonstatten, sofern die Argumente `#:julia-package-name` und `#:julia-package-uuid` übergeben werden.

`maven-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system maven)` exportiert. Darin wird eine Erstellungsprozedur für Maven-Pakete (`https://maven.apache.org`) implemen-

tiert. Maven ist ein Werkzeug zur Verwaltung der Abhängigkeiten und des „Lebenszyklus“ eines Java-Projekts. Um Maven zu benutzen, werden Abhängigkeiten und Plugins in einer Datei `pom.xml` angegeben, die Maven ausliest. Wenn Maven Abhängigkeiten oder Plugins fehlen, lädt es sie herunter und erstellt damit das Paket.

Durch Guix' Maven-Erstellungssystem wird gewährleistet, dass Maven im Offline-Modus läuft und somit nicht versucht, Abhängigkeiten herunterzuladen. Maven wird in einen Fehler laufen, wenn eine Abhängigkeit fehlt. Bevor Maven ausgeführt wird, wird der Inhalt der `pom.xml` (auch in Unterprojekten) so verändert, dass darin die diejenige Version von Abhängigkeiten und Plugins aufgeführt wird, die in Guix' Erstellungsumgebung vorliegt. Abhängigkeiten und Plugins müssen in das vorgetäuschte Maven-Repository unter `lib/m2` installiert werden; bevor Maven ausgeführt wird, werden symbolische Verknüpfungen in ein echtes Repository hergestellt. Maven wird angewiesen, dieses Repository für die Erstellung zu verwenden und installiert erstellte Artefakte dorthin. Geänderte Dateien werden ins Verzeichnis `lib/m2` der Paketausgabe kopiert.

Sie können eine `pom.xml`-Datei über das Argument `#:pom-file` festlegen oder die vom Erstellungssystem vorgegebene `pom.xml`-Datei im Quellverzeichnis verwenden lassen.

Sollten Sie die Version einer Abhängigkeit manuell angeben müssen, können Sie dafür das Argument `#:local-packages` benutzen. Es nimmt eine assoziative Liste entgegen, deren Schlüssel jeweils die `groupId` des Pakets und deren Wert eine assoziative Liste ist, deren Schlüssel wiederum die `artifactId` des Pakets und deren Wert die einzusetzende Version in `pom.xml` ist.

Manche Pakete haben Abhängigkeiten oder Plugins, die weder zur Laufzeit noch zur Erstellungszeit in Guix sinnvoll sind. Sie können sie entfernen, indem Sie das `#:exclude`-Argument verwenden, um die `pom.xml`-Datei zu bearbeiten. Sein Wert ist eine assoziative Liste, deren Schlüssel die `groupId` des Plugins oder der Abhängigkeit ist, die Sie entfernen möchten, und deren Wert eine Liste von zu entfernenden `artifactId`-Vorkommen angibt.

Sie können statt der vorgegebenen `jdk`- und `maven`-Pakete andere Pakete mit dem entsprechenden Argument, `#:jdk` bzw. `#:maven`, verwenden.

Mit dem Argument `#:maven-plugins` geben Sie eine Liste von Maven-Plugins zur Nutzung während der Erstellung an, im gleichen Format wie das `inputs`-Feld der Paketdeklaration. Der Vorgabewert ist `(default-maven-plugins)`; diese Variable wird exportiert, falls Sie sie benutzen möchten.

`minetest-mod-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system minetest)` exportiert. Mit ihr ist ein Erstellungssystem für Mods für Minetest (`https://www.minetest.net`) implementiert, was bedeutet, Lua-Code, Bilder und andere Ressourcen an den Ort zu kopieren, wo Minetest nach Mods sucht. Das Erstellungssystem verkleinert auch PNG-Bilder und prüft, dass Minetest die Mod fehlerfrei laden kann.

`minify-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system minify)` exportiert. Sie implementiert eine Prozedur zur Minifikation einfacher JavaScript-Pakete.

Es fügt `uglify-js` zur Menge der Eingaben hinzu und komprimiert damit alle JavaScript-Dateien im `src`-Verzeichnis. Ein anderes Programm zur Minifikation kann verwendet werden, indem es mit dem Parameter `#:uglify-js` angegeben wird; es wird erwartet, dass das angegebene Paket den minifizierten Code auf der Standardausgabe ausgibt.

Wenn die Eingabe-JavaScript-Dateien nicht alle im `src`-Verzeichnis liegen, kann mit dem Parameter `#:javascript-files` eine Liste der Dateinamen übergeben werden, auf die das Minifikationsprogramm aufgerufen wird.

`ocaml-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system ocaml)` exportiert. Mit ihr ist ein Erstellungssystem für OCaml-Pakete (`https://ocaml.org`) implementiert, was bedeutet, dass es die richtigen auszuführenden Befehle für das jeweilige Paket auswählt. OCaml-Pakete können sehr unterschiedliche Befehle erwarten. Dieses Erstellungssystem probiert manche davon durch.

Wenn im Paket eine Datei `setup.ml` auf oberster Ebene vorhanden ist, wird `ocaml setup.ml -configure`, `ocaml setup.ml -build` und `ocaml setup.ml -install` ausgeführt. Das Erstellungssystem wird annehmen, dass die Datei durch OASIS (`http://oasis.forge.ocamlcore.org/`) erzeugt wurde, und wird das Präfix setzen und Tests aktivieren, wenn diese nicht abgeschaltet wurden. Sie können Befehlszeilenoptionen zum Konfigurieren und Erstellen mit den Parametern `#:configure-flags` und `#:build-flags` übergeben. Der Schlüssel `#:test-flags` kann übergeben werden, um die Befehlszeilenoptionen zu ändern, mit denen die Tests aktiviert werden. Mit dem Parameter `#:use-make?` kann dieses Erstellungssystem für die build- und install-Phasen abgeschaltet werden.

Verfügt das Paket über eine `configure`-Datei, wird angenommen, dass diese von Hand geschrieben wurde mit einem anderen Format für Argumente als bei einem Skript des `gnu-build-system`. Sie können weitere Befehlszeilenoptionen mit dem Schlüssel `#:configure-flags` hinzufügen.

Falls dem Paket ein `Makefile` beiliegt (oder `#:use-make?` auf `#t` gesetzt wurde), wird dieses benutzt und weitere Befehlszeilenoptionen können mit dem Schlüssel `#:make-flags` zu den build- und install-Phasen hinzugefügt werden.

Letztlich gibt es in manchen Pakete keine solchen Dateien, sie halten sich aber an bestimmte Konventionen, wo ihr eigenes Erstellungssystem zu finden ist. In diesem Fall führt Guix' OCaml-Erstellungssystem `ocaml pkg/pkg.ml` oder `ocaml pkg/build.ml` aus und kümmert sich darum, dass der Pfad zu dem benötigten findlib-Modul passt. Weitere Befehlszeilenoptionen können über den Schlüssel `#:build-flags` übergeben werden. Um die Installation kümmert sich `opam-installer`. In diesem Fall muss das `opam`-Paket im `native-inputs`-Feld der Paketdefinition stehen.

Beachten Sie, dass die meisten OCaml-Pakete davon ausgehen, dass sie in dasselbe Verzeichnis wie OCaml selbst installiert werden, was wir in Guix aber nicht so haben wollen. Solche Pakete installieren ihre `.so`-Dateien in das Verzeichnis ihres Moduls, was für die meisten anderen Einrichtungen funktioniert, weil es im OCaml-Compilerverzeichnis liegt. Jedoch können so in Guix die Bibliotheken nicht gefunden werden, deswegen benutzen wir `CAML_LD_LIBRARY_PATH`. Diese Umgebungsvariable

zeigt auf `lib/ocaml/site-lib/stubslibs` und dorthin sollten `.so`-Bibliotheken installiert werden.

`python-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system python)` exportiert. Sie implementiert mehr oder weniger die konventionelle Erstellungsprozedur, wie sie für Python-Pakete üblich ist, d.h. erst wird `python setup.py build` ausgeführt und dann `python setup.py install --prefix=/gnu/store/...`.

Für Pakete, die eigenständige Python-Programme nach `bin/` installieren, sorgt dieses Erstellungssystem dafür, dass die Programme in ein Wrapper-Skript verpackt werden, welches die eigentlichen Programme mit einer Umgebungsvariablen `GUIX_PYTHONPATH` aufruft, die alle Python-Bibliotheken auflistet, von denen die Programme abhängen.

Welches Python-Paket benutzt wird, um die Erstellung durchzuführen, kann mit dem Parameter `#:python` bestimmt werden. Das ist nützlich, wenn wir erzwingen wollen, dass ein Paket mit einer bestimmten Version des Python-Interpretierers arbeitet. Das kann nötig sein, wenn das Programm nur mit einer einzigen Interpretiererversion kompatibel ist.

Standardmäßig ruft Guix `setup.py` auf, was zu `setuptools` gehört, ähnlich wie es auch `pip` tut. Manche Pakete sind mit setuptools (und pip) inkompatibel, deswegen können Sie diese Einstellung abschalten, indem Sie den Parameter `#:use-setuptools?` auf `#f` setzen.

Wenn eine der Ausgaben `"python"` heißt, wird das Paket dort hinein installiert und *nicht* in die vorgegebene Ausgabe `"out"`. Das ist für Pakete gedacht, bei denen ein Python-Paket nur einen Teil der Software ausmacht, man also die Phasen des `python-build-system` mit einem anderen Erstellungssystem zusammen verwenden wollen könnte. Oft kann man das bei Anbindungen für Python gebrauchen.

`pyproject-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `guix build-system pyproject` exportiert. Es basiert auf *python-build-system* und fügt Unterstützung für `pyproject.toml` und PEP 517 (`https://peps.python.org/pep-0517/`) hinzu. Auch kommt Unterstützung für verschiedene Build Backends und Testrahmen hinzu.

Die Programmierschnittstelle unterscheidet sich leicht von *python-build-system*:

- `#:use-setuptools?` und `#:test-target` wurden entfernt.
- `#:build-backend` ist neu. Die Vorgabe dafür ist `#false`, so dass das richtige Backend wenn möglich aufgrund von `pyproject.toml` bestimmt wird.
- `#:test-backend` ist neu. Die Vorgabe dafür ist `#false`, so dass das richtige Test-Backend wenn möglich aufgrund der Paketeingaben gewählt wird.
- `#:test-flags` ist neu. Vorgegeben ist `'()`. Die Optionen darin werden als Befehlszeilenargumente an den Test-Befehl übermittelt. Beachten Sie, dass ausführliche Ausgaben immer aktiviert werden, falls dies für das Backend implementiert ist.

Wir stufen es als „experimentell“ ein, weil die Details der Implementierung noch *nicht* in Stein gemeißelt sind. Dennoch würden wir es begrüßen, wenn Sie es für

neue Python-Projekte verwenden (auch für solche, die `setup.py` haben). Die Programmierschnittstelle wird sich noch ändern, aber wir werden uns um im Guix-Kanal aufkommende Probleme kümmern.

Schlussendlich wird dieses Erstellungssystem für veraltet erklärt werden und in *python-build-system* aufgehen, wahrscheinlich irgendwann im Jahr 2024.

`perl-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system perl)` exportiert. Mit ihr wird die Standard-Erstellungsprozedur für Perl-Pakete implementiert, welche entweder darin besteht, `perl Build.PL --prefix=/gnu/store/...` gefolgt von `Build` und `Build install` auszuführen, oder `perl Makefile.PL PREFIX=/gnu/store/...` gefolgt von `make` und `make install` auszuführen, je nachdem, ob eine Datei `Build.PL` oder eine Datei `Makefile.PL` in der Paketdistribution vorliegt. Den Vorrang hat erstere, wenn sowohl `Build.PL` als auch `Makefile.PL` in der Paketdistribution existieren. Der Vorrang kann umgekehrt werden, indem `#t` für den Parameter `#:make-maker?` angegeben wird.

Der erste Aufruf von `perl Makefile.PL` oder `perl Build.PL` übergibt die im Parameter `#:make-maker-flags` bzw. `#:module-build-flags` angegebenen Befehlszeilenoptionen, je nachdem, was verwendet wird.

Welches Perl-Paket dafür benutzt wird, kann mit `#:perl` angegeben werden.

`renpy-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system renpy)` exportiert. Sie implementiert mehr oder weniger die herkömmliche Erstellungsprozedur, die für Ren'py-Spiele benutzt wird. Das bedeutet, das `#:game` wird einmal geladen, wodurch Bytecode dafür erzeugt wird.

Des Weiteren wird ein Wrapper-Skript in `bin/` und eine *.desktop-Datei in `share/applications` erzeugt. Mit beiden davon kann man das Spiel starten.

Welches Ren'py-Paket benutzt wird, gibt man mit `#:renpy` an. Spiele können auch in andere Ausgaben als in „out“ installiert werden, indem man `#:output` benutzt.

`qt-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system qt)` exportiert. Sie ist für Anwendungen gedacht, die Qt oder KDE benutzen.

Dieses Erstellungssystem fügt die folgenden zwei Phasen zu denen von `cmake-build-system` hinzu:

`check-setup`

Die Phase `check-setup` bereitet die Umgebung für Überprüfungen vor, wie sie von Qt-Test-Programmen üblicherweise benutzt werden. Zurzeit werden nur manche Umgebungsvariable gesetzt: `QT_QPA_PLATFORM=offscreen`, `DBUS_FATAL_WARNINGS=0` und `CTEST_OUTPUT_ON_FAILURE=1`.

Diese Phase wird vor der `check`-Phase hinzugefügt. Es handelt sich um eine eigene Phase, die nach Bedarf angepasst werden kann.

`qt-wrap` In der Phase `qt-wrap` wird nach Qt5-Plugin-Pfaden, QML-Pfaden und manchen XDG-Daten in den Ein- und Ausgaben gesucht. Wenn solch

ein Pfad gefunden wird, werden für alle Programme in den Verzeichnissen `bin/`, `sbin/`, `libexec/` und `lib/libexec/` in der Ausgabe Wrapper-Skripte erzeugt, die die nötigen Umgebungsvariablen definieren.

Es ist möglich, bestimmte Paketausgaben von diesem Wrapping-Prozess auszunehmen, indem Sie eine Liste ihrer Namen im Parameter `#:qt-wrap-excluded-outputs` angeben. Das ist nützlich, wenn man von einer Ausgabe weiß, dass sie keine Qt-Binärdateien enthält, und diese Ausgabe durch das Wrappen ohne Not eine weitere Abhängigkeit von Qt, KDE oder Ähnlichem bekäme.

Diese Phase wird nach der `install`-Phase hinzugefügt.

`r-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system r)` exportiert. Sie entspricht einer Implementierung der durch R-Pakete (`https://r-project.org`) genutzten Erstellungsprozedur, die wenig mehr tut, als '`R CMD INSTALL --library=/gnu/store/...`' in einer Umgebung auszuführen, in der die Umgebungsvariable `R_LIBS_SITE` die Pfade aller R-Pakete unter den Paketeingaben enthält. Tests werden nach der Installation mit der R-Funktion `tools::testInstalledPackage` ausgeführt.

`rakudo-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system rakudo)` exportiert. Sie implementiert die Erstellungsprozedur, die von Rakudo (`https://rakudo.org/`) für Perl6-Pakete (`https://perl6.org/`) benutzt wird. Pakete werden ins Verzeichnis `/gnu/store/.../NAME-VERSION/share/perl6` abgelegt und Binärdateien, Bibliotheksdateien und Ressourcen werden installiert, zudem werden die Dateien im Verzeichnis `bin/` in Wrapper-Skripte verpackt. Tests können übersprungen werden, indem man `#f` im Parameter `tests?` übergibt.

Welches rakudo-Paket benutzt werden soll, kann mit dem Parameter `rakudo` angegeben werden. Das perl6-tap-harness-Paket, das für die Tests benutzt wird, kann mit `#:prove6` ausgewählt werden; es kann auch entfernt werden, indem man `#f` für den Parameter `with-prove6?` übergibt. Welches perl6-zef-Paket für Tests und Installation verwendet wird, kann mit dem Parameter `#:zef` angegeben werden; es kann auch entfernt werden, indem man `#f` für den Parameter `with-zef?` übergibt.

`rebar-build-system` [Scheme-Variable]

Diese Variable wird von `(guix build-system rebar)` exportiert. Sie implementiert eine Erstellungsprozedur, die auf rebar3 (`https://rebar3.org`) aufbaut, einem Erstellungssystem für Programme, die in der Sprache Erlang geschrieben sind.

Das Erstellungssystem fügt sowohl `rebar3` als auch `erlang` zu den Eingaben hinzu. Sollen stattdessen andere Pakete benutzt werden, können diese jeweils mit den Parametern `#:rebar` und `#:erlang` spezifiziert werden.

Dieses Erstellungssystem basiert auf `gnu-build-system`, bei dem aber die folgenden Phasen geändert wurden:

`unpack` Diese Phase entpackt die Quelle wie es auch das `gnu-build-system` tut, schaut nach einer Datei `contents.tar.gz` auf der obersten Ebene des

Quellbaumes und wenn diese existiert, wird auch sie entpackt. Das erleichtert den Umgang mit Paketen, die auf `https://hex.pm/`, der Paketsammlung für Erlang und Elixir, angeboten werden.

`bootstrap`
`configure`
: Die Phasen `bootstrap` und `configure` fehlen, weil Erlang-Pakete normalerweise nicht konfiguriert werden müssen.

`build`
: In dieser Phase wird `rebar3 compile` mit den Befehlszeilenoptionen aus `#:rebar-flags` ausgeführt.

`check`
: Sofern *nicht* `#:tests? #f` übergeben wird, wird in dieser Phase `rebar3 eunit` ausgeführt oder das Gleiche auf ein anderes bei `#:test-target` angegebenes Ziel. Dabei werden die Befehlszeilenoptionen aus `#:rebar-flags` übergeben.

`install`
: Hier werden die im standardmäßigen Profil *default* erzeugten Dateien installiert oder die aus einem mit `#:install-profile` angegebenen anderen Profil.

`texlive-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system texlive)` exportiert. Mit ihr werden TeX-Pakete in Stapelverarbeitung („batch mode“) mit der angegebenen Engine erstellt. Das Erstellungssystem setzt die Variable `TEXINPUTS` so, dass alle TeX-Quelldateien unter den Eingaben gefunden werden können.

Standardmäßig wird `luatex` auf allen Dateien mit der Dateiendung `ins` ausgeführt. Eine andere Engine oder ein anderes Format kann mit dem Argument `#:tex-format` angegeben werden. Verschiedene Erstellungsziele können mit dem Argument `#:build-targets` festgelegt werden, das eine Liste von Dateinamen erwartet. Das Erstellungssystem fügt nur `texlive-bin` und `texlive-latex-base` zu den Eingaben hinzu (beide kommen aus dem Modul `(gnu packages tex)`. Für beide kann das zu benutzende Paket jeweils mit den Argumenten `#:texlive-bin` oder `#:texlive-latex-base` geändert werden.

Der Parameter `#:tex-directory` sagt dem Erstellungssystem, wohin die installierten Dateien im texmf-Verzeichnisbaum installiert werden sollen.

`ruby-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system ruby)` exportiert. Sie steht für eine Implementierung der RubyGems-Erstellungsprozedur, die für Ruby-Pakete benutzt wird, wobei `gem build` gefolgt von `gem install` ausgeführt wird.

Das `source`-Feld eines Pakets, das dieses Erstellungssystem benutzt, verweist typischerweise auf ein Gem-Archiv, weil Ruby-Entwickler dieses Format benutzen, wenn sie ihre Software veröffentlichen. Das Erstellungssystem entpackt das Gem-Archiv, spielt eventuell Patches für den Quellcode ein, führt die Tests aus, verpackt alles wieder in ein Gem-Archiv und installiert dieses. Neben Gem-Archiven darf das Feld auch auf Verzeichnisse und Tarballs verweisen, damit es auch möglich ist, unveröffentlichte Gems aus einem Git-Repository oder traditionelle Quellcode-Veröffentlichungen zu benutzen.

Welches Ruby-Paket benutzt werden soll, kann mit dem Parameter `#:ruby` festgelegt werden. Eine Liste zusätzlicher Befehlszeilenoptionen für den Aufruf des `gem`-Befehls kann mit dem Parameter `#:gem-flags` angegeben werden.

`waf-build-system` [Scheme-Variable]

Diese Variable wird durch das Modul `(guix build-system waf)` exportiert. Damit ist eine Erstellungsprozedur rund um das `waf`-Skript implementiert. Die üblichen Phasen – `configure`, `build` und `install` – sind implementiert, indem deren Namen als Argumente an das `waf`-Skript übergeben werden.

Das `waf`-Skript wird vom Python-Interpetierer ausgeführt. Mit welchem Python-Paket das Skript ausgeführt werden soll, kann mit dem Parameter `#:python` angegeben werden.

`scons-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system scons)` exportiert. Sie steht für eine Implementierung der Erstellungsprozedur, die das SCons-Softwarekonstruktionswerkzeug („software construction tool“) benutzt. Das Erstellungssystem führt `scons` aus, um das Paket zu erstellen, führt mit `scons test` Tests aus und benutzt `scons install`, um das Paket zu installieren.

Zusätzliche Optionen, die an `scons` übergeben werden sollen, können mit dem Parameter `#:scons-flags` angegeben werden. Die voreingestellten Erstellungs- und Installationsziele können jeweils durch `#:build-targets` und `#:install-targets` ersetzt werden. Die Python-Version, die benutzt werden soll, um SCons auszuführen, kann festgelegt werden, indem das passende SCons-Paket mit dem Parameter `#:scons` ausgewählt wird.

`haskell-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system haskell)` exportiert. Sie bietet Zugang zur Cabal-Erstellungsprozedur, die von Haskell-Paketen benutzt wird, was bedeutet, `runhaskell Setup.hs configure --prefix=/gnu/store/...` und `runhaskell Setup.hs build` auszuführen. Statt das Paket mit dem Befehl `runhaskell Setup.hs install` zu installieren, benutzt das Erstellungssystem `runhaskell Setup.hs copy` gefolgt von `runhaskell Setup.hs register`, um keine Bibliotheken im Store-Verzeichnis des Compilers zu speichern, auf dem keine Schreibberechtigung besteht. Zusätzlich generiert das Erstellungssystem Dokumentation durch Ausführen von `runhaskell Setup.hs haddock`, außer `#:haddock? #f` wurde übergeben. Optional können an Haddock Parameter mit Hilfe des Parameters `#:haddock-flags` übergeben werden. Wird die Datei `Setup.hs` nicht gefunden, sucht das Erstellungssystem stattdessen nach `Setup.lhs`.

Welcher Haskell-Compiler benutzt werden soll, kann über den `#:haskell`-Parameter angegeben werden. Als Vorgabewert verwendet er `ghc`.

`dub-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system dub)` exportiert. Sie verweist auf eine Implementierung des Dub-Erstellungssystems, das von D-Paketen benutzt wird. Dabei werden `dub build` und `dub run` ausgeführt. Die Installation wird durch manuelles Kopieren der Dateien durchgeführt.

Welcher D-Compiler benutzt wird, kann mit dem Parameter `#:ldc` festgelegt werden, was als Vorgabewert `ldc` benutzt.

`emacs-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system emacs)` exportiert. Darin wird eine Installationsprozedur ähnlich der des Paketsystems von Emacs selbst implementiert (siehe Abschnitt "Packages" in *The GNU Emacs Manual*).

Zunächst wird eine Datei `Paket-autoloads.el` erzeugt, dann werden alle Emacs-Lisp-Dateien zu Bytecode kompiliert. Anders als beim Emacs-Paketsystem werden die Info-Dokumentationsdateien in das Standardverzeichnis für Dokumentation verschoben und die Datei `dir` gelöscht. Die Dateien des Elisp-Pakets werden direkt in `share/emacs/site-lisp` installiert.

`font-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system font)` exportiert. Mit ihr steht eine Installationsprozedur für Schriftarten-Pakete zur Verfügung für vom Anbieter vorkompilierte TrueType-, OpenType- und andere Schriftartendateien, die nur an die richtige Stelle kopiert werden müssen. Dieses Erstellungssystem kopiert die Schriftartendateien an den Konventionen folgende Orte im Ausgabeverzeichnis.

`meson-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system meson)` exportiert. Sie enthält die Erstellungsprozedur für Pakete, die Meson (`https://mesonbuild.com`) als ihr Erstellungssystem benutzen.

Mit ihr werden sowohl Meson als auch Ninja (`https://ninja-build.org/`) zur Menge der Eingaben hinzugefügt; die Pakete dafür können mit den Parametern `#:meson` und `#:ninja` geändert werden, wenn nötig.

Dieses Erstellungssystem ist eine Erweiterung für das `gnu-build-system`, aber mit Änderungen an den folgenden Phasen, die Meson-spezifisch sind:

`configure`
: Diese Phase führt den `meson`-Befehl mit den in `#:configure-flags` angegebenen Befehlszeilenoptionen aus. Die Befehlszeilenoption `--build-type` wird immer auf `debugoptimized` gesetzt, solange nichts anderes mit dem Parameter `#:build-type` angegeben wurde.

`build`
: Diese Phase ruft `ninja` auf, um das Paket standardmäßig parallel zu erstellen. Die Vorgabeeinstellung, dass parallel erstellt wird, kann verändert werden durch Setzen von `#:parallel-build?`.

`check`
: Die Phase führt '`meson test`' mit einer unveränderlichen Grundmenge von Optionen aus. Diese Grundmenge können Sie mit dem Argument `#:test-options` erweitern, um zum Beispiel einen bestimmten Testkatalog auszuwählen oder zu überspringen.

`install`
: Diese Phase führt `ninja install` aus und kann nicht verändert werden.

Dazu fügt das Erstellungssystem noch folgende neue Phasen:

`fix-runpath`
: In dieser Phase wird sichergestellt, dass alle Binärdateien die von ihnen benötigten Bibliotheken finden können. Die benötigten Bibliotheken wer-

den in den Unterverzeichnissen des Pakets, das erstellt wird, gesucht, und zum `RUNPATH` hinzugefügt, wann immer es nötig ist. Auch werden diejenigen Referenzen zu Bibliotheken aus der Erstellungsphase wieder entfernt, die bei `meson` hinzugefügt wurden, aber eigentlich zur Laufzeit nicht gebraucht werden, wie Abhängigkeiten nur für Tests.

`glib-or-gtk-wrap`
: Diese Phase ist dieselbe, die auch im `glib-or-gtk-build-system` zur Verfügung gestellt wird, und mit Vorgabeeinstellungen wird sie nicht durchlaufen. Wenn sie gebraucht wird, kann sie mit dem Parameter `#:glib-or-gtk?` aktiviert werden.

`glib-or-gtk-compile-schemas`
: Diese Phase ist dieselbe, die auch im `glib-or-gtk-build-system` zur Verfügung gestellt wird, und mit Vorgabeeinstellungen wird sie nicht durchlaufen. Wenn sie gebraucht wird, kann sie mit dem Parameter `#:glib-or-gtk?` aktiviert werden.

`linux-module-build-system` [Scheme-Variable]

Mit `linux-module-build-system` können Linux-Kernelmodule erstellt werden.

Dieses Erstellungssystem ist eine Erweiterung des `gnu-build-system`, bei der aber die folgenden Phasen geändert wurden:

`configure`
: Diese Phase konfiguriert die Umgebung so, dass das externe Kernel-Modul durch das Makefile des Linux-Kernels erstellt werden kann.

`build` Diese Phase benutzt das Makefile des Linux-Kernels, um das externe Kernel-Modul zu erstellen.

`install` Diese Phase benutzt das Makefile des Linux-Kernels zur Installation des externen Kernel-Moduls.

Es ist möglich und hilfreich, den für die Erstellung des Moduls zu benutzenden Linux-Kernel anzugeben (in der `arguments`-Form eines Pakets, dass das `linux-module-build-system` als Erstellungssystem benutzt, wird dazu der Schlüssel `#:linux` benutzt).

`node-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system node)` exportiert. Sie stellt eine Implementierung der Erstellungsprozedur von Node.js (`https://nodejs.org`) dar, die annäherungsweise der Funktion des Befehls `npm install` gefolgt vom Befehl `npm test` entspricht.

Welches Node.js-Paket zur Interpretation der `npm`-Befehle benutzt wird, kann mit dem Parameter `#:node` angegeben werden. Dessen Vorgabewert ist `node`.

Letztlich gibt es für die Pakete, die bei weitem nichts so komplexes brauchen, ein „triviales“ Erstellungssystem. Es ist in dem Sinn trivial, dass es praktisch keine Hilfestellungen gibt: Es fügt keine impliziten Eingaben hinzu und hat kein Konzept von Erstellungsphasen.

`trivial-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system trivial)` exportiert.

Diesem Erstellungssystem muss im Argument `#:builder` ein Scheme-Ausdruck übergeben werden, der die Paketausgabe(n) erstellt – wie bei `build-expression->derivation` (siehe Abschnitt 9.10 [Ableitungen], Seite 167).

`channel-build-system` [Scheme-Variable]

Diese Variable wird vom Modul `(guix build-system channel)` exportiert.

Dieses Erstellungssystem ist in erster Linie auf interne Nutzung ausgelegt. Ein Paket mit diesem Erstellungssystem muss im `source`-Feld eine Kanalspezifikation stehen haben (siehe Kapitel 7 [Kanäle], Seite 77); alternativ kann für `source` der Name eines Verzeichnisses angegeben werden, dann muss aber ein Argument `#:commit` mit dem gewünschten Commit (einer hexadezimalen Zeichenkette) übergeben werden, so dass dieser erstellt wird.

Damit ergit sich ein Paket mit einer Guix-Instanz des angegebenen Kanals, ähnlich der, die `guix time-machine` erstellen würde.

9.6 Erstellungsphasen

Fast alle Erstellungssysteme für Pakete implementieren ein Konzept von *Erstellungsphasen*: einer Abfolge von Aktionen, die vom Erstellungssystem ausgeführt werden, wenn Sie das Paket erstellen. Dabei fallen in den Store installierte Nebenerzeugnisse an. Eine Ausnahme ist der Erwähnung wert: das magere `trivial-build-system` (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130).

Wie im letzten Abschnitt erläutert, stellen diese Erstellungssysteme eine standardisierte Liste von Phasen zur Verfügung. Für `gnu-build-system` sind dies die hauptsächlichen Erstellungsphasen:

`set-paths`
: Suchpfade in Umgebungsvariablen definieren. Dies geschieht für alle Eingabepakete. `PATH` wird so auch festgelegt. Siehe Abschnitt 9.8 [Suchpfade], Seite 161.

`unpack`
: Den Quell-Tarball entpacken und das Arbeitsverzeichnis wechseln in den entpackten Quellbaum. Wenn die Quelle bereits ein Verzeichnis ist, wird es in den Quellbaum kopiert und dorthin gewechselt.

`patch-source-shebangs`
: „Shebangs“ in Quelldateien beheben, damit Sie sich auf die richtigen Store-Dateipfade beziehen. Zum Beispiel könnte `#!/bin/sh` zu `#!/gnu/store/...-bash-4.3/bin/sh` geändert werden.

`configure`
: Das Skript `configure` mit einigen vorgegebenen Befehlszeilenoptionen ausführen, wie z.B. mit `--prefix=/gnu/store/...`, sowie mit den im `#:configure-flags`-Argument angegebenen Optionen.

`build`
: `make` ausführen mit den Optionen aus der Liste in `#:make-flags`. Wenn das Argument `#:parallel-build?` auf wahr gesetzt ist (was der Vorgabewert ist), wird `make -j` zum Erstellen ausgeführt.

`check` `make check` (oder statt `check` ein anderes bei `#:test-target` angegebenes Ziel) ausführen, außer falls `#:tests? #f` gesetzt ist. Wenn das Argument `#:parallel-tests?` auf wahr gesetzt ist (der Vorgabewert), führe `make check -j` aus.

`install` `make install` mit den in `#:make-flags` aufgelisteten Optionen ausführen.

`patch-shebangs`
Shebangs in den installierten ausführbaren Dateien beheben.

`strip` Symbole zur Fehlerbehebung aus ELF-Dateien entfernen (außer `#:strip-binaries?` ist auf falsch gesetzt) und in die `debug`-Ausgabe kopieren, falls diese verfügbar ist (siehe Kapitel 17 [Dateien zur Fehlersuche installieren], Seite 692).

`validate-runpath`
Den `RUNPATH` von ELF-Binärdateien validieren, sofern `#:validate-runpath?` falsch ist (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130).

Die Validierung besteht darin, sicherzustellen, dass jede gemeinsame Bibliothek, die von einer ELF-Binärdatei gebraucht wird (sie sind in `DT_NEEDED`-Einträgen im `PT_DYNAMIC`-Segment der ELF-Datei aufgeführt), im `DT_RUNPATH`-Eintrag der Datei gefunden wird. Mit anderen Worten wird gewährleistet, dass es beim Ausführen oder Benutzen einer solchen Binärdatei nicht zur Laufzeit zu einem Fehler kommt, eine Datei würde nicht gefunden. Siehe Abschnitt “Options” in *The GNU Linker* für mehr Informationen zum `RUNPATH`.

Andere Erstellungssysteme haben ähnliche, aber etwas unterschiedliche Phasen. Zum Beispiel hat das `cmake-build-system` gleichnamige Phasen, aber seine `configure`-Phase führt `cmake` statt `./configure` aus. Andere Erstellungssysteme wie z.B. `python-build-system` haben eine völlig andere Liste von Standardphasen. Dieser gesamte Code läuft *erstellungsseitig*: Er wird dann ausgewertet, wenn Sie das Paket wirklich erstellen, in einem eigenen Erstellungprozess nur dafür, den der Erstellungs-Daemon erzeugt (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

Erstellungsphasen werden durch assoziative Listen (kurz „Alists“) repräsentiert (siehe Abschnitt “Association Lists” in *Referenzhandbuch zu GNU Guile*) wo jeder Schlüssel ein Symbol für den Namen der Phase ist und der assoziierte Wert eine Prozedur ist, die eine beliebige Anzahl von Argumenten nimmt. Nach Konvention empfangen diese Prozeduren dadurch Informationen über die Erstellung in Form von *Schlüsselwort-Parametern*, die darin benutzt oder ignoriert werden können.

Zum Beispiel werden die `%standard-phases`, das ist die Variable mit der Alist der Erstellungsphasen, in `(guix build gnu-build-system)` so definiert[3]:

```
;; Die Erstellungsphasen von 'gnu-build-system'.

(define* (unpack #:key source #:allow-other-keys)
  ;; Quelltarball extrahieren.
  (invoke "tar" "xvf" source))
```

[3] Wir stellen hier nur eine vereinfachte Sicht auf diese Erstellungsphasen vor. Wenn Sie alle Details wollen, schauen Sie sich `(guix build gnu-build-system)` an!

```
(define* (configure #:key outputs #:allow-other-keys)
  ;; 'configure'-Skript ausführen. In Ausgabe "out" installieren.
  (let ((out (assoc-ref outputs "out")))
    (invoke "./configure"
            (string-append "--prefix=" out))))

(define* (build #:allow-other-keys)
  ;; Kompilieren.
  (invoke "make"))

(define* (check #:key (test-target "check") (tests? #true)
                #:allow-other-keys)
  ;; Testkatalog ausführen.
  (if tests?
      (invoke "make" test-target)
      (display "test suite not run\n")))

(define* (install #:allow-other-keys)
  ;; Dateien ins bei 'configure' festgelegte Präfix installieren.
  (invoke "make" "install"))

(define %standard-phases
  ;; Die Liste der Standardphasen (der Kürze halber lassen wir einige
  ;; aus). Jedes Element ist ein Paar aus Symbol und Prozedur.
  (list (cons 'unpack unpack)
        (cons 'configure configure)
        (cons 'build build)
        (cons 'check check)
        (cons 'install install)))
```

Hier sieht man wie `%standard-phases` als eine Liste von Paaren aus Symbol und Prozedur (siehe Abschnitt “Pairs” in *Referenzhandbuch zu GNU Guile*) definiert ist. Das erste Paar assoziiert die `unpack`-Prozedur mit dem `unpack`-Symbol – es gibt einen Namen an. Das zweite Paar definiert die `configure`-Phase auf ähnliche Weise, ebenso die anderen. Wenn ein Paket erstellt wird, das `gnu-build-system` benutzt, werden diese Phasen der Reihe nach ausgeführt. Sie können beim Erstellen von Paketen im Erstellungsprotokoll den Namen jeder gestarteten und abgeschlossenen Phase sehen.

Schauen wir uns jetzt die Prozeduren selbst an. Jede davon wird mit `define*` definiert, dadurch können bei `#:key` die Schlüsselwortparameter aufgelistet werden, die die Prozedur annimmt, wenn gewünscht auch zusammen mit einem Vorgabewert, und durch `#:allow-other-keys` wird veranlasst, dass andere Schlüsselwortparameter ignoriert werden (siehe Abschnitt “Optional Arguments” in *Referenzhandbuch zu GNU Guile*).

Die `unpack`-Prozedur berücksichtigt den `source`-Parameter; das Erstellungssystem benutzt ihn, um den Dateinamen des Quell-Tarballs (oder des Checkouts aus einer Versionskontrolle) zu finden. Die anderen Parameter ignoriert sie. Die `configure`-Phase interessiert sich nur für den `outputs`-Parameter, eine Alist, die die Namen von Paketausgaben auf ih-

re Dateinamen im Store abbildet (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60). Sie extrahiert den Dateinamen für `out`, die Standardausgabe, und gibt ihn an `./configure` als das Installationspräfix weiter, wodurch `make install` zum Schluss alle Dateien in dieses Verzeichnis kopieren wird (siehe Abschnitt "Configuration" in *GNU Coding Standards*). `build` und `install` ignorieren all ihre Argumente. `check` berücksichtigt das Argument `test-target`, worin der Name des Makefile-Ziels angegeben wird, um die Tests auszuführen. Es wird stattdessen eine Nachricht angezeigt und die Tests übersprungen, wenn `tests?` falsch ist.

Die Liste der Phasen, die für ein bestimmtes Paket benutzt werden, kann über den `#:phases`-Parameter an das Erstellungssystem geändert werden. Das Ändern des Satzes von Erstellungsphasen funktioniert so, dass eine neue Alist von Phasen aus der oben beschriebenen `%standard-phases`-Alist heraus erzeugt wird. Dazu können die Standardprozeduren zur Bearbeitung von assoziativen Listen wie `alist-delete` benutzt werden (siehe Abschnitt "SRFI-1 Association Lists" in *Referenzhandbuch zu GNU Guile*), aber es ist bequemer, dafür die Prozedur `modify-phases` zu benutzen (siehe Abschnitt 9.7 [Werkzeuge zur Erstellung], Seite 154).

Hier ist ein Beispiel für eine Paketdefinition, die die `configure`-Phase aus `%standard-phases` entfernt und eine neue Phase vor der `build`-Phase namens `set-prefix-in-makefile` einfügt:

```
(define-public beispiel
  (package
    (name "beispiel")
    ;; wir lassen die anderen Felder aus
    (build-system gnu-build-system)
    (arguments
     '(#:phases (modify-phases %standard-phases
                  (delete 'configure)
                  (add-before 'build 'set-prefix-in-makefile
                    (lambda* (#:key outputs #:allow-other-keys)
                      ;; Makefile anpassen, damit die 'PREFIX'-
                      ;; Variable auf "out" verweist.
                      (let ((out (assoc-ref outputs "out")))
                        (substitute* "Makefile"
                          (("PREFIX =.*")
                           (string-append "PREFIX = "
                                          out "\n")))))))))))
```

Die neu eingefügte Phase wurde als anonyme Prozedur geschrieben. Solche namenlosen Prozeduren schreibt man mit `lambda*`. Oben berücksichtigt sie den `outputs`-Parameter, den wir zuvor gesehen haben. Siehe Abschnitt 9.7 [Werkzeuge zur Erstellung], Seite 154, für mehr Informationen über die in dieser Phase benutzten Hilfsmittel und für mehr Beispiele zu `modify-phases`.

Sie sollten im Kopf behalten, dass Erstellungsphasen aus Code bestehen, der erst dann ausgewertet wird, wenn das Paket erstellt wird. Das ist der Grund, warum der gesamte `modify-phases`-Ausdruck oben quotiert wird. Quotiert heißt, er steht nach einem `'` oder Apostrophenzeichen: Er wird nicht sofort als Code ausgewertet, sondern nur zur späteren

Ausführung vorgemerkt (wir sagen *staged*, siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175, für eine Erläuterung von Code-Staging und den beteiligten *Code-Schichten* (oder „Strata“).

9.7 Werkzeuge zur Erstellung

Sobald Sie anfangen, nichttriviale Paketdefinitionen (siehe Abschnitt 9.2 [Pakete definieren], Seite 109) oder andere Erstellungsaktionen (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) zu schreiben, würden Sie sich wahrscheinlich darüber freuen, Helferlein für „Shell-artige“ Aktionen vordefiniert zu bekommen, also Code, den Sie benutzen können, um Verzeichnisse anzulegen, Dateien rekursiv zu kopieren oder zu löschen, Erstellungsphasen anzupassen und Ähnliches. Das Modul `(guix build utils)` macht solche nützlichen Werkzeugprozeduren verfügbar.

Die meisten Erstellungssysteme laden `(guix build utils)` (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). Wenn Sie also eigene Erstellungsphasen für Ihre Paketdefinitionen schreiben, können Sie in den meisten Fällen annehmen, dass diese Prozeduren bei der Auswertung sichtbar sein werden.

Beim Schreiben von G-Ausdrücken können Sie auf der „Erstellungsseite“ `(guix build utils)` mit `with-imported-modules` importieren und anschließend mit der `use-modules`-Form sichtbar machen (siehe Abschnitt “Using Guile Modules” in *Referenzhandbuch zu GNU Guile*):

```
(with-imported-modules '((guix build utils))  ;importieren
  (computed-file "leerer-verzeichnisbaum"
                 #~(begin
                     ;; Sichtbar machen.
                     (use-modules (guix build utils))

                     ;; Jetzt kann man problemlos 'mkdir-p' nutzen.
                     (mkdir-p (string-append #$output "/a/b/c")))))
```

Der Rest dieses Abschnitts stellt eine Referenz der meisten Werkzeugprozeduren dar, die `(guix build utils)` anbietet.

9.7.1 Umgehen mit Store-Dateinamen

Dieser Abschnitt dokumentiert Prozeduren, die sich mit Dateinamen von Store-Objekten befassen.

`%store-directory` [Scheme-Prozedur]
: Liefert den Verzeichnisnamen des Stores.

`store-file-name? `*`Datei`* [Scheme-Prozedur]
: Liefert wahr zurück, wenn sich *Datei* innerhalb des Stores befindet.

`strip-store-file-name `*`Datei`* [Scheme-Prozedur]
: Liefert den Namen der *Datei*, die im Store liegt, ohne den Anfang `/gnu/store` und ohne die Prüfsumme am Namensanfang. Als Ergebnis ergibt sich typischerweise eine Zeichenkette aus `"`*`Paket-Version`*`"`.

`package-name->name+version` *Name* [Scheme-Prozedur]
: Liefert für den Paket-*Name*n (so etwas wie `"foo-0.9.1b"`) zwei Werte zurück: zum einen `"foo"` und zum anderen `"0.9.1b"`. Wenn der Teil mit der Version fehlt, werden der *Name* und `#f` zurückgeliefert. Am ersten Bindestrich, auf den eine Ziffer folgt, wird der Versionsteil abgetrennt.

9.7.2 Dateitypen

Bei den folgenden Prozeduren geht es um Dateien und Dateitypen.

`directory-exists?` *Verzeichnis* [Scheme-Prozedur]
: Liefert `#t`, wenn das *Verzeichnis* existiert und ein Verzeichnis ist.

`executable-file?` *Datei* [Scheme-Prozedur]
: Liefert `#t`, wenn die *Datei* existiert und ausführbar ist.

`symbolic-link?` *Datei* [Scheme-Prozedur]
: Liefert `#t`, wenn die *Datei* eine symbolische Verknüpfung ist (auch bekannt als „Symlink“).

`elf-file?` *Datei* [Scheme-Prozedur]
`ar-file?` *Datei* [Scheme-Prozedur]
`gzip-file?` *Datei* [Scheme-Prozedur]
: Liefert `#t`, wenn die *Datei* jeweils eine ELF-Datei, ein `ar`-Archiv (etwa eine statische Bibliothek mit `.a`) oder eine gzip-Datei ist.

`reset-gzip-timestamp` *Datei* [*#:keep-mtime? #t*] [Scheme-Prozedur]
: Wenn die *Datei* eine gzip-Datei ist, wird ihr eingebetteter Zeitstempel zurückgesetzt (wie bei `gzip --no-name`) und wahr geliefert. Ansonsten wird `#f` geliefert. Wenn *keep-mtime?* wahr ist, wird der Zeitstempel der letzten Modifikation von *Datei* beibehalten.

9.7.3 Änderungen an Dateien

Die folgenden Prozeduren und Makros helfen beim Erstellen, Ändern und Löschen von Dateien. Sie machen Funktionen ähnlich zu Shell-Werkzeugen wie `mkdir -p`, `cp -r`, `rm -r` und `sed` verfügbar. Sie ergänzen Guiles ausgiebige aber kleinschrittige Dateisystemschnittstelle (siehe Abschnitt “POSIX” in *Referenzhandbuch zu GNU Guile*).

`with-directory-excursion` *Verzeichnis Rumpf...* [Scheme-Syntax]
: Den *Rumpf* ausführen mit dem *Verzeichnis* als aktuellem Verzeichnis des Prozesses.

  Im Grunde ändert das Makro das aktuelle Arbeitsverzeichnis auf *Verzeichnis* bevor der *Rumpf* ausgewertet wird, mittels `chdir` (siehe Abschnitt “Processes” in *Referenzhandbuch zu GNU Guile*). Wenn der dynamische Bereich von *Rumpf* wieder verlassen wird, wechselt es wieder ins anfängliche Verzeichnis zurück, egal ob der *Rumpf* durch normales Zurückliefern eines Ergebnisses oder durch einen nichtlokalen Sprung wie etwa eine Ausnahme verlassen wurde.

`mkdir-p` *Verzeichnis* [Scheme-Prozedur]
: Das *Verzeichnis* und all seine Vorgänger erstellen.

`install-file` *Datei Verzeichnis* [Scheme-Prozedur]
Verzeichnis erstellen, wenn es noch nicht existiert, und die *Datei* mit ihrem Namen dorthin kopieren.

`make-file-writable` *Datei* [Scheme-Prozedur]
Dem Besitzer der *Datei* Schreibberechtigung darauf erteilen.

`copy-recursively` *Quelle Zielort* [*#:log* [Scheme-Prozedur]
(*current-output-port*)] [*#:follow-symlinks? #f*] [*#:copy-file*
copy-file] [#:keep-mtime? #f] [#:keep-permissions? #t] Das Verzeichnis *Quelle* rekursiv an den *Zielort* kopieren. Wenn *follow-symlinks?* wahr ist, folgt die Rekursion symbolischen Verknüpfungen, ansonsten werden die Verknüpfungen als solche beibehalten. Zum Kopieren regulärer Dateien wird *copy-file* aufgerufen. Wenn *keep-mtime?* wahr ist, bleibt der Zeitstempel der letzten Änderung an den Dateien in *Quelle* dabei bei denen am *Zielort* erhalten. Wenn *keep-permissions?* wahr ist, bleiben Dateiberechtigungen erhalten. Ein ausführliches Protokoll wird in den bei *log* angegebenen Port geschrieben.

`delete-file-recursively` *Verzeichnis* [*#:follow-mounts?* [Scheme-Prozedur]
#f] *Das* `Verzeichnis` *rekursiv löschen, wie bei*
`rm -rf`, ohne symbolischen Verknüpfungen zu folgen. Auch Einhängepunkten wird *nicht* gefolgt, außer falls *follow-mounts?* wahr ist. Fehler dabei werden angezeigt aber ignoriert.

`substitute*` *Datei* ((*Regexp Muster-Variable...*) *Rumpf...*) [Scheme-Syntax]
... Den regulären
Ausdruck *Regexp* in der *Datei* durch die durch *Rumpf* berechnete Zeichenkette ersetzen. Bei der Auswertung von *Rumpf* wird jede *Muster-Variable* an den Teilausdruck an der entsprechenden Position der Regexp gebunden. Zum Beispiel:

```
(substitute* file
  (("Hallo")
   "Guten Morgen\n")
  (("foo([a-z]+)bar(.*)$" alles Buchstaben Ende)
   (string-append "baz" Buchstaben Ende)))
```

Jedes Mal, wenn eine Zeile in der *Datei* den Text `Hallo` enthält, wird dieser durch `Guten Morgen` ersetzt. Jedes Mal, wenn eine Zeile zum zweiten regulären Ausdruck passt, wird `alles` an die vollständige Übereinstimmung gebunden, `Buchstaben` wird an den ersten Teilausdruck gebunden und `Ende` an den letzten.

Wird für eine *Muster-Variable* nur `_` geschrieben, so wird keine Variable an die Teilzeichenkette an der entsprechenden Position im Muster gebunden.

Alternativ kann statt einer *Datei* auch eine Liste von Dateinamen angegeben werden. In diesem Fall wird jede davon den Substitutionen unterzogen.

Seien Sie vorsichtig bei der Nutzung von `$`, um auf das Ende einer Zeile zu passen. `$` passt nämlich *nicht* auf den Zeilenumbruch am Ende einer Zeile.

9.7.4 Dateien suchen

Dieser Abschnitt beschreibt Prozeduren, um Dateien zu suchen und zu filtern.

`file-name-predicate` *Regexp* [Scheme-Prozedur]
: Liefert ein Prädikat, das gegeben einen Dateinamen, dessen Basisnamen auf *Regexp* passt, wahr liefert.

`find-files` *Verzeichnis* [*Prädikat*] [*#:stat lstat*] [*#:directories? #f*] [*#:fail-on-error? #f*] *Liefert die* [Scheme-Prozedur]
: lexikografisch sortierte Liste der Dateien innerhalb *Verzeichnis*, für die das *Prädikat* wahr liefert. An *Prädikat* werden zwei Argumente übergeben: Der absolute Dateiname und der zugehörige Stat-Puffer. Das vorgegebene Prädikat liefert immer wahr. Als *Prädikat* kann auch ein regulärer Ausdruck benutzt werden; in diesem Fall ist er äquivalent zu `(file-name-predicate` *Prädikat*`)`. Mit *stat* werden Informationen über die Datei ermittelt; wenn dafür `lstat` benutzt wird, bedeutet das, dass symbolische Verknüpfungen nicht verfolgt werden. Wenn *directories?* wahr ist, dann werden auch Verzeichnisse aufgezählt. Wenn *fail-on-error?* wahr ist, dann wird bei einem Fehler eine Ausnahme ausgelöst.

Nun folgen ein paar Beispiele, wobei wir annehmen, dass das aktuelle Verzeichnis der Wurzel des Guix-Quellbaums entspricht.

```
;; Alle regulären Dateien im aktuellen Verzeichnis auflisten.
(find-files ".")
⇒ ("./.dir-locals.el" "./.gitignore" ...)

;; Alle .scm-Dateien unter gnu/services auflisten.
(find-files "gnu/services" "\\.scm$")
⇒ ("gnu/services/admin.scm" "gnu/services/audio.scm" ...)

;; ar-Dateien im aktuellen Verzeichnis auflisten.
(find-files "." (lambda (file stat) (ar-file? file)))
⇒ ("./libformat.a" "./libstore.a" ...)
```

`which` *Programm* [Scheme-Prozedur]
: Liefert den vollständigen Dateinamen für das *Programm*, der in `$PATH` gesucht wird, oder `#f`, wenn das *Programm* nicht gefunden werden konnte.

`search-input-file` *Eingaben Name* [Scheme-Prozedur]
`search-input-directory` *Eingaben Name* [Scheme-Prozedur]
: Liefert den vollständigen Dateinamen von *Name*, das in allen *Eingaben* gesucht wird. `search-input-file` sucht nach einer regulären Datei, während `search-input-directory` nach einem Verzeichnis sucht. Wenn der *Name* nicht vorkommt, wird eine Ausnahme ausgelöst.

  Hierbei muss für *Eingaben* eine assoziative Liste wie `inputs` oder `native-inputs` übergeben werden, die für Erstellungsphasen zur Verfügung steht (siehe Abschnitt 9.6 [Erstellungsphasen], Seite 150).

Hier ist ein (vereinfachtes) Beispiel, wie `search-input-file` in einer Erstellungsphase des `wireguard-tools`-Pakets benutzt wird:

```
(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))))))
```

9.7.5 Programme aufrufen

Im Modul finden Sie Prozeduren, die geeignet sind, Prozesse zu erzeugen. Hauptsächlich handelt es sich um praktische Wrapper für Guiles `system*` (siehe Abschnitt "Processes" in *Referenzhandbuch zu GNU Guile*).

`invoke` *Programm Argumente...* [Scheme-Prozedur]
: *Programm* mit *Argumente* aufrufen. Es wird eine `&invoke-error`-Ausnahme ausgelöst, wenn der Exit-Code ungleich null ist, ansonsten wird `#t` zurückgeliefert.

  Der Vorteil gegenüber `system*` ist, dass Sie den Rückgabewert nicht zu überprüfen brauchen. So vermeiden Sie umständlichen Code in Shell-Skript-haften Schnipseln etwa in Erstellungsphasen von Paketen.

`invoke-error?` *c* [Scheme-Prozedur]
: Liefert wahr, wenn *c* ein `&invoke-error`-Zustand ist.

`invoke-error-program` *c* [Scheme-Prozedur]
`invoke-error-arguments` *c* [Scheme-Prozedur]
`invoke-error-exit-status` *c* [Scheme-Prozedur]
`invoke-error-term-signal` *c* [Scheme-Prozedur]
`invoke-error-stop-signal` *c* [Scheme-Prozedur]
: Auf bestimmte Felder von *c* zugreifen, einem `&invoke-error`-Zustand.

`report-invoke-error` *c* [*Port*] [Scheme-Prozedur]
: Auf *Port* (nach Vorgabe der `current-error-port`) eine Meldung für *c*, einem `&invoke-error`-Zustand, menschenlesbar ausgeben.

  Normalerweise würden Sie das so benutzen:

```
(use-modules (srfi srfi-34) ;für '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` *Programm Argumente...* [Scheme-Prozedur]
: *Programm* mit *Argumente* aufrufen und dabei die Standardausgabe und Standardfehlerausgabe von *Programm* einfangen. Wenn *Programm* erfolgreich ausgeführt wird,

wird *nichts* ausgegeben und der unbestimmte Wert `*unspecified*` zurückgeliefert. Andernfalls wird ein `&message`-Fehlerzustand ausgelöst, der den Status-Code und die Ausgabe von *Programm* enthält.

Hier ist ein Beispiel:

```
(use-modules (srfi srfi-34) ;für 'guard'
             (srfi srfi-35) ;für 'message-condition?'
             (guix build utils))

(guard (c ((message-condition? c)
           (display (condition-message c))))
  (invoke/quiet "date")  ;alles in Ordnung
  (invoke/quiet "date" "--imaginary-option"))

⊣ 'date --imaginary-option' exited with status 1; output follows:

    date: Unbekannte Option »--imaginary-option«
    „date --help“ liefert weitere Informationen.
```

9.7.6 Erstellungsphasen

`(guix build utils)` enthält auch Werkzeuge, um die von Erstellungssystemen benutzten Erstellungsphasen zu verändern (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). Erstellungsphasen werden durch assoziative Listen oder „Alists“ repräsentiert (siehe Abschnitt “Association Lists” in *Referenzhandbuch zu GNU Guile*), wo jeder Schlüssel ein Symbol ist, das den Namen der Phase angibt, und der assoziierte Wert eine Prozedur ist (siehe Abschnitt 9.6 [Erstellungsphasen], Seite 150).

Die zum Kern von Guile („Guile core“) gehörenden Prozeduren und das Modul `(srfi srfi-1)` stellen beide Werkzeuge zum Bearbeiten von Alists zur Verfügung. Das Modul `(guix build utils)` ergänzt sie um Werkzeuge, die speziell für Erstellungsphasen gedacht sind.

`modify-phases` *Phasen Klausel...* [Scheme-Syntax]

Die *Phasen* der Reihe nach entsprechend jeder *Klausel* ändern. Die Klauseln dürfen eine der folgenden Formen haben:

```
(delete alter-Phasenname)
(replace alter-Phasenname neue-Phase)
(add-before alter-Phasenname neuer-Phasenname neue-Phase)
(add-after alter-Phasenname neuer-Phasenname neue-Phase)
```

Jeder *Phasenname* oben ist ein Ausdruck, der zu einem Symbol auswertet, und *neue-Phase* ist ein Ausdruck, der zu einer Prozedur auswertet.

Folgendes Beispiel stammt aus der Definition des `grep`-Pakets. Es fügt eine neue Phase namens `egrep-und-fgrep-korrigieren` hinzu, die auf die `install`-Phase folgen soll. Diese Phase ist eine Prozedur (`lambda*` bedeutet, sie ist eine Prozedur ohne eigenen Namen), die ein Schlüsselwort `#:outputs` bekommt und die restlichen Schlüsselwortargumente ignoriert (siehe Abschnitt “Optional Arguments” in *Referenzhandbuch zu GNU Guile* für mehr Informationen zu `lambda*` und optionalen sowie Schlüsselwort-Argumenten). In der Phase

wird `substitute*` benutzt, um die installierten Skripte `egrep` und `fgrep` so zu verändern, dass sie `grep` anhand seines absoluten Dateinamens aufrufen:

```
(modify-phases %standard-phases
  (add-after 'install 'egrep-und-fgrep-korrigieren
    ;; 'egrep' und 'fgrep' patchen, damit diese 'grep' über den
    ;; absoluten Dateinamen ausführen, statt es in $PATH zu suchen.
    (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")))))))
```

In dem Beispiel, das nun folgt, werden Phasen auf zweierlei Art geändert: Die Standard-`configure`-Phase wird gelöscht, meistens weil das Paket über kein `configure`-Skript oder etwas Ähnliches verfügt, und die vorgegebene `install`-Phase wird durch eine ersetzt, in der die zu installierenden ausführbaren Dateien manuell kopiert werden.

```
(modify-phases %standard-phases
  (delete 'configure)      ;kein 'configure'-Skript
  (replace 'install
    (lambda* (#:key outputs #:allow-other-keys)
      ;; Das Makefile im Paket enthält kein "install"-Ziel,
      ;; also müssen wir es selber machen.
      (let ((bin (string-append (assoc-ref outputs "out")
                                "/bin")))
        (install-file "footswitch" bin)
        (install-file "scythe" bin)))))
```

9.7.7 Wrapper

Es kommt vor, dass Befehle nur richtig funktionieren, wenn bestimmte Umgebungsvariable festgelegt sind. Meistens geht es dabei um Suchpfade (siehe Abschnitt 9.8 [Suchpfade], Seite 161). Wenn man sie nicht zuweist, finden die Programme vielleicht benötigte Dateien oder andere Befehle *nicht* oder sie nehmen die „falschen“, je nachdem, in welcher Umgebung man sie ausführt. Einige Beispiele:

- ein Shell-Skript, wo erwartet wird, dass die darin benutzten Befehle in `PATH` zu finden sind,
- ein Guile-Programm, wo erwartet wird, dass dessen Module in `GUILE_LOAD_PATH` und `GUILE_LOAD_COMPILED_PATH` liegen,
- eine Qt-Anwendung, für die bestimmte Plugins in `QT_PLUGIN_PATH` erwartet werden.

Das Ziel einer Paketautorin ist, dass Befehle immer auf gleiche Weise funktionieren statt von externen Einstellungen abhängig zu sein. Eine Möglichkeit, das zu bewerkstelligen, ist, Befehle in ein dünnes *Wrapper*-Skript einzukleiden, welches diese Umgebungsvariablen festlegt und so dafür sorgt, dass solche Laufzeitabhängigkeiten sicherlich gefunden werden. Der Wrapper würde in den obigen Beispielen also benutzt, um `PATH`, `GUILE_LOAD_PATH` oder `QT_PLUGIN_PATH` festzulegen.

Das Wrappen wird erleichtert durch ein paar Hilfsprozeduren im Modul `(guix build utils)`, mit denen Sie Befehle wrappen können.

`wrap-program` *Programm* [*#:sh* `sh`] [*#:rest* `Variable`] *Einen Wrapper für* `Programm` [Scheme-Prozedur]

anlegen. Die Liste *Variable* sollte so aussehen:

```
'(Variable Trennzeichen Position Liste-von-Verzeichnissen)
```

Das *Trennzeichen* ist optional. Wenn Sie keines angeben, wird `:` genommen.

Zum Beispiel kopiert dieser Aufruf:

```
(wrap-program "foo"
              '("PATH" ":" = ("/gnu/.../bar/bin"))
              '("CERT_PATH" suffix ("/gnu/.../baz/certs"
                                    "/qux/certs")))
```

`foo` nach `.foo-real` und erzeugt die Datei `foo` mit folgendem Inhalt:

```
#!ort/mit/bin/bash
export PATH="/gnu/.../bar/bin"
export CERT_PATH="$CERT_PATH${CERT_PATH:+:}/gnu/.../baz/certs:/qux/certs"
exec -a $0 ort/mit/.foo-real "$@"
```

Wenn *Programm* bereits mit `wrap-program` gewrappt wurde, wird sein bestehender Wrapper um die Definitionen jeder Variable in der *Variable*-Liste ergänzt. Ansonsten wird ein Wrapper mit *sh* als Interpretierer angelegt.

`wrap-script` *Programm* [*#:guile* `guile`] [*#:rest* `Variable`] *Das Skript* `Programm` *in* [Scheme-Prozedur]

einen Wrapper wickeln, damit *Variable* vorher festgelegt werden. Das Format für *Variable* ist genau wie bei der Prozedur `wrap-program`. Der Unterschied zu `wrap-program` ist, dass kein getrenntes Shell-Skript erzeugt wird, sondern das *Programm* selbst abgeändert wird, indem zu Beginn von *Programm* ein Guile-Skript platziert wird. In der Sprache des Skripts wird das Guile-Skript als Kommentar interpretiert.

Besondere Kommentare zur Kodierung der Datei, wie es sie bei Python geben kann, werden auf der zweiten Zeile neu erzeugt.

Beachten Sie, dass diese Prozedur auf dieselbe Datei nur einmal angewandt werden kann, denn sie auf Guile-Skripts loszulassen wird *nicht* unterstützt.

9.8 Suchpfade

Zahlreiche Programme und Bibliotheken suchen nach Eingabedaten in einem *Suchpfad*, d.h. einer Liste von Verzeichnissen: Shells wie Bash suchen ausführbare Dateien im Befehls-Suchpfad, ein C-Compiler sucht `.h`-Dateien in seinem Header-Suchpfad, der Python-Interpretierer sucht `.py`-Dateien in seinem Suchpfad, der Rechtschreibprüfer hat einen Suchpfad für Wörterbücher und so weiter.

Suchpfade kann man für gewöhnlich über Umgebungsvariable festlegen (siehe Abschnitt "Environment Variables" in *Referenzhandbuch der GNU-C-Bibliothek*). Zum Beispiel ändern Sie die oben genannten Suchpfade, indem Sie den Wert der Umgebungsvariablen `PATH`, `C_INCLUDE_PATH`, `PYTHONPATH` (oder `GUIX_PYTHONPATH`) und `DICPATH` festlegen –

Sie wissen schon, diese Variablen, die auf `PATH` enden und wenn Sie etwas daran falsch machen, werden Dinge „nicht gefunden“.

Vielleicht ist Ihnen auf der Befehlszeile aufgefallen, dass Guix Bescheid weiß, welche Umgebungsvariablen definiert sein müssen und wie. Wenn Sie Pakete in Ihr Standardprofil installieren, wird die Datei `~/.guix-profile/etc/profile` angelegt, die Sie mit `source` in Ihre Shell übernehmen können, damit die Variablen richtig festgelegt sind. Genauso werden Ihnen, wenn Sie mit `guix shell` eine Umgebung mit Python und der Python-Bibliothek NumPy erzeugen lassen und die Befehlszeilenoption `--search-paths` angeben, die Variablen `PATH` und `GUIX_PYTHONPATH` gezeigt (siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86):

```
$ 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"
```

Wenn Sie `--search-paths` weglassen, werden diese Umgebungsvariablen direkt festgelegt, so dass Python NumPy vorfinden kann:

```
$ 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'
```

Damit das funktioniert, wurden in der Definition des `python`-Pakets der dafür nötige Suchpfad und die zugehörige Umgebungsvariable `GUIX_PYTHONPATH` *deklariert*. Das sieht so aus:

```
(package
  (name "python")
  (version "3.9.9")
  ;; davor stehen andere Felder ...
  (native-search-paths
   (list (search-path-specification
          (variable "GUIX_PYTHONPATH")
          (files (list "lib/python/3.9/site-packages"))))))
```

Die Aussage hinter dem `native-search-paths`-Feld ist, dass wenn das `python`-Paket benutzt wird, die Umgebungsvariable `GUIX_PYTHONPATH` so definiert wird, dass alle Unterverzeichnisse `lib/python/3.9/site-packages` in seiner Umgebung enthalten sind. (Mit `native-` meinen wir, dass in einer Umgebung zur Cross-Kompilierung nur native Eingaben zum Suchpfad hinzugefügt werden dürfen; siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113.) In unserem NumPy-Beispiel oben enthält das Profil, in dem `python` auftaucht, genau ein solches Unterverzeichnis, auf das `GUIX_PYTHONPATH` festgelegt wird. Wenn es mehrere `lib/python/3.9/site-packages` gibt – etwa wenn wir über Erstellungsumgebungen reden –, dann werden alle durch Doppelpunkte (`:`) getrennt zu `GUIX_PYTHONPATH` hinzugefügt.

Anmerkung: Wir weisen darauf hin, dass `GUIX_PYTHONPATH` als Teil der Definition des `python`-Pakets spezifiziert wird und *nicht* als Teil von `python-numpy`.

Der Grund ist, dass diese Umgebungsvariable zu Python „gehört“ und nicht zu NumPy: Python ist es, das den Wert der Variablen ausliest und befolgt.

Daraus folgt, dass wenn Sie ein Profil ohne `python` erzeugen, `GUIX_PYTHONPATH` *nicht* definiert wird, selbst wenn `.py`-Dateien zum Profil gehören:

```
$ guix shell python-numpy --search-paths --pure
export PATH="/gnu/store/...-profile/bin"
```

Das ist logisch, wenn wir das Profil alleine betrachten: Keine Software im Profil würde `GUIX_PYTHONPATH` auslesen.

Selbstverständlich gibt es mehr als nur eine Sorte Suchpfad: Es gibt Pakete, die mehrere Suchpfade berücksichtigen, solche mit anderen Trennzeichen als dem Doppelpunkt, solche, die im Suchpfad gleich mehrere Verzeichnisse aufnehmen, und so weiter. Ein weiterführendes Beispiel ist der Suchpfad von libxml2: Der Wert der Umgebungsvariablen `XML_CATALOG_FILES` wird durch Leerzeichen getrennt, er muss eine Liste von `catalog.xml`-Dateien (keinen Verzeichnissen) fassen und diese sind in `xml`-Unterverzeichnissen zu finden – ganz schön anspruchsvoll. Die Suchpfadspezifikation dazu sieht so aus:

```
(package
  (name "libxml2")
  ;; davor stehen andere Felder ...
  (native-search-paths
   (list (search-path-specification
          (variable "XML_CATALOG_FILES")
          (separator " ")
          (files '("xml"))
          (file-pattern "^catalog\\.xml$")
          (file-type 'regular)))))
```

Keine Angst; die meisten Suchpfadspezifikationen sind einfacher.

Im Modul `(guix search-paths)` wird der Datentyp für Suchpfadspezifikationen definiert sowie eine Reihe von Hilfsprozeduren. Es folgt nun die Referenz der Suchpfadspezifikationen.

`search-path-specification` [Datentyp]

Der Datentyp für Suchpfadspezifikationen.

`variable` Welchen Namen die Umgebungsvariable für diesen Suchpfad trägt (als Zeichenkette).

`files` Eine Liste der Unterverzeichnisse, die zum Suchpfad hinzugefügt werden sollen.

`separator` (Vorgabe: `":"`)
Die Zeichenkette, wodurch Komponenten des Suchpfads voneinander getrennt werden.

Für den Sonderfall, dass für `separator` der Wert `#f` gewählt wird, handelt es sich um einen „Suchpfad mit nur einer Komponente“, mit anderen Worten einen Suchpfad, der höchstens ein Element enthalten darf. Das ist für Fälle gedacht wie die `SSL_CERT_DIR`-Variable (die OpenSSL,

cURL und ein paar andere Pakete beachten) oder die `ASPELL_DICT_DIR`-Variable (auf die das Rechtschreibprüfprogramm GNU Aspell achtet), welche beide auf ein einzelnes Verzeichnis zeigen müssen.

`file-type` (Vorgabe: `'directory`)
: Welche Art von Datei passt – hier kann `'directory` (für Verzeichnisse) oder `'regular` (für reguläre Dateien) angegeben werden, aber auch jedes andere Symbol, was `stat:type` zurückliefern kann (siehe Abschnitt "File System" in *Referenzhandbuch zu GNU Guile*).

  Im libxml2-Beispiel oben sind es reguläre Dateien, die wir suchen, dagegen würde beim Python-Beispiel nach Verzeichnissen gesucht.

`file-pattern` (Vorgabe: `#f`)
: Das hier muss entweder `#f` oder ein regulärer Ausdruck sein, der angibt, welche Dateien *innerhalb* der mit dem Feld `files` angegebenen Unterverzeichnisse darauf passen.

  Auch hier sehen Sie im libxml2-Beispiel eine Situation, wo dies gebraucht wird.

Manche Suchpfade sind an mehr als ein Paket gekoppelt. Um sie nicht doppelt und dreifach zu spezifizieren, sind manche davon vordefiniert in `(guix search-paths)`.

`$SSL_CERT_DIR` [Scheme-Variable]
`$SSL_CERT_FILE` [Scheme-Variable]
: Mit diesen beiden Suchpfaden wird angegeben, wo X.509-Zertifikate zu finden sind (siehe Abschnitt 12.11 [X.509-Zertifikate], Seite 606).

Diese vordefinierten Suchpfade kann man wie im folgenden Beispiel benutzen:

```
(package
  (name "curl")
  ;; eigentlich stehen hier noch ein paar Felder ...
  (native-search-paths (list $SSL_CERT_DIR $SSL_CERT_FILE)))
```

Wie macht man aus Suchpfadspezifikationen einerseits und einem Haufen Verzeichnisse andererseits nun eine Menge von Definitionen für Umgebungsvariable? Das ist die Aufgabe von `evaluate-search-paths`.

`evaluate-search-paths` *`Suchpfade Verzeichnisse`* [Scheme-Prozedur]
[`getenv`] Die `Suchpfade` auswerten. Sie werden
: als Liste von Suchpfadspezifikationen übergeben und untersucht werden *Verzeichnisse*, eine Liste von Verzeichnisnamen. Das Ergebnis wird als Liste von Paaren aus Spezifikation und Wert zurückgeliefert. Wenn Sie *getenv* angeben, werden darüber die momentanen Festlegungen erfasst und nur die *nicht* bereits gültigen gemeldet.

Das Modul `(guix profiles)` enthält eine zugeschnittenere Hilfsprozedur `load-profile`, mit der Umgebungsvariable eines Profils festgelegt werden.

9.9 Der Store

Konzeptionell ist der *Store* der Ort, wo Ableitungen nach erfolgreicher Erstellung gespeichert werden – standardmäßig finden Sie ihn in `/gnu/store`. Unterverzeichnisse im Store werden *Store-Objekte* oder manchmal auch *Store-Pfade* genannt. Mit dem Store ist eine Datenbank assoziiert, die Informationen enthält wie zum Beispiel, welche Store-Pfade jeder Store-Pfad jeweils referenziert, und eine Liste, welche Store-Objekte *gültig* sind, also Ergebnisse erfolgreicher Erstellungen sind. Die Datenbank befindet sich in `localstatedir/guix/db`, wobei *localstatedir* das mit `--localstatedir` bei der Ausführung von „configure“ angegebene Zustandsverzeichnis ist, normalerweise `/var`.

Auf den Store wird *nur* durch den Daemon im Auftrag seiner Clients zugegriffen (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18). Um den Store zu verändern, verbinden sich Clients über einen Unix-Socket mit dem Daemon, senden ihm entsprechende Anfragen und lesen dann dessen Antwort – so etwas nennt sich entfernter Prozeduraufruf (englisch „Remote Procedure Call“ oder kurz RPC).

> **Anmerkung:** Benutzer dürfen *niemals* Dateien in `/gnu/store` direkt verändern, sonst wären diese nicht mehr konsistent und die Grundannahmen im funktionalen Modell von Guix, dass die Objekte unveränderlich sind, wären dahin (siehe Kapitel 1 [Einführung], Seite 1).
>
> Siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61, für Informationen, wie die Integrität des Stores überprüft und nach versehentlichen Veränderungen unter Umständen wiederhergestellt werden kann.

Das Modul `(guix store)` bietet Prozeduren an, um sich mit dem Daemon zu verbinden und entfernte Prozeduraufrufe durchzuführen. Diese werden im Folgenden beschrieben. Das vorgegebene Verhalten von `open-connection`, und daher allen `guix`-Befehlen, ist, sich mit dem lokalen Daemon oder dem an der in der Umgebungsvariablen `GUIX_DAEMON_SOCKET` angegeben URL zu verbinden.

`GUIX_DAEMON_SOCKET` [Umgebungsvariable]

Ist diese Variable gesetzt, dann sollte ihr Wert ein Dateipfad oder eine URI sein, worüber man sich mit dem Daemon verbinden kann. Ist der Wert der Pfad zu einer Datei, bezeichnet dieser einen Unix-Socket, mit dem eine Verbindung hergestellt werden soll. Ist er eine URI, so werden folgende URI-Schemata unterstützt:

`file`
`unix` Für Unix-Sockets. `file:///var/guix/daemon-socket/socket` kann gleichbedeutend auch als `/var/guix/daemon-socket/socket` angegeben werden.

`guix` Solche URIs benennen Verbindungen über TCP/IP ohne Verschlüsselung oder Authentifizierung des entfernten Rechners. Die URI muss den Hostnamen, also den Rechnernamen des entfernten Rechners, und optional eine Portnummer angeben (sonst wird als Vorgabe der Port 44146 benutzt):

```
guix://master.guix.example.org:1234
```

Diese Konfiguration ist für lokale Netzwerke wie etwa in Rechen-Clustern geeignet, wo sich nur vertrauenswürdige Knoten mit dem Erstellungs-Daemon z.B. unter `master.guix.example.org` verbinden können.

Die Befehlszeilenoption `--listen` von `guix-daemon` kann benutzt werden, damit er auf TCP-Verbindungen lauscht (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

`ssh` Mit solchen URIs kann eine Verbindung zu einem entfernten Daemon über SSH hergestellt werden. Diese Funktionalität setzt Guile-SSH voraus (siehe Abschnitt 2.2 [Voraussetzungen], Seite 8) sowie eine funktionierende `guile`-Binärdatei, deren Ort im `PATH` der Zielmaschine eingetragen ist. Authentisierung über einen öffentlichen Schlüssel oder GSSAPI ist möglich. Eine typische URL sieht so aus:

```
ssh://charlie@guix.example.org:22
```

Was `guix copy` betrifft, richtet es sich nach den üblichen OpenSSH-Client-Konfigurationsdateien (siehe Abschnitt 10.13 [Aufruf von guix copy], Seite 241).

In Zukunft könnten weitere URI-Schemata unterstützt werden.

> **Anmerkung:** Die Fähigkeit, sich mit entfernten Erstellungs-Daemons zu verbinden, sehen wir als experimentell an, Stand 1.4.0. Bitte diskutieren Sie mit uns jegliche Probleme oder Vorschläge, die Sie haben könnten (siehe Kapitel 22 [Mitwirken], Seite 708).

`open-connection` [*Uri*] [*#:reserve-space? #t*] [Scheme-Prozedur]
Sich mit dem Daemon über den Unix-Socket an *Uri* verbinden (einer Zeichenkette). Wenn *reserve-space?* wahr ist, lässt ihn das etwas zusätzlichen Speicher im Dateisystem reservieren, damit der Müllsammler auch dann noch funktioniert, wenn die Platte zu voll wird. Liefert ein Server-Objekt.

Uri nimmt standardmäßig den Wert von `%default-socket-path` an, was dem bei der Installation mit dem Aufruf von `configure` ausgewählten Vorgabeort entspricht, gemäß den Befehlszeilenoptionen, mit denen `configure` aufgerufen wurde.

`close-connection` *Server* [Scheme-Prozedur]
Die Verbindung zum *Server* trennen.

`current-build-output-port` [Scheme-Variable]
Diese Variable ist an einen SRFI-39-Parameter gebunden, der auf den Scheme-Port verweist, an den vom Daemon empfangene Erstellungsprotokolle und Fehlerprotokolle geschrieben werden sollen.

Prozeduren, die entfernte Prozeduraufrufe durchführen, nehmen immer ein Server-Objekt als ihr erstes Argument.

`valid-path?` *Server Pfad* [Scheme-Prozedur]
Liefert `#t`, wenn der *Pfad* ein gültiges Store-Objekt benennt, und sonst `#f` (ein ungültiges Objekt kann auf der Platte gespeichert sein, tatsächlich aber ungültig sein, zum Beispiel weil es das Ergebnis einer abgebrochenen oder fehlgeschlagenen Erstellung ist).

Ein `&store-protocol-error`-Fehlerzustand wird ausgelöst, wenn der *Pfad* nicht mit dem Store-Verzeichnis als Präfix beginnt (`/gnu/store`).

`add-text-to-store` *`Server Name Text`* [*`Referenzen`*] [Scheme-Prozedur]
: Den *Text* im Store in einer Datei namens *Name* ablegen und ihren Store-Pfad zurückliefern. *Referenzen* ist die Liste der Store-Pfade, die der Store-Pfad dann referenzieren soll.

`build-derivations` *`Store Ableitungen`* [*`Modus`*] *Die* *`Ableitungen`* *erstellen* (*eine Liste von* `<derivation>`-Objekten, `.drv`-Dateinamen oder Paaren aus je Ableitung und Ausgabe. Dabei gilt der angegebene *Modus* – vorgegeben ist `(build-mode normal)`. [Scheme-Prozedur]

Es sei erwähnt, dass im Modul `(guix monads)` eine Monade sowie monadische Versionen obiger Prozeduren angeboten werden, damit an Code, der auf den Store zugreift, bequemer gearbeitet werden kann (siehe Abschnitt 9.11 [Die Store-Monade], Seite 170).

Dieser Abschnitt ist im Moment noch unvollständig.

9.10 Ableitungen

Systemnahe Erstellungsaktionen sowie die Umgebung, in der selbige durchzuführen sind, werden durch *Ableitungen* dargestellt. Eine Ableitung enthält folgende Informationen:

- Die Ausgaben, die die Ableitung hat. Ableitungen erzeugen mindestens eine Datei bzw. ein Verzeichnis im Store, können aber auch mehrere erzeugen.
- Die Eingaben der Ableitung, also Abhängigkeiten zur Zeit ihrer Erstellung, die entweder andere Ableitungen oder einfache Dateien im Store sind (wie Patches, Erstellungsskripts usw.).
- Das System, wofür mit der Ableitung erstellt wird, also ihr Ziel – z.B. `x86_64-linux`.
- Der Dateiname eines Erstellungsskripts im Store, zusammen mit den Argumenten, mit denen es aufgerufen werden soll.
- Eine Liste zu definierender Umgebungsvariabler.

Ableitungen ermöglichen es den Clients des Daemons, diesem Erstellungsaktionen für den Store mitzuteilen. Es gibt davon zwei Arten, sowohl Darstellungen im Arbeitsspeicher jeweils für Client und Daemon als auch Dateien im Store, deren Namen auf `.drv` enden – diese Dateien werden als *Ableitungspfade* bezeichnet. Ableitungspfade können an die Prozedur `build-derivations` übergeben werden, damit die darin niedergeschriebenen Erstellungsaktionen durchgeführt werden (siehe Abschnitt 9.9 [Der Store], Seite 165).

Operationen wie das Herunterladen von Dateien und Checkouts von unter Versionskontrolle stehenden Quelldateien, bei denen der Hash des Inhalts im Voraus bekannt ist, werden als *Ableitungen mit fester Ausgabe* modelliert. Anders als reguläre Ableitungen sind die Ausgaben von Ableitungen mit fester Ausgabe unabhängig von ihren Eingaben – z.B. liefert das Herunterladen desselben Quellcodes dasselbe Ergebnis unabhängig davon, mit welcher Methode und welchen Werkzeugen er heruntergeladen wurde.

Den Ausgaben von Ableitungen – d.h. Erstellungergebnissen – ist eine Liste von *Referenzen* zugeordnet, die auch der entfernte Prozeduraufruf `references` oder der Befehl `guix gc --references` liefert (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61). Referenzen sind die Menge der Laufzeitabhängigkeiten von Erstellungsergebnissen. Referenzen sind eine Teilmenge der Eingaben von Ableitungen; die Teilmenge wird automatisch ermittelt, indem der Erstellungsdaemon alle Dateien unter den Ausgaben nach Referenzen durchsucht.

Das Modul `(guix derivations)` stellt eine Repräsentation von Ableitungen als Scheme-Objekte zur Verfügung, zusammen mit Prozeduren, um Ableitungen zu erzeugen und zu manipulieren. Die am wenigsten abstrahierte Methode, eine Ableitung zu erzeugen, ist mit der Prozedur `derivation`:

`derivation` *Store Name Ersteller Argumente* [*#: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 '()] [Scheme-Prozedur]

Eine Ableitungen mit den *Argumenten* erstellen und das resultierende `<derivation>`-Objekt liefern.

Wurden *hash* und *hash-algo* angegeben, wird eine *Ableitung mit fester Ausgabe* erzeugt – d.h. eine, deren Ausgabe schon im Voraus bekannt ist, wie z.B. beim Herunterladen einer Datei. Wenn des Weiteren auch *recursive?* wahr ist, darf die Ableitung mit fester Ausgabe eine ausführbare Datei oder ein Verzeichnis sein und *hash* muss die Prüfsumme eines Archivs mit dieser Ausgabe sein.

Ist *references-graphs* wahr, dann muss es eine Liste von Paaren aus je einem Dateinamen und einem Store-Pfad sein. In diesem Fall wird der Referenzengraph jedes Store-Pfads in einer Datei mit dem angegebenen Namen in der Erstellungsumgebung zugänglich gemacht, in einem einfachen Text-Format.

Ist *allowed-references* ein wahr, muss es eine Liste von Store-Objekten oder Ausgaben sein, die die Ausgabe der Ableitung referenzieren darf. Ebenso muss *disallowed-references*, wenn es auf wahr gesetzt ist, eine Liste von Dingen bezeichnen, die die Ausgaben *nicht* referenzieren dürfen.

Ist *leaked-env-vars* wahr, muss es eine Liste von Zeichenketten sein, die Umgebungsvariable benennen, die aus der Umgebung des Daemons in die Erstellungsumgebung überlaufen – ein „Leck“, englisch „leak“. Dies kann nur in Ableitungen mit fester Ausgabe benutzt werden, also wenn *hash* wahr ist. So ein Leck kann zum Beispiel benutzt werden, um Variable wie `http_proxy` an Ableitungen zu übergeben, die darüber Dateien herunterladen.

Ist *local-build?* wahr, wird die Ableitung als schlechter Kandidat für das Auslagern deklariert, der besser lokal erstellt werden sollte (siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13). Dies betrifft kleine Ableitungen, wo das Übertragen der Daten aufwendiger als ihre Erstellung ist.

Ist *substitutable?* falsch, wird deklariert, dass für die Ausgabe der Ableitung keine Substitute benutzt werden sollen (siehe Abschnitt 6.3 [Substitute], Seite 56). Das ist nützlich, wenn Pakete erstellt werden, die Details über den Prozessorbefehlssatz des Wirtssystems auslesen.

properties muss eine assoziative Liste enthalten, die „Eigenschaften“ der Ableitungen beschreibt. Sie wird genau so, wie sie ist, in der Ableitung gespeichert.

Hier ist ein Beispiel mit einem Shell-Skript, das als Ersteller benutzt wird. Es wird angenommen, dass *Store* eine offene Verbindung zum Daemon ist und *bash* auf eine ausführbare Bash im Store verweist:

```
(use-modules (guix utils)
```

```
              (guix store)
              (guix derivations))

(let ((builder   ; das Ersteller-Bash-Skript in den Store einfügen
       (add-text-to-store store "my-builder.sh"
                          "echo Hallo Welt > $out\n" '())))
  (derivation store "foo"
              bash `("-e" ,builder)
              #:inputs `((,bash) (,builder))
              #:env-vars '(("HOME" . "/homeless"))))
⇒ #<derivation /gnu/store/...-foo.drv => /gnu/store/...-foo>
```

Wie man sehen kann, ist es umständlich, diese grundlegende Methode direkt zu benutzen. Natürlich ist es besser, Erstellungsskripts in Scheme zu schreiben! Am besten schreibt man den Erstellungscode als „G-Ausdruck“ und übergibt ihn an `gexp->derivation`. Mehr Informationen finden Sie im Abschnitt Abschnitt 9.12 [G-Ausdrücke], Seite 175.

Doch es gab einmal eine Zeit, zu der `gexp->derivation` noch nicht existiert hatte und wo das Zusammenstellen von Ableitungen mit Scheme-Erstellungscode noch mit `build-expression->derivation` bewerkstelligt wurde, was im Folgenden beschrieben wird. Diese Prozedur gilt als veraltet und man sollte nunmehr die viel schönere Prozedur `gexp->derivation` benutzen.

`build-expression->derivation` *Store Name Ausdruck* [Scheme-Prozedur]
[*#: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] Liefert eine Ableitung, die den Scheme-Ausdruck *Ausdruck* als Ersteller einer Ableitung namens *Name* ausführt. *inputs* muss die Liste der Eingaben enthalten, jeweils als Tupel `(Name Ableitungspfad Unterableitung)`; wird keine *Unterableitung* angegeben, wird `"out"` angenommen. *Module* ist eine Liste der Namen von Guile-Modulen im momentanen Suchpfad, die in den Store kopiert, kompiliert und zur Verfügung gestellt werden, wenn der *Ausdruck* ausgeführt wird – z.B. `((guix build utils) (guix build gnu-build-system))`.

Der *Ausdruck* wird in einer Umgebung ausgewertet, in der `%outputs` an eine Liste von Ausgabe-/Pfad-Paaren gebunden wurde und in der `%build-inputs` an eine Liste von Zeichenkette-/Ausgabepfad-Paaren gebunden wurde, die aus den *inputs*-Eingaben konstruiert worden ist. Optional kann in *env-vars* eine Liste von Paaren aus Zeichenketten stehen, die Name und Wert von für den Ersteller sichtbaren Umgebungsvariablen angeben. Der Ersteller terminiert, indem er `exit` mit dem Ergebnis des *Ausdruck*s aufruft; wenn also der *Ausdruck* den Wert `#f` liefert, wird angenommen, dass die Erstellung fehlgeschlagen ist.

Ausdruck wird mit einer Ableitung *guile-for-build* erstellt. Wird kein *guile-for-build* angegeben oder steht es auf `#f`, wird stattdessen der Wert der Fluiden `%guile-for-build` benutzt.

Siehe die Erklärungen zur Prozedur `derivation` für die Bedeutung von *references-graphs*, *allowed-references*, *disallowed-references*, *local-build?* und *substitutable?*.

Hier ist ein Beispiel einer Ableitung mit nur einer Ausgabe, die ein Verzeichnis erzeugt, in dem eine einzelne Datei enthalten ist:

```
(let ((builder '(let ((out (assoc-ref %outputs "out")))
                  (mkdir out)    ; das Verzeichnis
                                 ; /gnu/store/...-goo erstellen
                  (call-with-output-file (string-append out "/test")
                    (lambda (p)
                      (display '(Hallo Guix) p))))))
  (build-expression->derivation store "goo" builder))

⇒ #<derivation /gnu/store/...-goo.drv => ...>
```

9.11 Die Store-Monade

Die auf dem Store arbeitenden Prozeduren, die in den vorigen Abschnitten beschrieben wurden, nehmen alle eine offene Verbindung zum Erstellungs-Daemon als ihr erstes Argument entgegen. Obwohl das ihnen zu Grunde liegende Modell funktional ist, weisen sie doch alle Nebenwirkungen auf oder hängen vom momentanen Zustand des Stores ab.

Ersteres ist umständlich, weil die Verbindung zum Erstellungs-Daemon zwischen all diesen Funktionen durchgereicht werden muss, so dass eine Komposition mit Funktionen ohne diesen Parameter unmöglich wird. Letzteres kann problematisch sein, weil Operationen auf dem Store Nebenwirkungen und/oder Abhängigkeiten von externem Zustand haben und ihre Ausführungsreihenfolge deswegen eine Rolle spielt.

Hier kommt das Modul `(guix monads)` ins Spiel. Im Rahmen dieses Moduls können *Monaden* benutzt werden und dazu gehört insbesondere eine für unsere Zwecke sehr nützliche Monade, die *Store-Monade*. Monaden sind ein Konstrukt, mit dem zwei Dinge möglich sind: eine Assoziation von Werten mit einem „Kontext“ (in unserem Fall ist das die Verbindung zum Store) und das Festlegen einer Reihenfolge für Berechnungen (hiermit sind auch Zugriffe auf den Store gemeint). Werte in einer Monade – solche, die mit weiterem Kontext assoziiert sind – werden *monadische Werte* genannt; Prozeduren, die solche Werte liefern, heißen *monadische Prozeduren*.

Betrachten Sie folgende „normale“ Prozedur:

```
(define (sh-symlink store)
  ;; Eine Ableitung liefern, die mit der ausführbaren Datei „bash“
  ;; symbolisch verknüpft.
  (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))))
```

Unter Verwendung von `(guix monads)` und `(guix gexp)` lässt sie sich als monadische Funktion aufschreiben:

```
(define (sh-symlink)
  ;; Ebenso, liefert aber einen monadischen Wert.
  (mlet %store-monad ((drv (package->derivation bash)))
    (gexp->derivation "sh"
```

```
                    #~(symlink (string-append #$drv "/bin/bash")
                               #$output))))
```

An der zweiten Version lassen sich mehrere Dinge beobachten: Der Parameter `Store` ist jetzt implizit geworden und wurde in die Aufrufe der monadischen Prozeduren `package->derivation` und `gexp->derivation` „eingefädelt“ und der von `package->derivation` gelieferte monadische Wert wurde mit `mlet` statt einem einfachen `let` *gebunden*.

Wie sich herausstellt, muss man den Aufruf von `package->derivation` nicht einmal aufschreiben, weil er implizit geschieht, wie wir später sehen werden (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175):

```
(define (sh-symlink)
  (gexp->derivation "sh"
                    #~(symlink (string-append #$bash "/bin/bash")
                               #$output)))
```

Die monadische `sh-symlink` einfach aufzurufen, bewirkt nichts. Wie jemand einst sagte: „Mit einer Monade geht man um, wie mit Gefangenen, gegen die man keine Beweise hat: Man muss sie laufen lassen.“ Um also aus der Monade auszubrechen und die gewünschte Wirkung zu erzielen, muss man `run-with-store` benutzen:

```
(run-with-store (open-connection) (sh-symlink))
⇒ /gnu/store/...-sh-symlink
```

Erwähnenswert ist, dass das Modul `(guix monad-repl)` die REPL von Guile um neue „Befehle“ erweitert, mit denen es leichter ist, mit monadischen Prozeduren umzugehen: `run-in-store` und `enter-store-monad` (siehe Abschnitt 9.14 [Interaktiv mit Guix arbeiten], Seite 186). Mit Ersterer wird ein einzelner monadischer Wert durch den Store „laufen gelassen“:

```
scheme@(guile-user)> ,run-in-store (package->derivation hello)
$1 = #<derivation /gnu/store/...-hello-2.9.drv => ...>
```

Mit Letzterer wird rekursiv eine weitere REPL betreten, in der alle Rückgabewerte automatisch durch den Store laufen gelassen werden:

```
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" "Hallo!")
$3 = "/gnu/store/...-foo"
store-monad@(guile-user) [1]> ,q
scheme@(guile-user)>
```

Beachten Sie, dass in einer `store-monad`-REPL keine nicht-monadischen Werte zurückgeliefert werden können.

Es gibt noch andere Meta-Befehle auf der REPL, so etwa `,build`, womit ein dateiartiges Objekt erstellt wird (siehe Abschnitt 9.14 [Interaktiv mit Guix arbeiten], Seite 186).

Die wichtigsten syntaktischen Formen, um mit Monaden im Allgemeinen umzugehen, werden im Modul `(guix monads)` bereitgestellt und sind im Folgenden beschrieben.

`with-monad` *Monade Rumpf ...* [Scheme-Syntax]

Alle `>>=`- oder `return`-Formen im *Rumpf* in der *Monade* auswerten.

`return` *Wert* [Scheme-Syntax]

Einen monadischen Wert liefern, der den übergebenen *Wert* kapselt.

`>>=` *mWert mProz ...* [Scheme-Syntax]

Den monadischen Wert *mWert binden*, wobei sein „Inhalt“ an die monadischen Prozeduren *mProz*... übergeben wird[4]. Es kann eine einzelne *mProz* oder mehrere davon geben, wie in diesem Beispiel:

```
(run-with-state
    (with-monad %state-monad
      (>>= (return 1)
           (lambda (x) (return (+ 1 x)))
           (lambda (x) (return (* 2 x)))))
  'irgendein-Zustand)

⇒ 4
⇒ irgendein-Zustand
```

`mlet` *Monade* ((*Variable mWert*) ...) *Rumpf ...* [Scheme-Syntax]

`mlet*` *Monade* ((*Variable mWert*) ...) *Rumpf ...* *Die* `Variablen` *an die monadischen Werte* `mWert` *im* [Scheme-Syntax]

Rumpf binden, der eine Folge von Ausdrücken ist. Wie beim bind-Operator kann man es sich vorstellen als „Auspacken“ des rohen, nicht-monadischen Werts, der im *mWert* steckt, wobei anschließend dieser rohe, nicht-monadische Wert im Sichtbarkeitsbereich des *Rumpf*s von der *Variable*n bezeichnet wird. Die Form (*Variable* `->` *Wert*) bindet die *Variable* an den „normalen“ *Wert*, wie es `let` tun würde. Die Bindungsoperation geschieht in der Reihenfolge von links nach rechts. Der letzte Ausdruck des *Rumpfs* muss ein monadischer Ausdruck sein und dessen Ergebnis wird das Ergebnis von `mlet` oder `mlet*` werden, wenn es durch die *Monad* laufen gelassen wurde.

`mlet*` verhält sich gegenüber `mlet` wie `let*` gegenüber `let` (siehe Abschnitt “Local Bindings” in *Referenzhandbuch zu GNU Guile*).

`mbegin` *Monade mAusdruck ...* [Scheme-System]

Der Reihe nach den *mAusdruck* und die nachfolgenden monadischen Ausdrücke binden und als Ergebnis das des letzten Ausdrucks liefern. Jeder Ausdruck in der Abfolge muss ein monadischer Ausdruck sein.

Dies verhält sich ähnlich wie `mlet`, außer dass die Rückgabewerte der monadischen Prozeduren ignoriert werden. In diesem Sinn verhält es sich analog zu `begin`, nur auf monadischen Ausdrücken.

`mwhen` *Bedingung mAusdr0 mAusdr* ...* [Scheme-System]

Wenn die *Bedingung* wahr ist, wird die Folge monadischer Ausdrücke *mAusdr0..mAusdr** wie bei `mbegin` ausgewertet. Wenn die *Bedingung* falsch ist, wird `*unspecified*` („unbestimmt“) in der momentanen Monade zurückgeliefert. Jeder Ausdruck in der Folge muss ein monadischer Ausdruck sein.

[4] Diese Operation wird gemeinhin „bind“ genannt, aber mit diesem Begriff wird in Guile eine völlig andere Prozedur bezeichnet, die nichts damit zu tun hat. Also benutzen wir dieses etwas kryptische Symbol als Erbe der Haskell-Programmiersprache.

`munless` *Bedingung mAusdr0 mAusdr* ...* [Scheme-System]
: Wenn die *Bedingung* falsch ist, wird die Folge monadischer Ausdrücke *mAusdr0..mAusdr** wie bei `mbegin` ausgewertet. Wenn die *Bedingung* wahr ist, wird `*unspecified*` („unbestimmt“) in der momentanen Monade zurückgeliefert. Jeder Ausdruck in der Folge muss ein monadischer Ausdruck sein.

Das Modul `(guix monads)` macht die *Zustandsmonade* (englisch „state monad“) verfügbar, mit der ein zusätzlicher Wert – der Zustand – durch die monadischen Prozeduraufrufe *gefädelt* werden kann.

`%state-monad` [Scheme-Variable]
: Die Zustandsmonade. Prozeduren in der Zustandsmonade können auf den gefädelten Zustand zugreifen und ihn verändern.

  Betrachten Sie das folgende Beispiel. Die Prozedur `Quadrat` liefert einen Wert in der Zustandsmonade zurück. Sie liefert das Quadrat ihres Arguments, aber sie inkrementiert auch den momentanen Zustandswert:

```
(define (Quadrat x)
  (mlet %state-monad ((Anzahl (current-state)))
    (mbegin %state-monad
      (set-current-state (+ 1 Anzahl))
      (return (* x x)))))

(run-with-state (sequence %state-monad (map Quadrat (iota 3))) 0)
⇒ (0 1 4)
⇒ 3
```

  Wird das „durch“ die Zustandsmonade `%state-monad` laufen gelassen, erhalten wir jenen zusätzlichen Zustandswert, der der Anzahl der Aufrufe von `Quadrat` entspricht.

`current-state` [Monadische Prozedur]
: Liefert den momentanen Zustand als einen monadischen Wert.

`set-current-state` *Wert* [Monadische Prozedur]
: Setzt den momentanen Zustand auf *Wert* und liefert den vorherigen Zustand als einen monadischen Wert.

`state-push` *Wert* [Monadische Prozedur]
: Hängt den *Wert* vorne an den momentanen Zustand an, der eine Liste sein muss. Liefert den vorherigen Zustand als monadischen Wert.

`state-pop` [Monadische Prozedur]
: Entfernt einen Wert vorne vom momentanen Zustand und liefert ihn als monadischen Wert zurück. Dabei wird angenommen, dass es sich beim Zustand um eine Liste handelt.

`run-with-state` *mWert* [*Zustand*] [Scheme-Prozedur]
: Den monadischen Wert *mWert* mit *Zustand* als initialem Zustand laufen lassen. Dies liefert zwei Werte: den Ergebniswert und den Ergebniszustand.

Die zentrale Schnittstelle zur Store-Monade, wie sie vom Modul `(guix store)` angeboten wird, ist die Folgende:

`%store-monad` [Scheme-Variable]

Die Store-Monade – ein anderer Name für `%state-monad`.

Werte in der Store-Monade kapseln Zugriffe auf den Store. Sobald ihre Wirkung gebraucht wird, muss ein Wert der Store-Monade „ausgewertet“ werden, indem er an die Prozedur `run-with-store` übergeben wird (siehe unten).

`run-with-store` *Store mWert* [*#:guile-for-build*] [*#:system* (*%current-system*)] [Scheme-Prozedur]

Den *mWert*, einen monadischen Wert in der Store-Monade, in der offenen Verbindung *Store* laufen lassen.

`text-file` *Name Text* [*Referenzen*] [Monadische Prozedur]

Als monadischen Wert den absoluten Dateinamen im Store für eine Datei liefern, deren Inhalt der der Zeichenkette *Text* ist. *Referenzen* ist dabei eine Liste von Store-Objekten, die die Ergebnis-Textdatei referenzieren wird; der Vorgabewert ist die leere Liste.

`binary-file` *Name Daten* [*Referenzen*] [Monadische Prozedur]

Den absoluten Dateinamen im Store als monadischen Wert für eine Datei liefern, deren Inhalt der des Byte-Vektors *Daten* ist. *Referenzen* ist dabei eine Liste von Store-Objekten, die die Ergebnis-Binärdatei referenzieren wird; der Vorgabewert ist die leere Liste.

`interned-file` *Datei* [*Name*] [*#:recursive? #t*] [*#:select?* (*const #t*)] [Monadische Prozedur]

Liefert den Namen der `Datei`,

nachdem sie in den Store interniert wurde. Dabei wird der *Name* als ihr Store-Name verwendet, oder, wenn kein *Name* angegeben wurde, der Basisname der *Datei*.

Ist *recursive?* wahr, werden in der *Datei* enthaltene Dateien rekursiv hinzugefügt; ist die *Datei* eine flache Datei und *recursive?* ist wahr, wird ihr Inhalt in den Store eingelagert und ihre Berechtigungs-Bits übernommen.

Steht *recursive?* auf wahr, wird (`select?` *Datei Stat*) für jeden Verzeichniseintrag aufgerufen, wobei *Datei* der absolute Dateiname und *Stat* das Ergebnis von `lstat` ist, außer auf den Einträgen, wo *select?* keinen wahren Wert liefert.

Folgendes Beispiel fügt eine Datei unter zwei verschiedenen Namen in den Store ein:

```
(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")
```

Das Modul (`guix packages`) exportiert die folgenden paketbezogenen monadischen Prozeduren:

`package-file` *Paket* [*Datei*] [*#:system* (*%current-system*)] [*#:target #f*] [*#:output* "*out*"] [Monadische Prozedur]

Liefert als monadischen Wert den absoluten Dateinamen der *Datei* innerhalb des Ausgabeverzeichnisses *output* des *Pakets*. Wird keine *Datei* angegeben, wird der Name des Aus-

gabeverzeichnisses *output* für das *Paket* zurückgeliefert. Ist *target* wahr, wird sein Wert als das Zielsystem bezeichnendes Tripel zum Cross-Kompilieren benutzt.

Beachten Sie, dass durch diese Prozedur das *Paket nicht* erstellt wird, also muss ihr Ergebnis keine bereits existierende Datei bezeichnen, kann aber. Wir empfehlen, diese Prozedur nur dann zu benutzen, wenn Sie wissen, was Sie tun.

`package->derivation` *Paket* [*System*] [Monadische Prozedur]
`package->cross-derivation` *Paket Ziel* [*System*] [Monadische Prozedur]
Monadische Version von `package-derivation` und `package-cross-derivation` (siehe Abschnitt 9.2 [Pakete definieren], Seite 109).

9.12 G-Ausdrücke

Es gibt also „Ableitungen“, die eine Abfolge von Erstellungsaktionen repräsentieren, die durchgeführt werden müssen, um ein Objekt im Store zu erzeugen (siehe Abschnitt 9.10 [Ableitungen], Seite 167). Diese Erstellungsaktionen werden durchgeführt, nachdem der Daemon gebeten wurde, die Ableitungen tatsächlich zu erstellen; dann führt der Daemon sie in einer isolierten Umgebung (einem sogenannten Container) aus (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

Wenig überraschend ist, dass wir diese Erstellungsaktionen gerne in Scheme schreiben würden. Wenn wir das tun, bekommen wir zwei verschiedene *Schichten* von Scheme-Code[5]: den „wirtsseitigen Code“ („host code“) – also Code, der Pakete definiert, mit dem Daemon kommuniziert etc. – und den „erstellungsseitigen Code“ („build code“) – also Code, der die Erstellungsaktionen auch wirklich umsetzt, indem Dateien erstellt werden, `make` aufgerufen wird und so weiter (siehe Abschnitt 9.6 [Erstellungsphasen], Seite 150).

Um eine Ableitung und ihre Erstellungsaktionen zu beschreiben, muss man normalerweise erstellungsseitigen Code im wirtsseitigen Code einbetten. Das bedeutet, man behandelt den erstellungsseitigen Code als Daten, was wegen der Homoikonizität von Scheme – dass Code genauso als Daten repräsentiert werden kann – sehr praktisch ist. Doch brauchen wir hier mehr als nur den normalen Quasimaskierungsmechanismus mit `quasiquote` in Scheme, wenn wir Erstellungsausdrücke konstruieren möchten.

Das Modul `(guix gexp)` implementiert *G-Ausdrücke*, eine Form von S-Ausdrücken, die zu Erstellungsausdrücken angepasst wurden. G-Ausdrücke (englisch „G-expressions“, kurz *Gexps*) setzen sich grundlegend aus drei syntaktischen Formen zusammen: `gexp`, `ungexp` und `ungexp-splicing` (alternativ einfach: `#~`, `#$` und `#$@`), die jeweils mit `quasiquote`, `unquote` und `unquote-splicing` vergleichbar sind (siehe Abschnitt “Expression Syntax” in *Referenzhandbuch zu GNU Guile*). Es gibt aber auch erhebliche Unterschiede:

- G-Ausdrücke sind dafür gedacht, in eine Datei geschrieben zu werden, wo sie von anderen Prozessen ausgeführt oder manipuliert werden können.
- Wenn ein abstraktes Objekt wie ein Paket oder eine Ableitung innerhalb eines G-Ausdrucks demaskiert wird, ist das Ergebnis davon dasselbe, wie wenn dessen Ausgabedateiname genannt worden wäre.

[5] Der Begriff *Schicht*, englisch Stratum, wurde in diesem Kontext von Manuel Serrano et al. in ihrer Arbeit an Hop geprägt. Oleg Kiselyov, der aufschlussreiche Essays und Code zu diesem Thema (`http://okmij.org/ftp/meta-programming/#meta-scheme`) geschrieben hat, nennt diese Art der Code-Generierung *Staging*, deutsch etwa Inszenierung bzw. Aufführung.

- G-Ausdrücke tragen Informationen über die Pakete oder Ableitungen mit sich, auf die sie sich beziehen, und diese Abhängigkeiten werden automatisch zu den sie benutzenden Erstellungsprozessen als Eingaben hinzugefügt.

Dieser Mechanismus ist nicht auf Pakete und Ableitung beschränkt: Es können *Compiler* definiert werden, die weitere abstrakte, hochsprachliche Objekte auf Ableitungen oder Dateien im Store „herunterbrechen“, womit diese Objekte dann auch in G-Ausdrücken eingefügt werden können. Zum Beispiel sind „dateiartige Objekte“ ein nützlicher Typ solcher abstrakter Objekte. Mit ihnen können Dateien leicht in den Store eingefügt und von Ableitungen und anderem referenziert werden (siehe unten `local-file` und `plain-file`).

Zur Veranschaulichung dieser Idee soll uns dieses Beispiel eines G-Ausdrucks dienen:

```
(define build-exp
  #~(begin
      (mkdir #$output)
      (chdir #$output)
      (symlink (string-append #$coreutils "/bin/ls")
               "list-files")))
```

Indem wir diesen G-Ausdruck an `gexp->derivation` übergeben, bekommen wir eine Ableitung, die ein Verzeichnis mit genau einer symbolischen Verknüpfung auf `/gnu/store/...-coreutils-8.22/bin/ls` erstellt:

```
(gexp->derivation "das-ding" build-exp)
```

Wie man es erwarten würde, wird die Zeichenkette `"/gnu/store/...-coreutils-8.22"` anstelle der Referenzen auf das Paket *coreutils* im eigentlichen Erstellungscode eingefügt und *coreutils* automatisch zu einer Eingabe der Ableitung gemacht. Genauso wird auch `#$output` (was äquivalent zur Schreibweise `(ungexp output)` ist) ersetzt durch eine Zeichenkette mit dem Namen der Ausgabe der Ableitung.

Im Kontext der Cross-Kompilierung bietet es sich an, zwischen Referenzen auf die *native* Erstellung eines Pakets – also der, die auf dem Wirtssystem ausgeführt werden kann – und Referenzen auf Cross-Erstellungen eines Pakets zu unterscheiden. Hierfür spielt `#+` dieselbe Rolle wie `#$`, steht aber für eine Referenz auf eine native Paketerstellung.

```
(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")
```

Im obigen Beispiel wird die native Erstellung der *coreutils* benutzt, damit `ln` tatsächlich auf dem Wirtssystem ausgeführt werden kann, aber danach die cross-kompilierte Erstellung von *emacs* referenziert.

Eine weitere Funktionalität von G-Ausdrücken stellen *importierte Module* dar. Manchmal will man bestimmte Guile-Module von der „wirtsseitigen Umgebung“ im G-Ausdruck

benutzen können, deswegen sollten diese Module in die „erstellungsseitige Umgebung“ importiert werden. Die `with-imported-modules`-Form macht das möglich:

```
(let ((build (with-imported-modules '((guix build utils))
               #~(begin
                   (use-modules (guix build utils))
                   (mkdir-p (string-append #$output "/bin"))))))
  (gexp->derivation "leeres-Verzeichnis"
                    #~(begin
                        #$build
                        (display "Erfolg!\n")
                        #t)))
```

In diesem Beispiel wird das Modul `(guix build utils)` automatisch in die isolierte Erstellungsumgebung unseres G-Ausdrucks geholt, so dass `(use-modules (guix build utils))` wie erwartet funktioniert.

Normalerweise möchten Sie, dass der *Abschluss* eines Moduls importiert wird – also das Modul und alle Module, von denen es abhängt – statt nur das Modul selbst. Ansonsten scheitern Versuche, das Modul zu benutzen, weil seine Modulabhängigkeiten fehlen. Die Prozedur `source-module-closure` berechnet den Abschluss eines Moduls, indem es den Kopf seiner Quelldatei analysiert, deswegen schafft die Prozedur hier Abhilfe:

```
(use-modules (guix modules))   ;„source-module-closure“ verfügbar machen

(with-imported-modules (source-module-closure
                         '((guix build utils)
                           (gnu build image)))
  (gexp->derivation "etwas-mit-vm"
                    #~(begin
                        (use-modules (guix build utils)
                                     (gnu build image))
                        ...)))
```

Auf die gleiche Art können Sie auch vorgehen, wenn Sie nicht bloß reine Scheme-Module importieren möchten, sondern auch „Erweiterungen“ wie Guile-Anbindungen von C-Bibliotheken oder andere „vollumfängliche“ Pakete. Sagen wir, Sie bräuchten das Paket `guile-json` auf der Erstellungsseite, dann könnten Sie es hiermit bekommen:

```
(use-modules (gnu packages guile))  ;für „guile-json“

(with-extensions (list guile-json)
  (gexp->derivation "etwas-mit-json"
                    #~(begin
                        (use-modules (json))
                        ...)))
```

Die syntaktische Form, in der G-Ausdrücke konstruiert werden, ist im Folgenden zusammengefasst.

`#~Ausdruck` [Scheme-Syntax]
`(gexp Ausdruck)` [Scheme-Syntax]

Liefert einen G-Ausdruck, der den *Ausdruck* enthält. Der *Ausdruck* kann eine oder mehrere der folgenden Formen enthalten:

`#$Objekt`
`(ungexp Objekt)`

Eine Referenz auf das *Objekt* einführen. Das *Objekt* kann einen der unterstützten Typen haben, zum Beispiel ein Paket oder eine Ableitung, so dass die `ungexp`-Form durch deren Ausgabedateiname ersetzt wird – z.B. `"/gnu/store/...-coreutils-8.22`.

Wenn das *Objekt* eine Liste ist, wird diese durchlaufen und alle unterstützten Objekte darin auf diese Weise ersetzt.

Wenn das *Objekt* ein anderer G-Ausdruck ist, wird sein Inhalt eingefügt und seine Abhängigkeiten zu denen des äußeren G-Ausdrucks hinzugefügt.

Wenn das *Objekt* eine andere Art von Objekt ist, wird es so wie es ist eingefügt.

`#$Objekt:Ausgabe`
`(ungexp Objekt Ausgabe)`

Dies verhält sich wie die Form oben, bezieht sich aber ausdrücklich auf die angegebene *Ausgabe* des *Objekt*s – dies ist nützlich, wenn das *Objekt* mehrere Ausgaben generiert (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60).

`#+Objekt`
`#+Objekt:Ausgabe`
`(ungexp-native Objekt)`
`(ungexp-native Objekt Ausgabe)`

Das Gleiche wie `ungexp`, jedoch wird im Kontext einer Cross-Kompilierung eine Referenz auf die *native* Erstellung des *Objekt*s eingefügt.

`#$output[:Ausgabe]`
`(ungexp output [Ausgabe])`

Fügt eine Referenz auf die angegebene *Ausgabe* dieser Ableitung ein, oder auf die Hauptausgabe, wenn keine *Ausgabe* angegeben wurde.

Dies ist nur bei G-Ausdrücken sinnvoll, die an `gexp->derivation` übergeben werden.

`#$@Liste`
`(ungexp-splicing Liste)`

Das Gleiche wie oben, jedoch wird nur der Inhalt der *Liste* in die äußere Liste eingespleißt.

`#+@Liste`
`(ungexp-native-splicing Liste)`

Das Gleiche, aber referenziert werden native Erstellungen der Objekte in der *Liste*.

G-Ausdrücke, die mit `gexp` oder `#~` erzeugt wurden, sind zur Laufzeit Objekte vom Typ `gexp?` (siehe unten).

`with-imported-modules` *Module Rumpf...* [Scheme-Syntax]
Markiert die in *Rumpf...* definierten G-Ausdrücke, dass sie in ihrer Ausführungsumgebung die angegebenen *Module* brauchen.

Jedes Objekt unter den *Module*n kann der Name eines Moduls wie `(guix build utils)` sein, oder es kann nacheinander ein Modulname, ein Pfeil und ein dateiartiges Objekt sein:

```
`((guix build utils)
  (guix gcrypt)
  ((guix config) => ,(scheme-file "config.scm"
                                  #~(define-module ...))))
```

Im Beispiel oben werden die ersten beiden Module vom Suchpfad genommen und das letzte aus dem angegebenen dateiartigen Objekt erzeugt.

Diese Form hat einen *lexikalischen* Sichtbarkeitsbereich: Sie wirkt sich auf die direkt in *Rumpf...* definierten G-Ausdrücke aus, aber nicht auf jene, die, sagen wir, in aus *Rumpf...* heraus aufgerufenen Prozeduren definiert wurden.

`with-extensions` *Erweiterungen Rumpf...* [Scheme-Syntax]
Markiert die in *Rumpf...* definierten G-Ausdrücke, dass sie *Erweiterungen* in ihrer Erstellungs- und Ausführungsumgebung benötigen. *Erweiterungen* sind typischerweise eine Liste von Paketobjekten wie zum Beispiel die im Modul `(gnu packages guile)` definierten.

Konkret werden die unter den *Erweiterungen* aufgeführten Pakete zum Ladepfad hinzugefügt, während die in *Rumpf...* aufgeführten importierten Module kompiliert werden und sie werden auch zum Ladepfad des von *Rumpf...* gelieferten G-Ausdrucks hinzugefügt.

`gexp?` *Objekt* [Scheme-Prozedur]
Liefert `#t`, wenn das *Objekt* ein G-Ausdruck ist.

G-Ausdrücke sind dazu gedacht, auf die Platte geschrieben zu werden, entweder als Code, der eine Ableitung erstellt, oder als einfache Dateien im Store. Die monadischen Prozeduren unten ermöglichen Ihnen das (siehe Abschnitt 9.11 [Die Store-Monade], Seite 170, wenn Sie mehr Informationen über Monaden suchen).

`gexp->derivation` *Name Ausdruck* [*#: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 *Name* "-builder")] [#:deprecation-warnings #f] [#:local-build? #f] [#:substitutable? #t] [#:properties '()] [#:guile-for-build #f] [Monadische Prozedur]
Liefert eine Ableitung unter dem *Name*n, die jeden *Ausdruck* (ein G-Ausdruck) mit *guile-for-build* (eine Ableitung) für das *System* erstellt; der *Ausdruck* wird dabei in einer Datei namens *script-name* gespeichert. Wenn „*target*“ wahr ist, wird es beim Cross-Kompilieren als Zieltripel für mit *Ausdruck* bezeichnete Pakete benutzt.

modules gilt als veraltet; stattdessen sollte `with-imported-modules` benutzt werden. Die Bedeutung ist, dass die *Module* im Ausführungskontext des *Ausdruck*s verfügbar gemacht werden; *modules* ist dabei eine Liste von Namen von Guile-Modulen, die im Modulpfad *module-path* gesucht werden, um sie in den Store zu kopieren, zu kompilieren und im Ladepfad während der Ausführung des *Ausdruck*s verfügbar zu machen – z.B. `((guix build utils) (guix build gnu-build-system))`.

effective-version bestimmt, unter welcher Zeichenkette die Erweiterungen des *Ausdruck*s zum Suchpfad hinzugefügt werden (siehe `with-extensions`) – z.B. `"2.2"`.

graft? bestimmt, ob vom *Ausdruck* benannte Pakete veredelt werden sollen, falls Veredelungen zur Verfügung stehen.

Ist *references-graphs* wahr, muss es eine Liste von Tupeln in einer der folgenden Formen sein:

```
(Dateiname Paket)
(Dateiname Paket Ausgabe)
(Dateiname Ableitung)
(Dateiname Ableitung Ausgabe)
(Dateiname Store-Objekt)
```

Bei jedem Element von *references-graphs* wird das rechts Stehende automatisch zu einer Eingabe des Erstellungsprozesses vom *Ausdruck* gemacht. In der Erstellungsumgebung enthält das, was mit *Dateiname* bezeichnet wird, den Referenzgraphen des entsprechenden Objekts in einem einfachen Textformat.

allowed-references muss entweder `#f` oder eine Liste von Ausgabenamen und Paketen sein. Eine solche Liste benennt Store-Objekte, die das Ergebnis referenzieren darf. Jede Referenz auf ein nicht dort aufgeführtes Store-Objekt löst einen Erstellungsfehler aus. Genauso funktioniert *disallowed-references*, was eine Liste von Objekten sein kann, die von den Ausgaben nicht referenziert werden dürfen.

deprecation-warnings bestimmt, ob beim Kompilieren von Modulen Warnungen angezeigt werden sollen, wenn auf als veraltet markierten Code zugegriffen wird. *deprecation-warnings* kann `#f`, `#t` oder `'detailed` (detailliert) sein.

Die anderen Argumente verhalten sich wie bei `derivation` (siehe Abschnitt 9.10 [Ableitungen], Seite 167).

Die im Folgenden erklärten Prozeduren `local-file`, `plain-file`, `computed-file`, `program-file` und `scheme-file` liefern *dateiartige Objekte*. Das bedeutet, dass diese Objekte, wenn sie in einem G-Ausdruck demaskiert werden, zu einer Datei im Store führen. Betrachten Sie zum Beispiel diesen G-Ausdruck:

```
#~(system* #$(file-append glibc "/sbin/nscd") "-f"
           #$(local-file "/tmp/my-nscd.conf"))
```

Der Effekt hiervon ist, dass `/tmp/my-nscd.conf` „interniert“ wird, indem es in den Store kopiert wird. Sobald er umgeschrieben wurde, zum Beispiel über `gexp->derivation`, referenziert der G-Ausdruck diese Kopie im `/gnu/store`. Die Datei in `/tmp` zu bearbeiten oder zu löschen, hat dann keinen Effekt mehr darauf, was der G-Ausdruck tut. `plain-file` kann in ähnlicher Weise benutzt werden, es unterscheidet sich aber darin, dass dort der Prozedur der Inhalt der Datei als eine Zeichenkette übergeben wird.

`local-file` *Datei* [*Name*] [*#:recursive? #f*] [*#:select? (const #t)*] *Liefert ein Objekt, dass die lokale* [Scheme-Prozedur]

Datei *Datei* repräsentiert und sie zum Store hinzufügen lässt; dieses Objekt kann in einem G-Ausdruck benutzt werden. Wurde für die *Datei* ein relativer Dateiname als literaler Ausdruck angegeben, wird sie relativ zur Quelldatei gesucht, in der diese Form steht. Wurde die *Datei nicht* als literale Zeichenkette angegeben, wird sie zur Laufzeit relativ zum aktuellen Arbeitsverzeichnis gesucht. Die *Datei* wird unter dem angegebenen *Name*n im Store abgelegt – als Vorgabe wird dabei der Basisname der *Datei* genommen.

Ist *recursive?* wahr, werden in der *Datei* enthaltene Dateien rekursiv hinzugefügt; ist die *Datei* eine flache Datei und *recursive?* ist wahr, wird ihr Inhalt in den Store eingelagert und ihre Berechtigungs-Bits übernommen.

Steht *recursive?* auf wahr, wird `(select? Datei Stat)` für jeden Verzeichniseintrag aufgerufen, wobei *Datei* der absolute Dateiname und *Stat* das Ergebnis von `lstat` ist, außer auf den Einträgen, wo *select?* keinen wahren Wert liefert.

Dies ist das deklarative Gegenstück zur monadischen Prozedur `interned-file` (siehe Abschnitt 9.11 [Die Store-Monade], Seite 170).

`plain-file` *Name Inhalt* [Scheme-Prozedur]

Liefert ein Objekt, das eine Textdatei mit dem angegebenen *Name*n repräsentiert, die den angegebenen *Inhalt* hat (eine Zeichenkette oder ein Bytevektor), welche zum Store hinzugefügt werden soll.

Dies ist das deklarative Gegenstück zu `text-file`.

`computed-file` *Name G-Ausdruck* [*#:local-build? #t*] [*#:options '()*] *Liefert ein Objekt, das das Store-Objekt* [Scheme-Prozedur]

mit dem *Name*n repräsentiert, eine Datei oder ein Verzeichnis, das vom *G-Ausdruck* berechnet wurde. Wenn *local-build?* auf wahr steht (wie vorgegeben), wird auf der lokalen Maschine erstellt. *options* ist eine Liste zusätzlicher Argumente, die an `gexp->derivation` übergeben werden.

Dies ist das deklarative Gegenstück zu `gexp->derivation`.

`gexp->script` *Name Ausdruck* [*#:guile (default-guile)*] [*#:module-path %load-path*] [*#:system (%current-system)*] [#:target #f] Liefert ein ausführbares Skript namens *Name*, das den *Ausdruck* mit dem angegebenen *guile* ausführt, wobei vom *Ausdruck* importierte Module in seinem Suchpfad stehen. Die Module des *Ausdruck*s werden dazu im Modulpfad *module-path* gesucht. [Monadische Prozedur]

Folgendes Beispiel erstellt ein Skript, das einfach nur den Befehl `ls` ausführt:

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

(gexp->script "list-files"
              #~(execl #$(file-append coreutils "/bin/ls")
                       "ls"))
```

Lässt man es durch den Store „laufen“ (siehe Abschnitt 9.11 [Die Store-Monade], Seite 170), erhalten wir eine Ableitung, die eine ausführbare Datei `/gnu/store/...-list-files` generiert, ungefähr so:

```
#!/gnu/store/...-guile-2.0.11/bin/guile -ds
!#
(execl "/gnu/store/...-coreutils-8.22"/bin/ls" "ls")
```

`program-file` *Name G-Ausdruck* [*#:guile #f*] [*#:module-path %load-path*] *Liefert ein Objekt, das eine* [Scheme-Prozedur]

ausführbare Store-Datei *Name* repräsentiert, die den *G-Ausdruck* ausführt. *guile* ist das zu verwendende Guile-Paket, mit dem das Skript ausgeführt werden kann. Importierte Module des *G-Ausdruck*s werden im Modulpfad *module-path* gesucht.

Dies ist das deklarative Gegenstück zu `gexp->script`.

`gexp->file` *Name G-Ausdruck* [*#:set-load-path? #t*] [*#:module-path %load-path*] [*#:splice? #f*] [*#:guile* (default-guile)] Liefert eine Ableitung, die eine Datei *Name* erstellen wird, deren Inhalt der *G-Ausdruck* ist. Ist *splice?* wahr, dann wird *G-Ausdruck* stattdessen als eine Liste von mehreren G-Ausdrücken behandelt, die alle in die resultierende Datei gespleißt werden. [Monadische Prozedur]

Ist *set-load-path?* wahr, wird in die resultierende Datei Code hinzugefügt, der den Ladepfad `%load-path` und den Ladepfad für kompilierte Dateien `%load-compiled-path` festlegt, die für die importierten Module des *G-Ausdruck*s nötig sind. Die Module des *G-Ausdruck*s werden im Modulpfad *module-path* gesucht.

Die resultierende Datei referenziert alle Abhängigkeiten des *G-Ausdruck*s oder eine Teilmenge davon.

`scheme-file` *Name Ausdruck* [*#:splice? #f*] [*#:set-load-path? #t*] *Liefert ein Objekt, das die* [Scheme-Prozedur]

Scheme-Datei *Name* mit dem *G-Ausdruck* als Inhalt repräsentiert.

Dies ist das deklarative Gegenstück zu `gexp->file`.

`text-file*` *Name Text ...* [Monadische Prozedur]

Liefert eine Ableitung als monadischen Wert, welche eine Textdatei erstellt, in der der gesamte *Text* enthalten ist. *Text* kann eine Folge nicht nur von Zeichenketten, sondern auch Objekten beliebigen Typs sein, die in einem G-Ausdruck benutzt werden können, also Paketen, Ableitungen, Objekte lokaler Dateien und so weiter. Die resultierende Store-Datei referenziert alle davon.

Diese Variante sollte gegenüber `text-file` bevorzugt verwendet werden, wann immer die zu erstellende Datei Objekte im Store referenzieren wird. Typischerweise ist das der Fall, wenn eine Konfigurationsdatei erstellt wird, die Namen von Store-Dateien enthält, so wie hier:

```
(define (profile.sh)
  ;; Liefert den Namen eines Shell-Skripts im Store,
  ;; welcher die Umgebungsvariable „PATH“ initialisiert.
  (text-file* "profile.sh"
              "export PATH=" coreutils "/bin:"
              grep "/bin:" sed "/bin\n"))
```

In diesem Beispiel wird die resultierende Datei `/gnu/store/...-profile.sh` sowohl *coreutils*, *grep* als auch *sed* referenzieren, so dass der Müllsammler diese nicht löscht, während die resultierende Datei noch lebendig ist.

`mixed-text-file` *Name Text ...* [Scheme-Prozedur]

Liefert ein Objekt, was die Store-Datei *Name* repräsentiert, die *Text* enthält. *Text* ist dabei eine Folge von Zeichenketten und dateiartigen Objekten wie zum Beispiel:

```
(mixed-text-file "profile"
                 "export PATH=" coreutils "/bin:" grep "/bin")
```

Dies ist das deklarative Gegenstück zu `text-file*`.

`file-union` *Name Dateien* [Scheme-Prozedur]

Liefert ein `<computed-file>`, das ein Verzeichnis mit allen *Dateien* enthält. Jedes Objekt in *Dateien* muss eine zweielementige Liste sein, deren erstes Element der im neuen Verzeichnis zu benutzende Dateiname ist und deren zweites Element ein G-Ausdruck ist, der die Zieldatei benennt. Hier ist ein Beispiel:

```
(file-union "etc"
            `(("hosts" ,(plain-file "hosts"
                                    "127.0.0.1 localhost"))
              ("bashrc" ,(plain-file "bashrc"
                                     "alias ls='ls --color=auto'"))))
```

Dies liefert ein Verzeichnis `etc`, das zwei Dateien enthält.

`directory-union` *Name Dinge* [Scheme-Prozedur]

Liefert ein Verzeichnis, was die Vereinigung (englisch „Union“) der *Dinge* darstellt, wobei *Dinge* eine Liste dateiartiger Objekte sein muss, die Verzeichnisse bezeichnen. Zum Beispiel:

```
(directory-union "guile+emacs" (list guile emacs))
```

Das liefert ein Verzeichnis, welches die Vereinigung der Pakete `guile` und `emacs` ist.

`file-append` *Objekt Suffix ...* [Scheme-Prozedur]

Liefert ein dateiartiges Objekt, das zur Aneinanderreihung von *Objekt* und *Suffix* umgeschrieben wird, wobei das *Objekt* ein herunterbrechbares Objekt und jedes *Suffix* eine Zeichenkette sein muss.

Betrachten Sie zum Beispiel diesen G-Ausdruck:

```
(gexp->script "uname-ausfuehren"
              #~(system* #$(file-append coreutils
                                        "/bin/uname")))
```

Denselben Effekt könnte man erreichen mit:

```
(gexp->script "uname-ausfuehren"
              #~(system* (string-append #$coreutils
                                        "/bin/uname")))
```

Es gibt jedoch einen Unterschied, nämlich enthält das resultierende Skript bei `file-append` tatsächlich den absoluten Dateinamen als Zeichenkette, während im anderen Fall das resultierende Skript einen Ausdruck `(string-append ...)` enthält, der den Dateinamen erst *zur Laufzeit* zusammensetzt.

`let-system` *System Rumpf...* [Scheme-Syntax]

`let-system` (*System Zielsystem*) *Rumpf...* [Scheme-Syntax]

System an das System binden, für das momentan erstellt wird – z.B. `"x86_64-linux"` –, während der *Rumpf* ausgeführt wird.

In der zweiten Form wird zusätzlich das *Ziel* an das aktuelle Ziel („Target“) bei der Cross-Kompilierung gebunden. Dabei handelt es sich um ein GNU-Tripel wie z.B. `"arm-linux-gnueabihf"` – oder um `#f`, wenn nicht cross-kompiliert wird.

`let-system` zu benutzen, bietet sich dann an, wenn einmal das in den G-Ausdruck gespleißte Objekt vom Zielsystem abhängen sollte, wie in diesem Beispiel:

```
#~(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 "weiß nicht!"))))
   "-net" "user" #$image)
```

`with-parameters` ((*Parameter Wert*) ...) *Ausdruck* [Scheme-Syntax]

Mit diesem Makro verhält es sich ähnlich wie mit der `parameterize`-Form für dynamisch gebundene *Parameter* (siehe Abschnitt “Parameters” in *Referenzhandbuch zu GNU Guile*). Der Hauptunterschied ist, dass es sich erst auswirkt, wenn das vom *Ausdruck* zurückgelieferte dateiartige Objekt auf eine Ableitung oder ein Store-Objekt heruntergebrochen wird.

Eine typische Anwendung von `with-parameters` ist, den für ein bestimmtes Objekt geltenden Systemtyp zwingend festzulegen:

```
(with-parameters ((%current-system "i686-linux"))
  coreutils)
```

Obiges Beispiel liefert ein Objekt, das der Erstellung von Coreutils für die i686-Architektur entspricht, egal was der aktuelle Wert von `%current-system` ist.

Natürlich gibt es zusätzlich zu in „wirtsseitigem“ Code eingebetteten G-Ausdrücken auch Module mit „erstellungsseitig“ nutzbaren Werkzeugen. Um klarzustellen, dass sie dafür gedacht sind, in der Erstellungsschicht benutzt zu werden, bleiben diese Module im Namensraum `(guix build ...)`.

Intern werden hochsprachliche, abstrakte Objekte mit ihrem Compiler entweder zu Ableitungen oder zu Store-Objekten *heruntergebrochen*. Wird zum Beispiel ein Paket heruntergebrochen, bekommt man eine Ableitung, während ein `plain-file` zu einem Store-Objekt heruntergebrochen wird. Das wird mit der monadischen Prozedur `lower-object` bewerkstelligt.

`lower-object` *Objekt* [*System*] [*#:target #f*] *Liefert die Ableitung oder das Store-Objekt, das dem* [Monadische Prozedur]

Objekt für *System* als Wert in der Store-Monade `%store-monad` entspricht, cross-kompiliert für das Zieltripel *target*, wenn *target* wahr ist. Das *Objekt* muss ein Objekt sein, für das es einen mit ihm assoziierten G-Ausdruck-Compiler gibt, wie zum Beispiel ein `<package>`.

`gexp->approximate-sexp` *G-Ausdruck* [Prozedur]

Es kann gelegentlich nützlich sein, einen G-Ausdruck in einen S-Ausdruck umzuwandeln, weil zum Beispiel manche Prüfer (siehe Abschnitt 10.8 [Aufruf von guix lint],

Seite 223) einen Blick auf die Erstellungsphasen eines Pakets werfen, um mögliche Fehler zu finden. Diese Umwandlung können Sie mit dieser Prozedur bewerkstelligen. Allerdings kann dabei manche Information verloren gehen. Genauer gesagt werden herunterbrechbare Objekte stillschweigend durch ein beliebiges Objekt ersetzt. Zurzeit ist dieses beliebige Objekt die Liste `(*approximate*)`, aber verlassen Sie sich nicht darauf, dass es so bleibt.

9.13 `guix repl` aufrufen

Der Befehl `guix repl` erleichtert es, Guix von Guile aus zu programmieren. Dazu startet er eine Guile-REPL (*Read-Eval-Print Loop*, kurz REPL, deutsch Lese-Auswerten-Schreiben-Schleife) zur interaktiven Programmierung (siehe Abschnitt "Using Guile Interactively" in *Referenzhandbuch zu GNU Guile*) oder er führt Guile-Skripts aus (siehe Abschnitt "Running Guile Scripts" in *Referenzhandbuch zu GNU Guile*). Im Vergleich dazu, einfach den Befehl `guile` aufzurufen, garantiert `guix repl`, dass alle Guix-Module und deren Abhängigkeiten im Suchpfad verfügbar sind.

Die allgemeine Syntax lautet:

```
guix repl Optionen [Datei Argumente]
```

Wird ein *Datei*-Argument angegeben, wird die angegebene *Datei* als Guile-Skript ausgeführt.

```
guix repl my-script.scm
```

Um Argumente an das Skript zu übergeben, geben Sie davor `--` an, damit Sie nicht als Argumente an `guix repl` verstanden werden:

```
guix repl -- my-script.scm --input=foo.txt
```

Wenn Sie möchten, dass ein Skript direkt aus der Shell heraus ausgeführt werden kann und diejenige ausführbare Datei von Guix benutzt wird, die sich im Suchpfad des Benutzers befindet, dann fügen Sie die folgenden zwei Zeilen ganz oben ins Skript ein.

```
#!/usr/bin/env -S guix repl --
!#
```

Ohne einen Dateinamen als Argument wird eine Guile-REPL gestartet, so dass man Guix interaktiv benutzen kann (siehe Abschnitt 9.14 [Interaktiv mit Guix arbeiten], Seite 186):

```
$ guix repl
scheme@(guile-user)> ,use (gnu packages base)
scheme@(guile-user)> coreutils
$1 = #<package coreutils@8.29 gnu/packages/base.scm:327 3e28300>
```

`guix repl` implementiert zusätzlich ein einfaches maschinenlesbares Protokoll für die REPL, das von `(guix inferior)` benutzt wird, um mit *Untergeordneten* zu interagieren, also mit getrennten Prozessen einer womöglich anderen Version von Guix.

Folgende *Optionen* gibt es:

`--type=`*Typ*
`-t` *Typ* Startet eine REPL des angegebenen *Typ*s, der einer der Folgenden sein darf:

- `guile` Die Voreinstellung, mit der eine normale, voll funktionsfähige Guile-REPL gestartet wird.

`machine` Startet eine REPL, die ein maschinenlesbares Protokoll benutzt. Dieses Protokoll wird vom Modul `(guix inferior)` gesprochen.

`--listen=`*Endpunkt*
: Der Vorgabe nach würde `guix repl` von der Standardeingabe lesen und auf die Standardausgabe schreiben. Wird diese Befehlszeilenoption angegeben, lauscht die REPL stattdessen auf dem *Endpunkt* auf Verbindungen. Hier sind Beispiele gültiger Befehlszeilenoptionen:

  `--listen=tcp:37146`
  : Verbindungen mit dem „localhost“ auf Port 37146 akzeptieren.

  `--listen=unix:/tmp/socket`
  : Verbindungen zum Unix-Socket `/tmp/socket` akzeptieren.

`--load-path=`*Verzeichnis*
`-L` *Verzeichnis*
: Das *Verzeichnis* vorne an den Suchpfad für Paketmodule anfügen (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

  Damit können Nutzer dafür sorgen, dass ihre eigenen selbstdefinierten Pakete für Skript oder REPL sichtbar sind.

`-q` Das Laden der `~/.guile`-Datei unterdrücken. Nach Voreinstellung würde diese Konfigurationsdatei beim Erzeugen einer REPL für `guile` geladen.

9.14 Interaktiv mit Guix arbeiten

Mit dem Befehl `guix repl` finden Sie sich in einer netten und freundlichen REPL wieder (das steht für Read-Eval-Print Loop, deutsch Lese-Auswerten-Schreiben-Schleife) (siehe Abschnitt 9.13 [Aufruf von guix repl], Seite 185). Wenn Sie mit Guix programmieren möchten – also eigene Pakete definieren, Manifeste schreiben, Dienste für Guix System oder Guix Home definieren und so –, dann wird Ihnen sicher gefallen, dass Sie Ihre Ideen auf der REPL ausprobieren können.

Wenn Sie Emacs benutzen, eignet sich dafür Geiser am meisten (siehe Abschnitt 22.3 [Perfekt eingerichtet], Seite 711), aber Emacs zu benutzen ist nicht unbedingt erforderlich, um Spaß an der REPL zu haben. Beim Verwenden von `guix repl` oder `guile` auf dem Terminal raten wir dazu, dass Sie Readline aktivieren, um Autovervollständigung zu genießen, und Colorized aktivieren für farbige Ausgaben. Führen Sie dazu dies aus:

```
guix install guile guile-readline guile-colorized
```

... und legen Sie dann eine Datei `.guile` in Ihrem Persönlichen Verzeichnis („Home“-Verzeichnis) an mit diesem Inhalt:

```
(use-modules (ice-9 readline) (ice-9 colorized))

(activate-readline)
(activate-colorized)
```

Auf der REPL können Sie Scheme-Code auswerten lassen. Sie tippen also einen Scheme-Ausdruck nach der Eingabeaufforderung und schon zeigt die REPL, zu was er ausgewertet wird:

```
$ guix repl
```

```
scheme@(guix-user)> (+ 2 3)
$1 = 5
scheme@(guix-user)> (string-append "a" "b")
$2 = "ab"
```

Interessant wird es, wenn Sie auf der REPL anfangen, mit Guix herumzubasteln. Dazu „importieren“ Sie zunächst das Modul `(guix)`, damit Sie Zugriff auf den Hauptteil der Programmierschnittstelle haben, und vielleicht wollen Sie auch noch andere nützliche Guix-Module importieren. Sie könnten `(use-modules (guix))` eintippen, denn es ist gültiger Scheme-Code zum Importieren eines Moduls (siehe Abschnitt “Using Guile Modules” in *Referenzhandbuch zu GNU Guile*), aber auf der REPL können Sie den *Befehl* `use` als Kurzschreibweise einsetzen (siehe Abschnitt “REPL Commands” in *Referenzhandbuch zu GNU Guile*):

```
scheme@(guix-user)> ,use (guix)
scheme@(guix-user)> ,use (gnu packages base)
```

Beachten Sie, dass am Anfang jedes REPL-Befehls ein führendes Komma stehen muss. Der Grund ist, dass ein REPL-Befehl wie `use` eben *kein* gültiger Scheme-Code ist; er wird von der REPL gesondert interpretiert.

Durch Guix wird die Guile-REPL um zusätzliche Befehle erweitert, die dort praktisch sind. Einer davon, der Befehl `build`, ist hier nützlich: Mit ihm wird sichergestellt, dass das angegebene dateiartige Objekt erstellt wurde; wenn nicht, wird es erstellt. In jedem Fall wird der Dateiname jeder Ausgabe der Erstellung zurückgeliefert. Im folgenden Beispiel erstellen wir die Pakete `coreutils` und `grep` sowie eine als `computed-file` angegebene Datei (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), um sogleich mit der Prozedur `scandir` aufzulisten, was für Dateien in Grep im Verzeichnis `/bin` enthalten sind:

```
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))
/gnu/store/...-x.drv wird erstellt ...
$4 = "/gnu/store/...-x"
scheme@(guix-user)> ,use(ice-9 ftw)
scheme@(guix-user)> (scandir (string-append $3 "/bin"))
$5 = ("." ".." "egrep" "fgrep" "grep")
```

Will man die einzelnen Schritte von Guix nachvollziehen, eignet sich der Befehl `lower`: Er nimmt ein dateiartiges Objekt, um es zu einer Ableitung oder einem Store-Objekt „herunterzubrechen“ (siehe Abschnitt 9.10 [Ableitungen], Seite 167):

```
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" "Hallo!")
$7 = "/gnu/store/...-x"
```

Die vollständige Liste der REPL-Befehle bekommen Sie zu sehen, wenn Sie `,help guix` eingeben. Hier eine Referenz:

`build` *`Objekt`* [REPL-Befehl]

Das *Objekt* herunterbrechen und erstellen, wenn es noch nicht erstellt ist. Als Ergebnis zurückgeliefert wird der Dateiname jeder Ausgabe.

`lower` *`Objekt`* [REPL-Befehl]

Das *Objekt* zu einer Ausgabe oder einem Dateinamen im Store herunterbrechen und diesen zurückliefern.

`verbosity` *`Stufe`* [REPL-Befehl]

Legt *Stufe* als die Ausführlichkeitsstufe für Erstellungen fest.

Das ist das Gleiche wie die Befehlszeilenoption `--verbosity` (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189): Stufe 0 zeigt gar nichts an, Stufe 1 nur Ereignisse bei der Erstellung und auf höheren Stufen bekommen Sie Erstellungsprotokolle angezeigt.

`run-in-store` *`Ausdruck`* [REPL-Befehl]

Den *Ausdruck*, einen monadischen Ausdruck, durch die Store-Monade laufen lassen. Siehe Abschnitt 9.11 [Die Store-Monade], Seite 170, für mehr Erklärungen.

`enter-store-monad` [REPL-Befehl]

Damit betreten Sie eine neue REPL, die monadische Ausdrücke auswerten kann (siehe Abschnitt 9.11 [Die Store-Monade], Seite 170). Sie können diese „innere“ REPL verlassen, indem Sie `,q` eintippen.

10 Zubehör

Dieser Abschnitt beschreibt die Befehlszeilenwerkzeuge von Guix. Manche davon richten sich hauptsächlich an Entwickler und solche Nutzer, die neue Paketdefinitionen schreiben, andere sind auch für ein breiteres Publikum nützlich. Sie ergänzen die Scheme-Programmierschnittstelle um bequeme Befehle.

10.1 Aufruf von `guix build`

Der Befehl `guix build` lässt Pakete oder Ableitungen samt ihrer Abhängigkeiten erstellen und gibt die resultierenden Pfade im Store aus. Beachten Sie, dass das Nutzerprofil dadurch nicht modifiziert wird – eine solche Installation bewirkt der Befehl `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). `guix build` wird also hauptsächlich von Entwicklern der Distribution benutzt.

Die allgemeine Syntax lautet:

```
guix build Optionen Paket-oder-Ableitung...
```

Zum Beispiel wird mit folgendem Befehl die neueste Version von Emacs und von Guile erstellt, das zugehörige Erstellungsprotokoll angezeigt und letztendlich werden die resultierenden Verzeichnisse ausgegeben:

```
guix build emacs guile
```

Folgender Befehl erstellt alle Pakete, die zur Verfügung stehen:

```
guix build --quiet --keep-going \
  $(guix package -A | awk '{ print $1 "@" $2 }')
```

Als *Paket-oder-Ableitung* muss entweder der Name eines in der Software-Distribution zu findenden Pakets, wie etwa `coreutils` oder `coreutils@8.20`, oder eine Ableitung wie `/gnu/store/...-coreutils-8.19.drv` sein. Im ersten Fall wird nach einem Paket mit entsprechendem Namen (und optional der entsprechenden Version) in den Modulen der GNU-Distribution gesucht (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

Alternativ kann die Befehlszeilenoption `--expression` benutzt werden, um einen Scheme-Ausdruck anzugeben, der zu einem Paket ausgewertet wird; dies ist nützlich, wenn zwischen mehreren gleichnamigen Paketen oder Paket-Varianten unterschieden werden muss.

Null oder mehr *Optionen* können angegeben werden. Zur Verfügung stehen die in den folgenden Unterabschnitten beschriebenen Befehlszeilenoptionen.

10.1.1 Gemeinsame Erstellungsoptionen

Einige dieser Befehlszeilenoptionen zur Steuerung des Erstellungsprozess haben `guix build` und andere Befehle, mit denen Erstellungen ausgelöst werden können, wie `guix package` oder `guix archive`, gemeinsam. Das sind folgende:

`--load-path=`*Verzeichnis*
`-L` *Verzeichnis*

> Das *Verzeichnis* vorne an den Suchpfad für Paketmodule anfügen (siehe Abschnitt 9.1 [Paketmodule], Seite 108).
>
> Damit können Nutzer dafür sorgen, dass ihre eigenen selbstdefinierten Pakete für die Befehlszeilenwerkzeuge sichtbar sind.

`--keep-failed`
`-K` Den Verzeichnisbaum, in dem fehlgeschlagene Erstellungen durchgeführt wurden, behalten. Wenn also eine Erstellung fehlschlägt, bleibt ihr Erstellungsbaum in `/tmp` erhalten. Der Name dieses Unterverzeichnisses wird am Ende dem Erstellungsprotokolls ausgegeben. Dies hilft bei der Suche nach Fehlern in Erstellungen. Der Abschnitt Abschnitt 10.1.4 [Fehlschläge beim Erstellen untersuchen], Seite 203, zeigt Ihnen Hinweise und Tricks, wie Erstellungsfehler untersucht werden können.

Diese Option impliziert `--no-offload` und sie hat keine Auswirkungen, wenn eine Verbindung zu einem entfernten Daemon über eine `guix://`-URI verwendet wurde (siehe Abschnitt 9.9 [Der Store], Seite 165).

`--keep-going`
`-k` Weitermachen, auch wenn ein Teil der Erstellungen fehlschlägt. Das bedeutet, dass der Befehl erst terminiert, wenn alle Erstellungen erfolgreich oder mit Fehler durchgeführt wurden.

Das normale Verhalten ist, abzubrechen, sobald eine der angegebenen Ableitungen fehlschlägt.

`--dry-run`
`-n` Die Ableitungen nicht erstellen.

`--fallback`
Wenn das Substituieren vorerstellter Binärdateien fehlschlägt, diese ersatzweise lokal selbst erstellen (siehe Abschnitt 6.3.6 [Fehler bei der Substitution], Seite 59).

`--substitute-urls=`*URLs*
Die *urls* als durch Leerraumzeichen getrennte Liste von Quell-URLs für Substitute anstelle der vorgegebenen URL-Liste für den `guix-daemon` verwenden (siehe [`guix-daemon` URLs], Seite 19).

Das heißt, die Substitute dürfen von den *urls* heruntergeladen werden, sofern sie mit einem durch den Systemadministrator autorisierten Schlüssel signiert worden sind (siehe Abschnitt 6.3 [Substitute], Seite 56).

Wenn als *urls* eine leere Zeichenkette angegeben wurde, verhält es sich, als wären Substitute abgeschaltet.

`--no-substitutes`
Benutze keine Substitute für Erstellungsergebnisse. Das heißt, dass alle Objekte lokal erstellt werden müssen, und kein Herunterladen von vorab erstellten Binärdateien erlaubt ist (siehe Abschnitt 6.3 [Substitute], Seite 56).

`--no-grafts`
Pakete nicht „veredeln“ (engl. „graft“). Praktisch heißt das, dass als Veredelungen verfügbare Paketaktualisierungen nicht angewandt werden. Der Abschnitt Kapitel 19 [Sicherheitsaktualisierungen], Seite 697, hat weitere Informationen zu Veredelungen.

`--rounds=`*n*
Jede Ableitung *n*-mal nacheinander erstellen und einen Fehler melden, wenn die aufeinanderfolgenden Erstellungsergebnisse nicht Bit für Bit identisch sind.

Das ist eine nützliche Methode, um nicht-deterministische Erstellungsprozesse zu erkennen. Nicht-deterministische Erstellungsprozesse sind ein Problem, weil Nutzer dadurch praktisch nicht *verifizieren* können, ob von Drittanbietern bereitgestellte Binärdateien unverfälscht sind. Der Abschnitt Abschnitt 10.12 [Aufruf von guix challenge], Seite 238, erklärt dies genauer.

Wenn dies zusammen mit `--keep-failed` benutzt wird, bleiben die sich unterscheidenden Ausgaben im Store unter dem Namen `/gnu/store/...-check`. Dadurch können Unterschiede zwischen den beiden Ergebnissen leicht erkannt werden.

`--no-offload`
: Nicht versuchen, an andere Maschinen ausgelagerte Erstellungen zu benutzen (siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13). Somit wird lokal erstellt, statt Erstellungen auf entfernte Maschinen auszulagern.

`--max-silent-time=`*Sekunden*
: Wenn der Erstellungs- oder Substitutionsprozess länger als *Sekunden*-lang keine Ausgabe erzeugt, wird er abgebrochen und ein Fehler beim Erstellen gemeldet.

  Standardmäßig wird die Einstellung für den Daemon benutzt (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

`--timeout=`*Sekunden*
: Entsprechend wird hier der Erstellungs- oder Substitutionsprozess abgebrochen und als Fehlschlag gemeldet, wenn er mehr als *Sekunden*-lang dauert.

  Standardmäßig wird die Einstellung für den Daemon benutzt (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

`-v` *Stufe*
`--verbosity=`*Stufe*
: Die angegebene Ausführlichkeitsstufe verwenden. Als *Stufe* muss eine ganze Zahl angegeben werden. Wird 0 gewählt, wird keine Ausgabe zur Fehlersuche angezeigt, 1 bedeutet eine knappe Ausgabe, 2 ist wie 1, aber zeigt zusätzlich an, von welcher URL heruntergeladen wird, und 3 lässt alle Erstellungsprotokollausgaben auf die Standardfehlerausgabe schreiben.

`--cores=`*n*
`-c` *n*
: Die Nutzung von bis zu *n* Prozessorkernen für die Erstellungen gestatten. Der besondere Wert `0` bedeutet, dass so viele wie möglich benutzt werden.

`--max-jobs=`*n*
`-M` *n*
: Höchstens *n* gleichzeitige Erstellungsaufträge erlauben. Im Abschnitt Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18, finden Sie Details zu dieser Option und der äquivalenten Option des `guix-daemon`.

`--debug=`*Stufe*
: Ein Protokoll zur Fehlersuche ausgeben, das vom Erstellungsdaemon kommt. Als *Stufe* muss eine ganze Zahl zwischen 0 und 5 angegeben werden; höhere Zahlen stehen für ausführlichere Ausgaben. Stufe 4 oder höher zu wählen, kann bei der Suche nach Fehlern, wie der Erstellungs-Daemon eingerichtet ist, helfen.

Intern ist `guix build` im Kern eine Schnittstelle zur Prozedur `package-derivation` aus dem Modul `(guix packages)` und zu der Prozedur `build-derivations` des Moduls `(guix derivations)`.

Neben auf der Befehlszeile übergebenen Optionen beachten `guix build` und andere `guix`-Befehle, die Erstellungen durchführen lassen, die Umgebungsvariable `GUIX_BUILD_OPTIONS`.

`GUIX_BUILD_OPTIONS` [Umgebungsvariable]

Nutzer können diese Variable auf eine Liste von Befehlszeilenoptionen definieren, die automatisch von `guix build` und anderen `guix`-Befehlen, die Erstellungen durchführen lassen, benutzt wird, wie in folgendem Beispiel:

```
$ export GUIX_BUILD_OPTIONS="--no-substitutes -c 2 -L /foo/bar"
```

Diese Befehlszeilenoptionen werden unabhängig von den auf der Befehlszeile übergebenen Befehlszeilenoptionen grammatikalisch analysiert und das Ergebnis an die bereits analysierten auf der Befehlszeile übergebenen Befehlszeilenoptionen angehängt.

10.1.2 Paketumwandlungsoptionen

Eine weitere Gruppe von Befehlszeilenoptionen, die `guix build` und auch `guix package` unterstützen, sind *Paketumwandlungsoptionen*. Diese Optionen ermöglichen es, *Paketvarianten* zu definieren – zum Beispiel können Pakete aus einem anderen Quellcode als normalerweise erstellt werden. Damit ist es leicht, angepasste Pakete schnell zu erstellen, ohne die vollständigen Definitionen von Paketvarianten einzutippen (siehe Abschnitt 9.2 [Pakete definieren], Seite 109).

Paketumwandlungsoptionen bleiben über Aktualisierungen hinweg erhalten: `guix upgrade` versucht, Umwandlungsoptionen, die vorher zur Erstellung des Profils benutzt wurden, auf die aktualisierten Pakete anzuwenden.

Im Folgenden finden Sie die verfügbaren Befehlszeilenoptionen. Die meisten Befehle unterstützen sie ebenso wie eine Option `--help-transform`, mit der all die verfügbaren Optionen und je eine Kurzbeschreibung dazu angezeigt werden. (Diese Optionen werden der Kürze wegen nicht in der Ausgabe von `--help` aufgeführt.)

`--tune[=`*CPU*`]`

Die optimierte Version als „tunebar“ markierter Pakete benutzen. *CPU* gibt die Prozessorarchitektur an, für die optimiert werden soll. Wenn als *CPU* die Bezeichnung `native` angegeben wird oder nichts angegeben wird, wird für den Prozessor optimiert, auf dem der Befehl `guix` läuft.

Gültige Namen für *CPU* sind genau die, die vom zugrunde liegenden Compiler erkannt werden. Vorgegeben ist, dass als Compiler die GNU Compiler Collection benutzt wird. Auf x86_64-Prozessoren gehören `nehalem`, `haswell`, und `skylake` zu den CPU-Namen (siehe Abschnitt “x86 Options” in *Using the GNU Compiler Collection (GCC)*).

Mit dem Erscheinen neuer Generationen von Prozessoren wächst der Standardbefehlssatz (die „Instruction Set Architecture“, ISA) um neue Befehle an, insbesondere wenn es um Befehle zur Parallelverarbeitung geht („Single-Instruction/Multiple-Data“, SIMD). Zum Beispiel implementieren

sowohl die Core2- als auch die Skylake-Prozessoren den x86_64-Befehlssatz, jedoch können nur letztere AVX2-SIMD-Befehle ausführen.

Der Mehrwert, den `--tune` bringt, besteht in erster Linie bei Programmen, für die SIMD-Fähigkeiten geeignet wären *und* die über keinen Mechanismus verfügen, zur Laufzeit die geeigneten Codeoptimierungen zuzuschalten. Pakete, bei denen die Eigenschaft `tunable?` angegeben wurde, werden bei der Befehlszeilenoption `--tune` als *tunebare Pakete* optimiert. Eine Paketdefinition, bei der diese Eigenschaft hinterlegt wurde, sieht so aus:

```
(package
  (name "hello-simd")
  ;; ...

  ;; Dieses Paket kann von SIMD-Erweiterungen profitieren,
  ;; deshalb markieren wir es als "tunebar".
  (properties '((tunable? . #t))))
```

Andere Pakete werden als *nicht* tunebar aufgefasst. Dadurch kann Guix allgemeine Binärdateien verwenden, wenn sich die Optimierung für einen bestimmten Prozessor wahrscheinlich *nicht* lohnt.

Bei der Erstellung tunebarer Pakete wird `-march=`*`CPU`* übergeben. Intern wird die Befehlszeilenoption `-march` durch einen Compiler-Wrapper an den eigentlichen Wrapper übergeben. Weil die Erstellungsmaschine den Code für die Mikroarchitektur vielleicht gar nicht ausführen kann, wird der Testkatalog bei der Erstellung tunebarer Pakete übersprungen.

Damit weniger Neuerstellungen erforderlich sind, werden die von tunebaren Paketen abhängigen Pakete mit den optimierten Paketen *veredelt* (siehe Kapitel 19 [Sicherheitsaktualisierungen], Seite 697). Wenn Sie `--no-grafts` übergeben, wirkt `--tune` deshalb *nicht* mehr.

Wir geben dieser Technik den Namen *Paket-Multiversionierung*: Mehrere Varianten des tunebaren Pakets können erstellt werden; eine für jede Prozessorvariante. Das ist die grobkörnige Entsprechung der *Funktions-Multiversionierung*, die in der GNU-Toolchain zu finden ist (siehe Abschnitt “Function Multiversioning” in *Using the GNU Compiler Collection (GCC)*).

`--with-source=`*`Quelle`*
`--with-source=`*`Paket=Quelle`*
`--with-source=`*`Paket@Version=Quelle`*

Den Paketquellcode für das *Paket* von der angegebenen *Quelle* holen und die *Version* als seine Versionsnummer verwenden. Die *Quelle* muss ein Dateiname oder eine URL sein wie bei `guix download` (siehe Abschnitt 10.3 [Aufruf von guix download], Seite 204).

Wird kein *Paket* angegeben, wird als Paketname derjenige auf der Befehlszeile angegebene Paketname angenommen, der zur Basis am Ende der *Quelle* passt – wenn z.B. als *Quelle* die Datei `/src/guile-2.0.10.tar.gz` angegeben wurde, entspricht das dem `guile`-Paket.

Ebenso wird, wenn keine *Version* angegeben wurde, die Version als Zeichenkette aus der *Quelle* abgeleitet; im vorherigen Beispiel wäre sie `2.0.10`.

Mit dieser Option können Nutzer versuchen, eine andere Version ihres Pakets auszuprobieren, als die in der Distribution enthaltene Version. Folgendes Beispiel lädt `ed-1.7.tar.gz` von einem GNU-Spiegelserver herunter und benutzt es als Quelle für das `ed`-Paket:

```
guix build ed --with-source=mirror://gnu/ed/ed-1.4.tar.gz
```

Für Entwickler wird es einem durch `--with-source` leicht gemacht, „Release Candidates“, also Vorabversionen, zu testen, oder sogar welchen Einfluss diese auf abhängige Pakete haben:

```
guix build elogind --with-source=.../shepherd-0.9.0rc1.tar.gz
```

. . . oder ein Checkout eines versionskontrollierten Repositorys in einer isolierten Umgebung zu erstellen:

```
$ git clone git://git.sv.gnu.org/guix.git
$ guix build guix --with-source=guix@1.0=./guix
```

`--with-input=`*Paket*`=`*Ersatz*

Abhängigkeiten vom *Paket* durch eine Abhängigkeit vom *Ersatz*-Paket ersetzen. Als *Paket* muss ein Paketname angegeben werden und als *Ersatz* eine Paketspezifikation wie `guile` oder `guile@1.8`.

Mit folgendem Befehl wird zum Beispiel Guix erstellt, aber statt der aktuellen stabilen Guile-Version hängt es von der alten Guile-Version `guile@2.0` ab:

```
guix build --with-input=guile=guile@2.0 guix
```

Die Ersetzung ist rekursiv und umfassend. In diesem Beispiel würde nicht nur `guix`, sondern auch seine Abhängigkeit `guile-json` (was auch von `guile` abhängt) für `guile@2.0` neu erstellt.

Implementiert wird das alles mit der Scheme-Prozedur `package-input-rewriting` (siehe Abschnitt 9.2 [Pakete definieren], Seite 109).

`--with-graft=`*Paket*`=`*Ersatz*

Dies verhält sich ähnlich wie mit `--with-input`, aber mit dem wichtigen Unterschied, dass nicht die gesamte Abhängigkeitskette neu erstellt wird, sondern das *Ersatz*-Paket erstellt und die ursprünglichen Binärdateien, die auf das *Paket* verwiesen haben, damit *veredelt* werden. Im Abschnitt Kapitel 19 [Sicherheitsaktualisierungen], Seite 697, finden Sie weitere Informationen über Veredelungen.

Zum Beispiel veredelt folgender Befehl Wget und alle Abhängigkeiten davon mit der Version 3.5.4 von GnuTLS, indem Verweise auf die ursprünglich verwendete GnuTLS-Version ersetzt werden:

```
guix build --with-graft=gnutls=gnutls@3.5.4 wget
```

Das hat den Vorteil, dass es viel schneller geht, als alles neu zu erstellen. Die Sache hat aber einen Haken: Veredelung funktioniert nur, wenn das *Paket* und sein *Ersatz* miteinander streng kompatibel sind – zum Beispiel muss, wenn diese eine Programmbibliothek zur Verfügung stellen, deren Binärschnittstelle („Application Binary Interface“, kurz ABI) kompatibel sein. Wenn das *Ersatz*-Paket auf irgendeine Art inkompatibel mit dem *Paket* ist, könnte das Ergebnispaket unbrauchbar sein. Vorsicht ist also geboten!

`--with-debug-info=`*Paket*

Das *Paket* auf eine Weise erstellen, die Informationen zur Fehlersuche enthält, und von ihm abhängige Pakete damit veredeln. Das ist nützlich, wenn das *Paket* noch keine Fehlersuchinformationen als installierbare `debug`-Ausgabe enthält (siehe Kapitel 17 [Dateien zur Fehlersuche installieren], Seite 692).

Als Beispiel nehmen wir an, Inkscape stürzt bei Ihnen ab und Sie möchten wissen, was dabei in GLib passiert. Die GLib-Bibliothek liegt tief im Abhängigkeitsgraphen von Inkscape und verfügt nicht über eine `debug`-Ausgabe; das erschwert die Fehlersuche. Glücklicherweise können Sie GLib mit Informationen zur Fehlersuche neu erstellen und an Inkscape anheften:

```
guix install inkscape --with-debug-info=glib
```

Nur GLib muss neu kompiliert werden, was in vernünftiger Zeit möglich ist. Siehe Kapitel 17 [Dateien zur Fehlersuche installieren], Seite 692, für mehr Informationen.

Anmerkung: Intern funktioniert diese Option, indem '`#:strip-binaries? #f`' an das Erstellungssystem des betreffenden Pakets übergeben wird (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). Die meisten Erstellungssysteme unterstützen diese Option, manche aber nicht. In diesem Fall wird ein Fehler gemeldet.

Auch wenn ein in C/C++ geschriebenes Paket ohne `-g` erstellt wird (was selten der Fall ist), werden Informationen zur Fehlersuche weiterhin fehlen, obwohl `#:strip-binaries?` auf falsch steht.

`--with-c-toolchain=`*Paket*`=`*Toolchain*

Mit dieser Befehlszeilenoption wird die Kompilierung des *Paket*s und aller davon abhängigen Objekte angepasst, so dass mit der *Toolchain* statt der vorgegebenen GNU-Toolchain für C/C++ erstellt wird.

Betrachten Sie dieses Beispiel:

```
guix build octave-cli \
  --with-c-toolchain=fftw=gcc-toolchain@10 \
  --with-c-toolchain=fftwf=gcc-toolchain@10
```

Mit dem obigen Befehl wird eine Variante der Pakete `fftw` und `fftwf` mit Version 10 der `gcc-toolchain` anstelle der vorgegebenen Toolchain erstellt, um damit anschließend eine diese benutzende Variante des GNU-Octave-Befehlszeilenprogramms zu erstellen. Auch GNU Octave selbst wird mit `gcc-toolchain@10` erstellt.

Das zweite Beispiel bewirkt eine Erstellung der „Hardware Locality"-Bibliothek (`hwloc`) sowie ihrer abhängigen Objekte bis einschließlich `intel-mpi-benchmarks` mit dem Clang-C-Compiler:

```
guix build --with-c-toolchain=hwloc=clang-toolchain \
           intel-mpi-benchmarks
```

Anmerkung: Es kann vorkommen, dass die Anwendungsbinärschnittstellen („Application Binary Interfaces", kurz ABIs)

der Toolchains inkompatibel sind. Das tritt vor allem bei der C++-Standardbibliothek und Bibliotheken zur Laufzeitunterstützung wie denen von OpenMP auf. Indem alle abhängigen Objekte mit derselben Toolchain erstellt werden, minimiert `--with-c-toolchain` das Risiko, dass es zu Inkompatibilitäten kommt, aber es kann nicht ganz ausgeschlossen werden. Bedenken Sie, für welches *Paket* Sie dies benutzen.

`--with-git-url=`*Paket*`=`*URL*

Das *Paket* aus dem neuesten Commit im `master`-Branch des unter der *URL* befindlichen Git-Repositorys erstellen. Git-Submodule des Repositorys werden dabei rekursiv geladen.

Zum Beispiel erstellt der folgende Befehl die NumPy-Python-Bibliothek unter Verwendung des neuesten Commits von Python auf dessen „master“-Branch.

```
guix build python-numpy \
  --with-git-url=python=https://github.com/python/cpython
```

Diese Befehlszeilenoption kann auch mit `--with-branch` oder `--with-commit` kombiniert werden (siehe unten).

Da es den neuesten Commit auf dem verwendeten Branch benutzt, ändert sich das Ergebnis natürlich mit der Zeit. Nichtsdestoweniger ist es eine bequeme Möglichkeit, ganze Softwarestapel auf dem neuesten Commit von einem oder mehr Paketen aufbauen zu lassen. Es ist besonders nützlich im Kontext Kontinuierlicher Integration (englisch „Continuous Integration“, kurz CI).

Checkouts bleiben zwischengespeichert als `~/.cache/guix/checkouts`, damit danach schneller auf dasselbe Repository zugegriffen werden kann. Eventuell möchten Sie das Verzeichnis ab und zu bereinigen, um Plattenplatz zu sparen.

`--with-branch=`*Paket*`=`*Branch*

Das *Paket* aus dem neuesten Commit auf dem *Branch* erstellen. Wenn das `source`-Feld des *Paket*s ein origin-Objekt mit der Methode `git-fetch` (siehe Abschnitt 9.2.2 [„origin“-Referenz], Seite 118) oder ein `git-checkout`-Objekt ist, wird die URL des Repositorys vom `source`-Feld genommen. Andernfalls müssen Sie die Befehlszeilenoption `--with-git-url` benutzen, um die URL des Git-Repositorys anzugeben.

Zum Beispiel wird mit dem folgenden Befehl `guile-sqlite3` aus dem neuesten Commit seines `master`-Branches erstellt und anschließend `guix` (was von `guile-sqlite3` abhängt) und `cuirass` (was von `guix` abhängt) unter Nutzung genau dieser `guile-sqlite3`-Erstellung erstellt:

```
guix build --with-branch=guile-sqlite3=master cuirass
```

`--with-commit=`*Paket*`=`*Commit*

Dies verhält sich ähnlich wie `--with-branch`, außer dass es den angegebenen *Commit* benutzt statt die Spitze eines angegebenen Branches. Als *Commit* muss ein gültiger SHA1-Bezeichner, ein Tag oder ein Bezeichner wie von `git describe` (wie `1.0-3-gabc123`) für einen Git-Commit angegeben werden.

`--with-patch=`*Paket*`=`*Datei*

Die *Datei* zur Liste der auf das *Paket* anzuwendenden Patches hinzufügen. Als *Paket* muss eine Spezifikation wie `python@3.8` oder `glibc` benutzt werden. In der *Datei* muss ein Patch enthalten sein; er wird mit den im Ursprung (`origin`) des *Pakets* angegebenen Befehlszeilenoptionen angewandt (siehe Abschnitt 9.2.2 [„origin“-Referenz], Seite 118). Die vorgegebenen Optionen enthalten `-p1` (siehe Abschnitt “patch Directories” in *Comparing and Merging Files*).

Zum Beispiel wird mit dem folgenden Befehl für die Neuerstellung von Coreutils die GNU-C-Bibliothek (glibc) wie angegeben gepatcht:

```
guix build coreutils --with-patch=glibc=./glibc-frob.patch
```

In diesem Beispiel wird glibc selbst und alles, was im Abhängigkeitsgraphen auf dem Weg zu Coreutils liegt, neu erstellt.

`--with-latest=`*Paket*

Sie hätten gerne das Neueste vom Neuen? Dann ist diese Befehlszeilenoption das Richtige für Sie! Damit wird jedes Vorkommen von *Paket* im Abhängigkeitsgraphen durch dessen neueste angebotene Version ersetzt, wie sie auch von `guix refresh` gemeldet würde (siehe Abschnitt 10.6 [Aufruf von guix refresh], Seite 214).

Dazu wird die neueste angebotene Version des *Pakets* ermittelt (wenn möglich), heruntergeladen und, *wenn* eine OpenPGP-Signatur mit dabei ist, es damit authentifiziert.

Zum Beispiel wird durch folgenden Befehl Guix mit der neuesten Version von Guile-JSON erstellt:

```
guix build guix --with-latest=guile-json
```

Es gibt jedoch Einschränkungen. Erstens gehen Sie in dem Fall, dass das Werkzeug nicht in der Lage ist oder nicht weiß, wie es den Quellcode authentifiziert, das Risiko ein, dass bösartiger Code ausgeführt wird; Ihnen wird dann eine Warnung angezeigt. Zweitens wird mit dieser Option einfach der Quellcode ausgetauscht und die übrige Paketdefinition bleibt erhalten. Manchmal reicht das nicht; es könnte sein, dass neue Abhängigkeiten hinzugefügt oder neue Patches angewandt werden müssen oder dass ganz allgemein Arbeiten zur Qualitätssicherung, die Guix-Entwickler normalerweise leisten, fehlen werden.

Sie sind gewarnt worden! Wenn aber kein Problem auftritt, können Sie das Paket zackig auf den neuesten Stand bringen. Wir ermutigen Sie dazu, Patches einzureichen, die die eigentliche Paketdefinition aktualisieren, sobald Sie die neue Version erfolgreich getestet haben (siehe Kapitel 22 [Mitwirken], Seite 708).

`--without-tests=`*Paket*

Das *Paket* erstellen, ohne seine Tests zu durchlaufen. Das erweist sich als nützlich, wenn Sie Testkataloge überspringen möchten, die viel Zeit in Anspruch nehmen, oder wenn der Testkatalog eines Pakets nichtdeterministisch fehlschlägt. Dies sollte mit Bedacht eingesetzt werden, denn das Ausführen des Testkatalogs sichert zu, dass ein Paket wie gewollt funktioniert.

Wenn die Tests abgeschaltet werden, ergibt sich ein anderes Store-Objekt. Dadurch muss, wenn diese Option benutzt wird, auch alles, was vom Paket abhängt, neu erstellt werden, wie Sie in diesem Beispiel sehen können:

```
guix install --without-tests=python python-notebook
```

Mit obigem Befehl wird `python-notebook` für ein `python` installiert, dessen Testkatalog nicht ausgeführt wurde. Dazu wird auch alles neu erstellt, was von `python` abhängt, einschließlich `python-notebook`.

Intern funktioniert `--without-tests`, indem es die Option `#:tests?` der `check`-Phase eines Pakets abändert (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). Beachten Sie, dass manche Pakete eine angepasste `check`-Phase benutzen, die eine Einstellung wie `#:tests? #f` nicht berücksichtigt. Deshalb wirkt sich `--without-tests` auf diese Pakete nicht aus.

Sie fragen sich sicher, wie Sie dieselbe Wirkung mit Scheme-Code erzielen können, zum Beispiel wenn Sie Ihr Manifest oder eine eigene Paketumwandlung schreiben? Siehe Abschnitt 9.3 [Paketvarianten definieren], Seite 121, für eine Übersicht über verfügbare Programmierschnittstellen.

10.1.3 Zusätzliche Erstellungsoptionen

Die unten aufgeführten Befehlszeilenoptionen funktionieren nur mit `guix build`.

`--quiet`
`-q` Schweigend erstellen, ohne das Erstellungsprotokoll anzuzeigen – dies ist äquivalent zu `--verbosity=0`. Nach Abschluss der Erstellung ist das Protokoll in `/var` (oder einem entsprechenden Ort) einsehbar und kann jederzeit mit der Befehlszeilenoption `--log-file` gefunden werden.

`--file=`*Datei*
`-f` *Datei* Das Paket, die Ableitung oder das dateiähnliche Objekt erstellen, zu dem der Code in der *Datei* ausgewertet wird (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

Zum Beispiel könnte in der *Datei* so eine Paketdefinition stehen (siehe Abschnitt 9.2 [Pakete definieren], Seite 109):

```
(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+))
```

Die *Datei* darf auch eine JSON-Darstellung von einer oder mehreren Paketdefinitionen sein. Wenn wir `guix build -f` auf einer `hello.json`-Datei mit dem folgenden Inhalt ausführen würden, würden die Pakete `myhello` und `greeter` erstellt werden:

```
[
  {
    "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": "https://example.com/greeter-1.0.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=`*Manifest*
`-m` *Manifest*

Alle Pakete erstellen, die im angegebenen *Manifest* stehen (siehe [profile-manifest], Seite 50).

`--expression=`*Ausdruck*
`-e` *Ausdruck*

Das Paket oder die Ableitung erstellen, zu der der *Ausdruck* ausgewertet wird.

Zum Beispiel kann der *Ausdruck* `(@ (gnu packages guile) guile-1.8)` sein, was diese bestimmte Variante der Version 1.8 von Guile eindeutig bezeichnet.

Alternativ kann der *Ausdruck* ein G-Ausdruck sein. In diesem Fall wird er als Erstellungsprogramm an `gexp->derivation` übergeben (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

Zudem kann der *Ausdruck* eine monadische Prozedur mit null Argumenten bezeichnen (siehe Abschnitt 9.11 [Die Store-Monade], Seite 170). Die Prozedur muss eine Ableitung als monadischen Wert zurückliefern, die dann durch `run-with-store` laufen gelassen wird.

`--source`
`-S` Die Quellcode-Ableitung der Pakete statt die Pakete selbst erstellen.

Zum Beispiel liefert `guix build -S gcc` etwas in der Art von `/gnu/store/...-gcc-4.7.2.tar.bz2`, also den Tarball mit dem GCC-Quellcode.

Der gelieferte Quell-Tarball ist das Ergebnis davon, alle Patches und Code-Schnipsel aufzuspielen, die im `origin`-Objekt des Pakets festgelegt wurden (siehe Abschnitt 9.2 [Pakete definieren], Seite 109).

Wie andere Arten von Ableitung kann auch das Ergebnis einer Quellcode-Ableitung mit der Befehlszeilenoption `--check` geprüft werden (siehe [build-check], Seite 201). Das ist nützlich, um zu überprüfen, ob ein (vielleicht bereits erstellter oder substituierter, also zwischengespeicherter) Paketquellcode zu ihrer deklarierten Hash-Prüfsumme passt.

Beachten Sie, dass `guix build -S` nur für die angegebenen Pakete den Quellcode herunterlädt. Dazu gehört *nicht* der Quellcode statisch gebundener Abhängigkeiten und der Quellcode alleine reicht nicht aus, um die Pakete zu reproduzieren.

`--sources`

Den Quellcode für *Paket-oder-Ableitung* und alle Abhängigkeiten davon rekursiv herunterladen und zurückliefern. Dies ist eine praktische Methode, eine lokale Kopie des gesamten Quellcodes zu beziehen, der nötig ist, um die Pakete zu erstellen, damit Sie diese später auch ohne Netzwerkzugang erstellen lassen können. Es handelt sich um eine Erweiterung der Befehlszeilenoption `--source`, die jeden der folgenden Argumentwerte akzeptiert:

`package` Mit diesem Wert verhält sich die Befehlszeilenoption `--sources` auf genau die gleiche Weise wie die Befehlszeilenoption `--source`.

`all` Erstellt die Quellcode-Ableitungen aller Pakete einschließlich allen Quellcodes, der als Teil der Eingaben im `inputs`-Feld aufgelistet ist. Dies ist der vorgegebene Wert, wenn sonst keiner angegeben wird.

```
$ guix build --sources tzdata
Folgende Ableitungen werden erstellt:
   /gnu/store/...-tzdata2015b.tar.gz.drv
   /gnu/store/...-tzcode2015b.tar.gz.drv
```

`transitive`
Die Quellcode-Ableitungen aller Pakete sowie aller transitiven Eingaben der Pakete erstellen. Damit kann z.B. Paket-Quellcode vorab heruntergeladen und später offline erstellt werden.

```
$ guix build --sources=transitive tzdata
Folgende Ableitungen werden erstellt:
   /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=`*System*

`-s` *System* Versuchen, für das angegebene *System* – z.B. `i686-linux` – statt für denselben Systemtyp wie auf dem Wirtssystem zu erstellen. Beim Befehl `guix build` können Sie diese Befehlszeilenoption mehrmals wiederholen, wodurch für jedes angegebene System eine Erstellung durchgeführt wird; andere Befehle ignorieren überzählige `-s`-Befehlszeilenoptionen.

> **Anmerkung:** Die Befehlszeilenoption `--system` dient der *nativen* Kompilierung (nicht zu verwechseln mit Cross-Kompilierung). Siehe `--target` unten für Informationen zur Cross-Kompilierung.

Ein Beispiel sind Linux-basierte Systeme, die verschiedene Persönlichkeiten emulieren können. Zum Beispiel können Sie `--system=i686-linux` auf einem `x86_64-linux`-System oder `--system=armhf-linux` auf einem `aarch64-linux`-System angeben, um Pakete in einer vollständigen 32-Bit-Umgebung zu erstellen.

> **Anmerkung:** Das Erstellen für ein `armhf-linux`-System ist ungeprüft auf allen `aarch64-linux`-Maschinen aktiviert, obwohl bestimmte aarch64-Chipsätze diese Funktionalität nicht unterstützen, darunter auch ThunderX.

Ebenso können Sie, wenn transparente Emulation mit QEMU und `binfmt_misc` aktiviert sind (siehe Abschnitt 12.9.29 [Virtualisierungsdienste], Seite 534), für jedes System Erstellungen durchführen, für das ein QEMU-`binfmt_misc`-Handler installiert ist.

Erstellungen für ein anderes System, das nicht dem System der Maschine, die Sie benutzen, entspricht, können auch auf eine entfernte Maschine mit der richtigen Architektur ausgelagert werden. Siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13, für mehr Informationen über das Auslagern.

`--target=`*Tripel*

Lässt für das angegebene *Tripel* cross-erstellen. Dieses muss ein gültiges GNU-Tripel wie z.B. `"aarch64-linux-gnu"` sein (siehe Abschnitt “Specifying Target Triplets” in *Autoconf*).

`--list-systems`

Listet alle unterstützten Systeme auf, die als Argument an `--system` gegeben werden können.

`--list-targets`

Listet alle unterstützten Ziele auf, die als Argument an `--target` gegeben werden können.

`--check` *Paket-oder-Ableitung* erneut erstellen, wenn diese bereits im Store verfügbar ist, und einen Fehler melden, wenn die Erstellungsergebnisse nicht Bit für Bit identisch sind.

Mit diesem Mechanismus können Sie überprüfen, ob zuvor installierte Substitute unverfälscht sind (siehe Abschnitt 6.3 [Substitute], Seite 56) oder auch ob das Erstellungsergebnis eines Pakets deterministisch ist. Siehe Abschnitt 10.12 [Aufruf von guix challenge], Seite 238, für mehr Hintergrundinformationen und Werkzeuge.

Wenn dies zusammen mit `--keep-failed` benutzt wird, bleiben die sich unterscheidenden Ausgaben im Store unter dem Namen `/gnu/store/...-check`. Dadurch können Unterschiede zwischen den beiden Ergebnissen leicht erkannt werden.

`--repair` Versuchen, die angegebenen Store-Objekte zu reparieren, wenn sie beschädigt sind, indem sie neu heruntergeladen oder neu erstellt werden.

Diese Operation ist nicht atomar und nur der Administratornutzer `root` kann sie verwenden.

`--derivations`
`-d` Liefert die Ableitungspfade und *nicht* die Ausgabepfade für die angegebenen Pakete.

`--root=`*Datei*
`-r` *Datei* Die *Datei* zu einer symbolischen Verknüpfung auf das Ergebnis machen und als Müllsammlerwurzel registrieren.

Dadurch wird das Ergebnis dieses Aufrufs von `guix build` vor dem Müllsammler geschützt, bis die *Datei* gelöscht wird. Wird diese Befehlszeilenoption *nicht* angegeben, können Erstellungsergebnisse vom Müllsammler geholt werden, sobald die Erstellung abgeschlossen ist. Siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61, für mehr Informationen zu Müllsammlerwurzeln.

`--log-file`
Liefert die Dateinamen oder URLs der Erstellungsprotokolle für das angegebene *Paket-oder-Ableitung* oder meldet einen Fehler, falls Protokolldateien fehlen.

Dies funktioniert, egal wie die Pakete oder Ableitungen angegeben werden. Zum Beispiel sind folgende Aufrufe alle äquivalent:

```
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)'
```

Wenn ein Protokoll lokal nicht verfügbar ist und sofern `--no-substitutes` nicht übergeben wurde, sucht der Befehl nach einem entsprechenden Protokoll auf einem der Substitutserver (die mit `--substitute-urls` angegeben werden können).

Stellen Sie sich zum Beispiel vor, sie wollten das Erstellungsprotokoll von GDB auf einem `aarch64`-System sehen, benutzen aber selbst eine `x86_64`-Maschine:

```
$ guix build --log-file gdb -s aarch64-linux
```

```
https://ci.guix.gnu.org/log/...-gdb-7.10
```

So haben Sie umsonst Zugriff auf eine riesige Bibliothek von Erstellungsprotokollen!

10.1.4 Fehlschläge beim Erstellen untersuchen

Wenn Sie ein neues Paket definieren (siehe Abschnitt 9.2 [Pakete definieren], Seite 109), werden Sie sich vermutlich einige Zeit mit der Fehlersuche beschäftigen und die Erstellung so lange anpassen, bis sie funktioniert. Dazu müssen Sie die Erstellungsbefehle selbst in einer Umgebung benutzen, die der, die der Erstellungsdaemon aufbaut, so ähnlich wie möglich ist.

Das Erste, was Sie dafür tun müssen, ist die Befehlszeilenoption `--keep-failed` oder `-K` von `guix build` einzusetzen, wodurch Verzeichnisbäume fehlgeschlagener Erstellungen in `/tmp` oder dem von Ihnen als `TMPDIR` ausgewiesenen Verzeichnis erhalten und nicht gelöscht werden (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

Im Anschluss können Sie mit `cd` in die Verzeichnisse dieses fehlgeschlagenen Erstellungsbaums wechseln und mit `source` dessen `environment-variables`-Datei laden, die alle Umgebungsvariablendefinitionen enthält, die zum Zeitpunkt des Fehlschlags der Erstellung galten. Sagen wir, Sie suchen Fehler in einem Paket `foo`, dann würde eine typische Sitzung so aussehen:

```
$ guix build foo -K
... Erstellung schlägt fehl
$ cd /tmp/guix-build-foo.drv-0
$ source ./environment-variables
$ cd foo-1.2
```

Nun können Sie Befehle (fast) so aufrufen, als wären Sie der Daemon, und Fehlerursachen in Ihrem Erstellungsprozess ermitteln.

Manchmal passiert es, dass zum Beispiel die Tests eines Pakets erfolgreich sind, wenn Sie sie manuell aufrufen, aber scheitern, wenn der Daemon sie ausführt. Das kann passieren, weil der Daemon Erstellungen in isolierten Umgebungen („Containern“) durchführt, wo, anders als in der obigen Umgebung, kein Netzwerkzugang möglich ist, `/bin/sh` nicht exisiert usw. (siehe Abschnitt 2.4.1 [Einrichten der Erstellungsumgebung], Seite 11).

In solchen Fällen müssen Sie den Erstellungsprozess womöglich aus einer zu der des Daemons ähnlichen isolierten Umgebung heraus ausprobieren:

```
$ 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
```

Hierbei erzeugt `guix shell -C` eine isolierte Umgebung und öffnet darin eine Shell (siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86). Der Teil mit `strace gdb` fügt die Befehle `strace` und `gdb` zur isolierten Umgebung hinzu, die Sie gut gebrauchen können, während Sie Fehler suchen. Wegen der Befehlszeilenoption `--no-grafts` bekommen Sie haargenau dieselbe Umgebung ohne veredelte Pakete (siehe Kapitel 19 [Sicherheitsaktualisierungen], Seite 697, für mehr Informationen zu Veredelungen).

Um der isolierten Umgebung des Erstellungsdaemons noch näher zu kommen, können wir `/bin/sh` entfernen:

```
[env]# rm /bin/sh
```

(Keine Sorge, das ist harmlos: All dies passiert nur in der zuvor von `guix shell` erzeugten Wegwerf-Umgebung.)

Der Befehl `strace` befindet sich wahrscheinlich nicht in Ihrem Suchpfad, aber wir können ihn so benutzen:

```
[env]# $GUIX_ENVIRONMENT/bin/strace -f -o log make check
```

Auf diese Weise haben Sie nicht nur die Umgebungsvariablen, die der Daemon benutzt, nachgebildet, sondern lassen auch den Erstellungsprozess in einer isolierten Umgebung ähnlich der des Daemons laufen.

10.2 `guix edit` aufrufen

So viele Pakete, so viele Quelldateien! Der Befehl `guix edit` erleichtert das Leben von sowohl Nutzern als auch Paketentwicklern, indem er Ihren Editor anweist, die Quelldatei mit der Definition des jeweiligen Pakets zu bearbeiten. Zum Beispiel startet dies:

```
guix edit gcc@4.9 vim
```

das mit der Umgebungsvariablen `VISUAL` oder `EDITOR` angegebene Programm und lässt es das Rezept von GCC 4.9.3 und von Vim anzeigen.

Wenn Sie ein Git-Checkout von Guix benutzen (siehe Abschnitt 22.1 [Erstellung aus dem Git], Seite 708) oder Ihre eigenen Pakete im `GUIX_PACKAGE_PATH` erstellt haben (siehe Abschnitt 9.1 [Paketmodule], Seite 108), werden Sie damit die Paketrezepte auch bearbeiten können. Andernfalls werden Sie zumindest in die Lage versetzt, die nur lesbaren Rezepte für sich im Moment im Store befindliche Pakete zu untersuchen.

Statt `GUIX_PACKAGE_PATH` zu benutzen, können Sie mit der Befehlszeilenoption `--load-path=`*`Verzeichnis`* (oder kurz `-L` *`Verzeichnis`*) das *Verzeichnis* vorne an den Paketmodul-Suchpfad anhängen und so Ihre eigenen Pakete sichtbar machen.

10.3 `guix download` aufrufen

Wenn Entwickler einer Paketdefinition selbige schreiben, müssen diese normalerweise einen Quellcode-Tarball herunterladen, seinen SHA256-Hash als Prüfsumme berechnen und diese in der Paketdefinition eintragen (siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Das Werkzeug `guix download` hilft bei dieser Aufgabe: Damit wird eine Datei von der angegebenen URI heruntergeladen, in den Store eingelagert und sowohl ihr Dateiname im Store als auch ihr SHA256-Hash als Prüfsumme angezeigt.

Dadurch, dass die heruntergeladene Datei in den Store eingefügt wird, wird Bandbreite gespart: Wenn der Entwickler schließlich versucht, das neu definierte Paket mit `guix build` zu erstellen, muss der Quell-Tarball nicht erneut heruntergeladen werden, weil er sich bereits im Store befindet. Es ist auch eine bequeme Methode, Dateien temporär aufzubewahren, die letztlich irgendwann gelöscht werden (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61).

Der Befehl `guix download` unterstützt dieselben URIs, die in Paketdefinitionen verwendet werden. Insbesondere unterstützt er `mirror://`-URIs. `https`-URIs (HTTP über TLS)

werden unterstützt, *vorausgesetzt* die Guile-Anbindungen für GnuTLS sind in der Umgebung des Benutzers verfügbar; wenn nicht, wird ein Fehler gemeldet. Siehe die Abschnitt "Guile Preparations" in *GnuTLS-Guile* für mehr Informationen.

Mit `guix download` werden HTTPS-Serverzertifikate verifiziert, indem die Zertifikate der X.509-Autoritäten in das durch die Umgebungsvariable `SSL_CERT_DIR` bezeichnete Verzeichnis heruntergeladen werden (siehe Abschnitt 12.11 [X.509-Zertifikate], Seite 606), außer `--no-check-certificate` wird benutzt.

Folgende Befehlszeilenoptionen stehen zur Verfügung:

`--hash=`*Algorithmus*
`-H` *Algorithmus*
: Einen Hash mit dem angegebenen *Algorithmus* berechnen. Siehe Abschnitt 10.4 [Aufruf von guix hash], Seite 205, für weitere Informationen.

`--format=`*Format*
`-f` *Format*
: Die Hash-Prüfsumme im angegebenen *Format* ausgeben. Für weitere Informationen, was gültige Werte für das *Format* sind, siehe Abschnitt 10.4 [Aufruf von guix hash], Seite 205.

`--no-check-certificate`
: X.509-Zertifikate von HTTPS-Servern *nicht* validieren.

  Wenn Sie diese Befehlszeilenoption benutzen, haben Sie *keinerlei Garantie*, dass Sie tatsächlich mit dem authentischen Server, der für die angegebene URL verantwortlich ist, kommunizieren. Das macht Sie anfällig gegen sogenannte „Man-in-the-Middle"-Angriffe.

`--output=`*Datei*
`-o` *Datei*
: Die heruntergeladene Datei *nicht* in den Store, sondern in die angegebene *Datei* abspeichern.

10.4 `guix hash` aufrufen

Der Befehl `guix hash` berechnet den Hash einer oder mehrerer Datei. Er ist primär ein Werkzeug, dass es bequemer macht, etwas zur Distribution beizusteuern: Damit wird die kryptografische Hash-Prüfsumme berechnet, die bei der Definition eines Pakets benutzt werden kann (siehe Abschnitt 9.2 [Pakete definieren], Seite 109).

Die allgemeine Syntax lautet:

```
guix hash Option Datei ...
```

Wird als *Datei* ein Bindestrich `-` angegeben, berechnet `guix hash` den Hash der von der Standardeingabe gelesenen Daten. `guix hash` unterstützt die folgenden Optionen:

`--hash=`*Algorithmus*
`-H` *Algorithmus*
: Mit dem angegebenen *Algorithmus* einen Hash berechnen. Die Vorgabe ist, `sha256` zu benutzen.

  Algorithmus muss der Name eines durch Libgcrypt über Guile-Gcrypt zur Verfügung gestellten kryptografischen Hashalgorithmus sein, z.B. `sha512` oder `sha3-256` (siehe Abschnitt "Hash Functions" in *Referenzhandbuch zu Guile-Gcrypt*).

`--format=`*Format*
`-f` *Format* Gibt die Prüfsumme im angegebenen *Format* aus.

Unterstützte Formate: `base64`, `nix-base32`, `base32`, `base16` (`hex` und `hexadecimal` können auch benutzt werden).

Wird keine Befehlszeilenoption `--format` angegeben, wird `guix hash` die Prüfsumme im `nix-base32`-Format ausgeben. Diese Darstellung wird bei der Definition von Paketen benutzt.

`--recursive`
`-r` Die Befehlszeilenoption `--recursive` ist veraltet. Benutzen Sie `--serializer=nar` (siehe unten). `-r` bleibt als bequeme Kurzschreibweise erhalten.

`--serializer=`*Typ*
`-S` *Typ* Die Prüfsumme der *Datei* auf die durch *Typ* angegebene Art berechnen.

Als *Typ* können Sie einen hiervon benutzen:

`none` Dies entspricht der Vorgabe: Die Prüfsumme des Inhalts der Datei wird berechnet.

`nar` In diesem Fall wird die Prüfsumme eines Normalisierten Archivs (kurz „Nar“) berechnet, das die *Datei* enthält, und auch ihre Kinder, wenn es sich um ein Verzeichnis handelt. Einige der Metadaten der *Datei* sind Teil dieses Archivs. Zum Beispiel unterscheidet sich die berechnete Prüfsumme, wenn die *Datei* eine reguläre Datei ist, je nachdem, ob die *Datei* ausführbar ist oder nicht. Metadaten wie der Zeitstempel haben keinen Einfluss auf die Prüfsumme (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74, für mehr Informationen über das Nar-Format).

`git` Die Prüfsumme der Datei oder des Verzeichnisses als Git-Baumstruktur berechnen, nach derselben Methode wie beim Git-Versionskontrollsystem.

`--exclude-vcs`
`-x` Wenn dies zusammen mit der Befehlszeilenoption `--recursive` angegeben wird, werden Verzeichnisse zur Versionskontrolle (`.bzr`, `.git`, `.hg`, etc.) vom Archiv ausgenommen.

Zum Beispiel würden Sie auf diese Art die Prüfsumme eines Git-Checkouts berechnen, was nützlich ist, wenn Sie die Prüfsumme für die Methode `git-fetch` benutzen (siehe Abschnitt 9.2.2 [„origin“-Referenz], Seite 118):

```
$ git clone http://example.org/foo.git
$ cd foo
$ guix hash -x --serializer=nar .
```

10.5 `guix import` aufrufen

Der Befehl `guix import` ist für Leute hilfreich, die ein Paket gerne mit so wenig Arbeit wie möglich zur Distribution hinzufügen würden – ein legitimer Anspruch. Der Befehl kennt

ein paar Sammlungen, aus denen mit ihm Paketmetadaten „importiert“ werden können. Das Ergebnis ist eine Paketdefinition oder eine Vorlage dafür in dem uns bekannten Format (siehe Abschnitt 9.2 [Pakete definieren], Seite 109).

Die allgemeine Syntax lautet:

```
guix import Importer Optionen...
```

Der *Importer* gibt die Quelle an, aus der Paketmetadaten importiert werden, und die *Optionen* geben eine Paketbezeichnung und andere vom *Importer* abhängige Daten an.

Manche Importer setzen voraus, dass der Befehl `gpgv` ausgeführt werden kann. Sie funktionieren nur, wenn GnuPG installiert und im `$PATH` enthalten ist; falls nötig können Sie `guix install gnupg` ausführen.

Derzeit sind folgende „Importer“ verfügbar:

`gnu` Metadaten für das angegebene GNU-Paket importieren. Damit wird eine Vorlage für die neueste Version dieses GNU-Pakets zur Verfügung gestellt, einschließlich der Prüfsumme seines Quellcode-Tarballs, seiner kanonischen Zusammenfassung und seiner Beschreibung.

Zusätzliche Informationen wie Paketabhängigkeiten und seine Lizenz müssen noch manuell ermittelt werden.

Zum Beispiel liefert der folgende Befehl eine Paketdefinition für GNU Hello:

```
guix import gnu hello
```

Speziell für diesen Importer stehen noch folgende Befehlszeilenoptionen zur Verfügung:

`--key-download=`*`Richtlinie`*
: Die Richtlinie zum Umgang mit fehlenden OpenPGP-Schlüsseln beim Verifizieren der Paketsignatur (auch „Beglaubigung“ genannt) festlegen, wie bei `guix refresh`. Siehe Abschnitt 10.6 [Aufruf von guix refresh], Seite 214.

`pypi` Metadaten aus dem Python Package Index (`https://pypi.python.org/`) importieren. Informationen stammen aus der JSON-formatierten Beschreibung, die unter `pypi.python.org` verfügbar ist, und enthalten meistens alle relevanten Informationen einschließlich der Abhängigkeiten des Pakets. Für maximale Effizienz wird empfohlen, das Hilfsprogramm `unzip` zu installieren, damit der Importer „Python Wheels“ entpacken und daraus Daten beziehen kann.

Der folgende Befehl importiert Metadaten für die neueste Version des Python-Pakets namens `itsdangerous`:

```
guix import pypi itsdangerous
```

Sie können auch um eine bestimmte Version bitten:

```
guix import pypi itsdangerous@1.1.0
```

`--recursive`
`-r`
: Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`gem` Metadaten von RubyGems (`https://rubygems.org/`) importieren. Informationen kommen aus der JSON-formatierten Beschreibung, die auf `rubygems.org` verfügbar ist, und enthält die relevantesten Informationen einschließlich der Laufzeitabhängigkeiten. Dies hat aber auch Schattenseiten – die Metadaten unterscheiden nicht zwischen Zusammenfassungen und Beschreibungen, daher wird dieselbe Zeichenkette für beides eingesetzt. Zudem fehlen Informationen zu nicht in Ruby geschriebenen Abhängigkeiten, die benötigt werden, um native Erweiterungen zu in Ruby geschriebenem Code zu erstellen. Diese herauszufinden bleibt dem Paketentwickler überlassen.

Der folgende Befehl importiert Metadaten aus dem Ruby-Paket `rails`.

```
guix import gem rails
```

Sie können auch um eine bestimmte Version bitten:

```
guix import gem rails@7.0.4
```

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`minetest` Importiert Metadaten aus der ContentDB (`https://content.minetest.net`). Informationen werden aus den JSON-formatierten Metadaten genommen, die über die Programmierschnittstelle („API“) von ContentDB (`https://content.minetest.net/help/api/`) angeboten werden, und enthalten die relevantesten Informationen wie zum Beispiel Abhängigkeiten. Allerdings passt das Ergebnis nicht ganz. Lizenzinformationen sind oft unvollständig. Der Commit-Hash fehlt manchmal. Die importierten Beschreibungen sind in Markdown formatiert, aber Guix braucht stattdessen Texinfo-Auszeichnungen. Texturpakete und Teilspiele werden nicht unterstützt.

Der folgende Befehl importiert Metadaten für die Mesecons-Mod von Jeija:

```
guix import minetest Jeija/mesecons
```

Man muss den Autorennamen nicht angeben:

```
guix import minetest mesecons
```

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`cpan` Importiert Metadaten von MetaCPAN (`https://www.metacpan.org/`). Informationen werden aus den JSON-formatierten Metadaten genommen, die über die Programmierschnittstelle („API“) von MetaCPAN (`https://fastapi.metacpan.org/`) angeboten werden, und enthalten die relevantesten Informationen wie zum Beispiel Modulabhängigkeiten. Lizenzinformationen sollten genau nachgeprüft werden. Wenn Perl im Store verfügbar ist, wird das Werkzeug `corelist` benutzt, um Kernmodule in der Abhängigkeitsliste wegzulassen.

Folgender Befehl importiert Metadaten für das Perl-Modul Acme::Boolean:

```
guix import cpan Acme::Boolean
```

`cran` Metadaten aus dem CRAN (`https://cran.r-project.org/`) importieren, der zentralen Sammlung für die statistische und grafische Umgebung GNU R (`https://r-project.org`).

Informationen werden aus der Datei namens `DESCRIPTION` des Pakets extrahiert.

Der folgende Befehl importiert Metadaten für das Cairo-R-Paket:

```
guix import cran Cairo
```

Sie können auch um eine bestimmte Version bitten:

```
guix import cran rasterVis@0.50.3
```

Wird zudem `--recursive` angegeben, wird der Importer den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für all die Pakete erzeugen, die noch nicht Teil von Guix sind.

Wird `--style=specification` angegeben, wird der Importer Paketdefinitionen erzeugen, deren Eingaben als Paketspezifikationen statt als Referenzen auf Paketvariable vorliegen. Das ist nützlich, wenn die erzeugten Paketdefinitionen in bestehende, nutzereigene Module eingefügt werden, weil dann die Liste der benutzten Paketmodule nicht angepasst werden muss. Die Vorgabe ist `--style=variable`.

Wird `--archive=bioconductor` angegeben, werden Metadaten vom Bioconductor (`https://www.bioconductor.org/`) importiert, einer Sammlung von R-Paketen zur Analyse und zum Verständnis von großen Mengen genetischer Daten in der Bioinformatik.

Informationen werden aus der Datei namens `DESCRIPTION` im Archiv des Pakets extrahiert.

Der folgende Befehl importiert Metadaten für das R-Paket GenomicRanges:

```
guix import cran --archive=bioconductor GenomicRanges
```

Schließlich können Sie auch solche R-Pakete importieren, die noch nicht auf CRAN oder im Bioconductor veröffentlicht wurden, solange sie in einem Git-Repository stehen. Benutzen Sie `--archive=git` gefolgt von der URL des Git-Repositorys.

```
guix import cran --archive=git https://github.com/immunogenomics/harmony
```

`texlive` Informationen über TeX-Pakete, die Teil der TeX-Live-Distribution (`https://www.tug.org/texlive/`) sind, aus der Datenbank von TeX Live importieren.

Paketinformationen werden der Paketdatenbank von TeX Live entnommen. Bei ihr handelt es sich um eine reine Textdatei, die im `texlive-bin`-Paket enthalten ist. Der Quellcode wird von unter Umständen mehreren Stellen im SVN-Repository des TeX-Live-Projekts heruntergeladen.

Der folgende Befehl importiert Metadaten für das TeX-Paket `fontspec`:

```
guix import texlive fontspec
```

`json` Paketmetadaten aus einer lokalen JSON-Datei importieren. Betrachten Sie folgende Beispiel-Paketdefinition im JSON-Format:

```
{
```

```
  "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"]
}
```

Die Felder sind genauso benannt wie bei einem `<package>`-Verbundstyp (siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Referenzen zu anderen Paketen stehen darin als JSON-Liste von mit Anführungszeichen quotierten Zeichenketten wie `guile` oder `guile@2.0`.

Der Importer unterstützt auch eine ausdrücklichere Definition der Quelldateien mit den üblichen Feldern eines `<origin>`-Verbunds:

```
{
  ...
  "source": {
    "method": "url-fetch",
    "uri": "mirror://gnu/hello/hello-2.10.tar.gz",
    "sha256": {
      "base32": "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kzl7c9lng89ndq1i"
    }
  }
  ...
}
```

Der folgende Befehl liest Metadaten aus der JSON-Datei `hello.json` und gibt einen Paketausdruck aus:

```
guix import json hello.json
```

`hackage` Metadaten aus Hackage (`https://hackage.haskell.org/`), dem zentralen Paketarchiv der Haskell-Gemeinde, importieren. Informationen werden aus Cabal-Dateien ausgelesen. Darin sind alle relevanten Informationen einschließlich der Paketabhängigkeiten enthalten.

Speziell für diesen Importer stehen noch folgende Befehlszeilenoptionen zur Verfügung:

`--stdin`
`-s` Eine Cabal-Datei von der Standardeingabe lesen.

`--no-test-dependencies`
`-t` Keine Abhängigkeiten übernehmen, die nur von Testkatalogen benötigt werden.

`--cabal-environment=`*`Aliste`*
`-e` *`Aliste`* *Aliste* muss eine assoziative Liste der Scheme-Programmiersprache sein, die die Umgebung definiert, in der bedingte Ausdrücke von

Cabal ausgewertet werden. Dabei werden folgende Schlüssel akzeptiert: `os`, `arch`, `impl` und eine Zeichenkette, die dem Namen einer Option (einer „Flag“) entspricht. Der mit einer „Flag“ assoziierte Wert muss entweder das Symbol `true` oder `false` sein. Der anderen Schlüsseln zugeordnete Wert muss mit der Definition des Cabal-Dateiformats konform sein. Der vorgegebene Wert zu den Schlüsseln `os`, `arch` and `impl` ist jeweils ‘`linux`’, ‘`x86_64`’ bzw. ‘`ghc`’.

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

Der folgende Befehl importiert Metadaten für die neuste Version des Haskell-„HTTP“-Pakets, ohne Testabhängigkeiten zu übernehmen und bei Übergabe von `false` als Wert der Flag ‘`network-uri`’:

```
guix import hackage -t -e "'((\"network-uri\" . false))" HTTP
```

Eine ganz bestimmte Paketversion kann optional ausgewählt werden, indem man nach dem Paketnamen anschließend ein At-Zeichen und eine Versionsnummer angibt wie in folgendem Beispiel:

```
guix import hackage mtl@2.1.3.1
```

`stackage` Der `stackage`-Importer ist ein Wrapper um den `hackage`-Importer. Er nimmt einen Paketnamen und schaut dafür die Paketversion nach, die Teil einer Stackage-Veröffentlichung (`https://www.stackage.org`) mit Langzeitunterstützung (englisch „Long-Term Support“, kurz LTS) ist, deren Metadaten er dann mit dem `hackage`-Importer bezieht. Beachten Sie, dass es Ihre Aufgabe ist, eine LTS-Veröffentlichung auszuwählen, die mit dem von Guix benutzten GHC-Compiler kompatibel ist.

Speziell für diesen Importer stehen noch folgende Befehlszeilenoptionen zur Verfügung:

`--no-test-dependencies`
`-t` Keine Abhängigkeiten übernehmen, die nur von Testkatalogen benötigt werden.

`--lts-version=`*Version*
`-l` *Version*
Version ist die gewünschte Version der LTS-Veröffentlichung. Wird keine angegeben, wird die neueste benutzt.

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

Der folgende Befehl importiert Metadaten für dasjenige Haskell-„HTTP“-Paket, das in der LTS-Stackage-Veröffentlichung mit Version 7.18 vorkommt:

```
guix import stackage --lts-version=7.18 HTTP
```

`elpa` Metadaten aus der Paketsammlung „Emacs Lisp Package Archive“ (ELPA) importieren (siehe Abschnitt “Packages” in *The GNU Emacs Manual*).

Speziell für diesen Importer stehen noch folgende Befehlszeilenoptionen zur Verfügung:

`--archive=`*Repo*
`-a` *Repo* Mit *Repo* wird die Archiv-Sammlung (ein „Repository“) bezeichnet, von dem die Informationen bezogen werden sollen. Derzeit sind die unterstützten Repositorys und ihre Bezeichnungen folgende:

- GNU (`https://elpa.gnu.org/packages`), bezeichnet mit `gnu`. Dies ist die Vorgabe.

  Pakete aus `elpa.gnu.org` wurden mit einem der Schlüssel im GnuPG-Schlüsselbund in `share/emacs/25.1/etc/package-keyring.gpg` (oder einem ähnlichen Pfad) des `emacs`-Pakets signiert (siehe Abschnitt “Package Installation” in *The GNU Emacs Manual*).
- NonGNU (`https://elpa.nongnu.org/nongnu/`), bezeichnet mit `nongnu`.
- MELPA-Stable (`https://stable.melpa.org/packages`), bezeichnet mit `melpa-stable`.
- MELPA (`https://melpa.org/packages`), bezeichnet mit `melpa`.

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`crate` Metadaten aus der Paketsammlung crates.io für Rust crates.io (`https://crates.io`) importieren, wie Sie in diesem Beispiel sehen:

```
guix import crate blake2-rfc
```

Mit dem Crate-Importer können Sie auch eine Version als Zeichenkette angeben:

```
guix import crate constant-time-eq@0.1.0
```

Zu den zusätzlichen Optionen gehören:

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`elm` Metadaten aus der Paketsammlung package.elm-lang.org (`https://package.elm-lang.org`) für Elm importieren, wie Sie in diesem Beispiel sehen:

```
guix import elm elm-explorations/webgl
```

Mit dem Elm-Importer können Sie auch eine Version als Zeichenkette angeben:

```
guix import elm elm-explorations/webgl@1.1.3
```

Zu den zusätzlichen Optionen gehören:

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`opam` Metadaten aus der Paketsammlung OPAM (`https://opam.ocaml.org/`) der OCaml-Gemeinde importieren.

Zu den zusätzlichen Optionen gehören:

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`--repo` Vorgegeben ist, nach Paketen im offiziellen OPAM-Repository zu suchen. Mit dieser Befehlszeilenoption, die mehr als einmal angegeben werden kann, können Sie andere Repositorys hinzufügen, in denen nach Paketen gesucht wird. Als gültige Argumente akzeptiert werden:

- der Name eines bekannten Repositorys, entweder `opam`, `coq` (äquivalent zu `coq-released`), `coq-core-dev`, `coq-extra-dev` oder `grew`.
- die URL eines Repository, wie sie der Befehl `opam repository add` erfordert (zum Beispiel wäre `https://opam.ocaml.org` die URL, die dem Namen `opam` oben entspricht).
- den Pfad zur lokalen Kopie eines Repositorys (ein Verzeichnis mit einem Unterverzeichnis `packages/`).

Die an diese Befehlszeilenoption übergebenen Repositorys sind in der Reihenfolge Ihrer Präferenz anzugeben. Die zusätzlich angegebenen Repositorys ersetzen *nicht* das voreingestellte Repository `opam`; es bleibt immer als letzte Möglichkeit stehen.

Beachten Sie ebenso, dass Versionsnummern mehrerer Repositorys *nicht* verglichen werden, sondern einfach das Paket aus dem ersten Repository (von links nach rechts), das mindestens eine Version des angeforderten Pakets enthält, genommen wird und die importierte Version der neuesten *aus nur diesem Repository* entspricht.

`go` Metadaten für ein Go-Modul von proxy.golang.org (`https://proxy.golang.org`) importieren.

```
guix import go gopkg.in/yaml.v2
```

Es ist möglich, bei der Paketspezifikation ein Suffix `@VERSION` anzugeben, um diese bestimmte Version zu importieren.

Zu den zusätzlichen Optionen gehören:

`--recursive`
`-r` Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`--pin-versions`
: Wenn Sie diese Option verwenden, behält der Importer genau diese Version der Go-Modul-Abhängigkeiten bei, statt die neuesten verfügbaren Versionen zu benutzen. Das kann hilfreich sein, wenn Sie versuchen, Pakete zu importieren, die für deren Erstellung rekursiv von alten Versionen ihrer selbst abhängen. Wenn Sie diesen Modus nutzen, wird das Symbol für das Paket durch Anhängen der Versionsnummer an seinen Namen gebildet, damit mehrere Versionen desselben Pakets gleichzeitig existieren können.

`egg`
: Metadaten für CHICKEN-Eggs (`https://wiki.call-cc.org/eggs`) importieren. Die Informationen werden aus den `PAKET.egg`-Dateien genommen, die im Git-Repository eggs-5-all (`git://code.call-cc.org/eggs-5-all`) stehen. Jedoch gibt es dort nicht alle Informationen, die wir brauchen: Ein Feld für die Beschreibung (`description`) fehlt und die benutzten Lizenzen sind ungenau (oft wird BSD angegeben statt BSD-N).

```
guix import egg sourcehut
```

Sie können auch um eine bestimmte Version bitten:

```
guix import egg arrays@1.0
```

Zu den zusätzlichen Optionen gehören:

`--recursive`
`-r`
: Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`hexpm`
: Metadaten aus der Paketsammlung hex.pm für Erlang und Elixir hex.pm (`https://hex.pm`) importieren, wie Sie in diesem Beispiel sehen:

```
guix import hexpm stun
```

Der Importer versucht, das richtige Erstellungssystem für das Paket zu erkennen.

Mit dem hexpm-Importer können Sie auch eine Version als Zeichenkette angeben:

```
guix import hexpm cf@0.3.0
```

Zu den zusätzlichen Optionen gehören:

`--recursive`
`-r`
: Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

`guix import` verfügt über eine modulare Code-Struktur. Mehr Importer für andere Paketformate zu haben, wäre nützlich, und Ihre Hilfe ist hierbei gerne gesehen (siehe Kapitel 22 [Mitwirken], Seite 708).

10.6 `guix refresh` aufrufen

Die Zielgruppe des Befehls `guix refresh` zum Auffrischen von Paketen sind in erster Linie Paketautoren. Als Nutzer könnten Sie an der Befehlszeilenoption `--with-latest` Interesse

haben, die auf `guix refresh` aufbaut, um Ihnen Superkräfte bei der Paketaktualisierung zu verleihen (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192). Nach Vorgabe werden mit `guix refresh` alle Pakete in der Distribution gemeldet, die nicht der neuesten Version des Anbieters entsprechen, indem Sie dies ausführen:

```
$ 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
```

Alternativ können die zu betrachtenden Pakete dabei angegeben werden, was zur Ausgabe einer Warnung führt, wenn es für Pakete kein Aktualisierungsprogramm gibt:

```
$ 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` durchsucht die Paketsammlung beim Anbieter jedes Pakets und bestimmt, was die höchste Versionsnummer ist, zu der es dort eine Veröffentlichung gibt. Zum Befehl gehören Aktualisierungsprogramme, mit denen bestimmte Typen von Paketen automatisch aktualisiert werden können: GNU-Pakete, ELPA-Pakete usw. – siehe die Dokumentation von `--type` unten. Es gibt jedoch auch viele Pakete, für die noch keine Methode enthalten ist, um das Vorhandensein einer neuen Veröffentlichung zu prüfen. Der Mechanismus ist aber erweiterbar, also können Sie gerne mit uns in Kontakt treten, wenn Sie eine neue Methode hinzufügen möchten!

`--recursive`
: Hiermit werden die angegebenen Pakete betrachtet und außerdem alle Pakete, von denen sie abhängen.

```
$ 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
...
```

Manchmal unterscheidet sich der vom Anbieter benutzte Name von dem Paketnamen, der in Guix verwendet wird, so dass `guix refresh` etwas Unterstützung braucht. Die meisten Aktualisierungsprogramme folgen der Eigenschaft `upstream-name` in Paketdefinitionen, die diese Unterstützung bieten kann.

```
(define-public network-manager
  (package
    (name "network-manager")
    ;; ...
    (properties '((upstream-name . "NetworkManager")))))
```

Wenn `--update` übergeben wird, werden die Quelldateien der Distribution verändert, so dass für diese Paketrezepte die aktuelle Version und die aktuelle Hash-Prüfsumme des Quellcode-Tarballs eingetragen wird (siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Dazu werden der neueste Quellcode-Tarball jedes Pakets sowie die jeweils zugehörige OpenPGP-Signatur heruntergeladen; mit Letzterer wird der heruntergeladene Tarball gegen seine Signatur mit `gpgv` authentifiziert und schließlich dessen Hash berechnet.

Beachten Sie, dass GnuPG dazu installiert sein und in `$PATH` vorkommen muss. Falls dies nicht der Fall ist, führen Sie `guix install gnupg` aus.

Wenn der öffentliche Schlüssel, mit dem der Tarball signiert wurde, im Schlüsselbund des Benutzers fehlt, wird versucht, ihn automatisch von einem Schlüssel-Server zu holen. Wenn das klappt, wird der Schlüssel zum Schlüsselbund des Benutzers hinzugefügt, ansonsten meldet `guix refresh` einen Fehler.

Die folgenden Befehlszeilenoptionen werden unterstützt:

`--expression=`*Ausdruck*
`-e` *Ausdruck*
: Als Paket benutzen, wozu der *Ausdruck* ausgewertet wird.

  Dies ist nützlich, um genau ein bestimmtes Paket zu referenzieren, wie in diesem Beispiel:

  ```
  guix refresh -l -e '(@@ (gnu packages commencement) glibc-final)'
  ```

  Dieser Befehls listet auf, was alles von der „endgültigen“ Erstellung von libc abhängt (praktisch alle Pakete).

`--update`
`-u`
: Die Quelldateien der Distribution (die Paketrezepte) werden direkt „in place“ verändert. Normalerweise führen Sie dies aus einem Checkout des Guix-Quellbaums heraus aus (siehe Abschnitt 22.2 [Guix vor der Installation ausführen], Seite 710):

  ```
  $ ./pre-inst-env guix refresh -s non-core -u
  ```

  Siehe Abschnitt 9.2 [Pakete definieren], Seite 109, für mehr Informationen zu Paketdefinitionen.

`--select=[`*Teilmenge*`]`
`-s` *Teilmenge*
: Wählt alle Pakete aus der *Teilmenge* aus, die entweder `core` oder `non-core` sein muss.

  Die `core`-Teilmenge bezieht sich auf alle Pakete, die den Kern der Distribution ausmachen, d.h. Pakete, aus denen heraus „alles andere“ erstellt wird. Dazu gehören GCC, libc, Binutils, Bash und so weiter. In der Regel ist die Folge einer Änderung an einem dieser Pakete in der Distribution, dass alle anderen neu erstellt werden müssen. Daher sind solche Änderungen unangenehm für Nutzer, weil sie einiges an Erstellungszeit oder Bandbreite investieren müssen, um die Aktualisierung abzuschließen.

  Die `non-core`-Teilmenge bezieht sich auf die übrigen Pakete. Sie wird typischerweise dann benutzt, wenn eine Aktualisierung der Kernpakete zu viele Umstände machen würde.

`--manifest=`*Datei*
`-m` *Datei*
: Wählt alle Pakete im in der *Datei* stehenden Manifest aus. Das ist nützlich, um zu überprüfen, welche Pakete aus dem Manifest des Nutzers aktualisiert werden können.

`--type=`*Aktualisierungsprogramm*
`-t` *Aktualisierungsprogramm*

Nur solche Pakete auswählen, die vom angegebenen *Aktualisierungsprogramm* behandelt werden. Es darf auch eine kommagetrennte Liste mehrerer Aktualisierungsprogramme angegeben werden. Zurzeit kann als *Aktualisierungsprogramm* eines der folgenden angegeben werden:

`gnu` Aktualisierungsprogramm für GNU-Pakete,

`savannah` Aktualisierungsprogramm auf Savannah (`https://savannah.gnu.org`) angebotener Pakete,

`sourceforge` Aktualisierungsprogramm auf SourceForge (`https://sourceforge.net`) angebotener Pakete,

`gnome` Aktualisierungsprogramm für GNOME-Pakete,

`kde` Aktualisierungsprogramm für KDE-Pakete,

`xorg` Aktualisierungsprogramm für X.org-Pakete,

`kernel.org` Aktualisierungsprogramm auf kernel.org angebotener Pakete,

`egg` Aktualisierungsprogramm für Egg-Pakete (`https://wiki.call-cc.org/eggs/`),

`elpa` Aktualisierungsprogramm für ELPA-Pakete (`https://elpa.gnu.org/`),

`cran` Aktualisierungsprogramm für CRAN-Pakete (`https://cran.r-project.org/`),

`bioconductor` Aktualisierungsprogramm für R-Pakete vom Bioconductor (`https://www.bioconductor.org/`),

`cpan` Aktualisierungsprogramm für CPAN-Pakete (`https://www.cpan.org/`),

`pypi` Aktualisierungsprogramm für PyPI-Pakete (`https://pypi.python.org`),

`gem` Aktualisierungsprogramm für RubyGems-Pakete (`https://rubygems.org`).

`github` Aktualisierungsprogramm für GitHub-Pakete (`https://github.com`).

`hackage` Aktualisierungsprogramm für Hackage-Pakete (`https://hackage.haskell.org`).

`stackage` Aktualisierungsprogramm für Stackage-Pakete (`https://www.stackage.org`).

`crate` Aktualisierungsprogramm für Crates-Pakete (`https://crates.io`).

`launchpad`
: Aktualisierungsprogramm für Launchpad (`https://launchpad.net`).

`generic-html`
: allgemeines Aktualisierungsprogramm, das mit einem Webcrawler die HTML-Seite, auf der der Quell-Tarball des Pakets angeboten wird, falls vorhanden, durchsucht.

`generic-git`
: allgemeines Aktualisierungsprogramm, das auf in Git-Repositorys angebotene Pakete anwendbar ist. Es wird versucht, Versionen anhand der Namen von Git-Tags zu erkennen, aber wenn Tag-Namen nicht korrekt zerteilt und verglichen werden, können Anwender die folgenden Eigenschaften im Paket festlegen.

  - `release-tag-prefix`: einen regulären Ausdruck, der zum Präfix des Tag-Namens passt.
  - `release-tag-suffix`: einen regulären Ausdruck, der zum Suffix des Tag-Namens passt.
  - `release-tag-version-delimiter`: eine Zeichenkette, die im Tag-Namen die Zahlen trennt, aus denen sich die Version zusammensetzt.
  - `accept-pre-releases`: ob Vorabveröffentlichungen berücksichtigt werden sollen. Vorgegeben ist, sie zu ignorieren. Setzen Sie dies auf `#t`, um sie hinzuzunehmen.

```
(package
  (name "foo")
  ;; ...
  (properties
    '((release-tag-prefix . "^release0-")
      (release-tag-suffix . "[a-z]?$")
      (release-tag-version-delimiter . ":"))))
```

Zum Beispiel prüft folgender Befehl nur auf mögliche Aktualisierungen von auf `elpa.gnu.org` angebotenen Emacs-Paketen und von CRAN-Paketen:

```
$ 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`
: Eine Liste verfügbarer Aktualisierungsprogramme anzeigen und terminieren (siehe `--type` oben).

  Für jedes Aktualisierungsprogramm den Anteil der davon betroffenen Pakete anzeigen; zum Schluss wird der Gesamtanteil von irgendeinem Aktualisierungsprogramm betroffener Pakete angezeigt.

An `guix refresh` können auch ein oder mehrere Paketnamen übergeben werden wie in diesem Beispiel:

```
$ ./pre-inst-env guix refresh -u emacs idutils gcc@4.8
```

Der Befehl oben aktualisiert speziell das `emacs`- und das `idutils`-Paket. Eine Befehlszeilenoption `--select` hätte dann keine Wirkung. Vielleicht möchten Sie auch die Definitionen zu den in Ihr Profil installierten Paketen aktualisieren:

```
$ ./pre-inst-env guix refresh -u \
       $(guix package --list-installed | cut -f1)
```

Wenn Sie sich fragen, ob ein Paket aktualisiert werden sollte oder nicht, kann es helfen, sich anzuschauen, welche Pakete von der Aktualisierung betroffen wären und auf Kompatibilität hin geprüft werden sollten. Dazu kann die folgende Befehlszeilenoption zusammen mit einem oder mehreren Paketnamen an `guix refresh` übergeben werden:

`--list-dependent`
`-l`
: Auflisten, welche abhängigen Pakete auf oberster Ebene neu erstellt werden müssten, wenn eines oder mehrere Pakete aktualisiert würden.

  Siehe Abschnitt 10.10 [Aufruf von guix graph], Seite 228, für Informationen dazu, wie Sie die Liste der Abhängigen eines Pakets visualisieren können.

Bedenken Sie, dass die Befehlszeilenoption `--list-dependent` das Ausmaß der nach einer Aktualisierungen benötigten Neuerstellungen nur *annähert*. Es könnten auch unter Umständen mehr Neuerstellungen anfallen.

```
$ guix refresh --list-dependent flex
Die folgenden 120 Pakete zu erstellen, würde zur Folge haben, dass 213 abhängige Paket
hop@2.4.0 emacs-geiser@0.13 notmuch@0.18 mu@0.9.9.5 cflow@1.4 idutils@4.6 ...
```

Der oben stehende Befehl gibt einen Satz von Paketen aus, die Sie erstellen wollen könnten, um die Kompatibilität einer Aktualisierung des `flex`-Pakets beurteilen zu können.

`--list-transitive`
: Die Pakete auflisten, von denen eines oder mehrere Pakete abhängen.

```
$ 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 .
```

Der oben stehende Befehl gibt einen Satz von Paketen aus, die, wenn sie geändert würden, eine Neuerstellung des `flex`-Pakets auslösen würden.

Mit den folgenden Befehlszeilenoptionen können Sie das Verhalten von GnuPG anpassen:

`--gpg=`*`Befehl`*
: Den *Befehl* als GnuPG-2.x-Befehl einsetzen. Der *Befehl* wird im `$PATH` gesucht.

`--keyring=`*`Datei`*
: Die *Datei* als Schlüsselbund mit Anbieterschlüsseln verwenden. Die *Datei* muss im *Keybox-Format* vorliegen. Keybox-Dateien haben normalerweise einen Namen, der auf `.kbx` endet. Sie können mit Hilfe von GNU Privacy Guard (GPG) bearbeitet werden (siehe Abschnitt "kbxutil" in *Using the GNU Privacy Guard* für Informationen über ein Werkzeug zum Bearbeiten von Keybox-Dateien).

  Wenn diese Befehlszeilenoption nicht angegeben wird, benutzt `guix refresh` die Keybox-Datei `~/.config/guix/upstream/trustedkeys.kbx` als Schlüsselbund für Signierschlüssel von Anbietern. OpenPGP-Signaturen werden mit Schlüsseln aus diesem Schlüsselbund überprüft; fehlende Schlüssel

werden auch in diesen Schlüsselbund heruntergeladen (siehe `--key-download` unten).

Sie können Schlüssel aus Ihrem normalerweise benutzten GPG-Schlüsselbund in eine Keybox-Datei exportieren, indem Sie Befehle wie diesen benutzen:

```
gpg --export rms@gnu.org | kbxutil --import-openpgp >> mykeyring.kbx
```

Ebenso können Sie wie folgt Schlüssel in eine bestimmte Keybox-Datei herunterladen:

```
gpg --no-default-keyring --keyring mykeyring.kbx \
  --recv-keys 3CE464558A84FDC69DB40CFB090B11993D9AEBB5
```

Siehe Abschnitt "GPG Configuration Options" in *Using the GNU Privacy Guard* für mehr Informationen zur Befehlszeilenoption `--keyring` von GPG.

`--key-download=`*Richtlinie*
: Fehlende OpenPGP-Schlüssel gemäß dieser *Richtlinie* behandeln, für die eine der Folgenden angegeben werden kann:

  `always` Immer fehlende OpenPGP-Schlüssel herunterladen und zum GnuPG-Schlüsselbund des Nutzers hinzufügen.

  `never` Niemals fehlende OpenPGP-Schlüssel herunterladen, sondern einfach abbrechen.

  `interactive`
  : Ist ein Paket mit einem unbekannten OpenPGP-Schlüssel signiert, wird der Nutzer gefragt, ob der Schlüssel heruntergeladen werden soll oder nicht. Dies entspricht dem vorgegebenen Verhalten.

`--key-server=`*Host*
: Den mit *Host* bezeichneten Rechner als Schlüsselserver für OpenPGP benutzen, wenn ein öffentlicher Schlüssel importiert wird.

`--load-path=`*Verzeichnis*
`-L` *Verzeichnis*
: Das *Verzeichnis* vorne an den Suchpfad für Paketmodule anfügen (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

  Damit können Nutzer dafür sorgen, dass ihre eigenen selbstdefinierten Pakete für die Befehlszeilenwerkzeuge sichtbar sind.

Das `github`-Aktualisierungsprogramm benutzt die GitHub-Programmierschnittstelle (`https://developer.github.com/v3/`) (die „Github-API“), um Informationen über neue Veröffentlichungen einzuholen. Geschieht dies oft, z.B. beim Auffrischen aller Pakete, so wird GitHub irgendwann aufhören, weitere API-Anfragen zu beantworten. Normalerweise sind 60 API-Anfragen pro Stunde erlaubt, für eine vollständige Auffrischung aller GitHub-Pakete in Guix werden aber mehr benötigt. Wenn Sie sich bei GitHub mit Ihrem eigenen API-Token authentisieren, gelten weniger einschränkende Grenzwerte. Um einen API-Token zu benutzen, setzen Sie die Umgebungsvariable `GUIX_GITHUB_TOKEN` auf einen von `https://github.com/settings/tokens` oder anderweitig bezogenen API-Token.

10.7 `guix style` aufrufen

Mit dem Befehl `guix style` können sowohl Nutzer als auch Paketentwickler ihre Paketdefinitionen und Konfigurationsdateien entsprechend der aktuell modischen Trends umgestalten. Sie können entweder ganze Dateien mit der Befehlszeilenoption `--whole-file` umformatieren oder die gewählten *Stilregeln* auf einzelne Paketdefinitionen anwenden. Im Moment werden Regeln für die folgenden Stilfragen angeboten:

- Den Stil der Paketeingaben nach den Projektkonventionen auszurichten (siehe Abschnitt 22.5.4 [Formatierung von Code], Seite 724), sowie
- Paketeingaben in den „neuen Stil“ umzuschreiben, wie unten erklärt wird.

Zurzeit läuft eine Umstellung der Notation, wie Paketeingaben aufgeschrieben werden (siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113, für weitere Informationen zu Paketeingaben). Bis Version 1.3.0 wurden Paketeingaben im „alten Stil“ verfasst, d.h. man schrieb zu jeder Eingabe ausdrücklich eine Bezeichnung dazu, in der Regel der Paketname:

```
(package
  ;; ...
  ;; Der „alte Stil“ (veraltet).
  (inputs `(("libunistring" ,libunistring)
            ("libffi" ,libffi))))
```

Der alte Stil gilt heute als veraltet. Der bevorzugte Stil sieht so aus:

```
(package
  ;; ...
  ;; Der „neue Stil“.
  (inputs (list libunistring libffi)))
```

Ebenso gilt als veraltet, Eingaben mit `alist-delete` und seinen Freunden zu verarbeiten; bevorzugt wird `modify-inputs` (siehe Abschnitt 9.3 [Paketvarianten definieren], Seite 121, für weitere Informationen zu `modify-inputs`).

In den allermeisten Fällen ist das eine rein mechanische Änderung der oberflächlichen Syntax und die Pakete müssen dazu nicht einmal neu erstellt werden. Diese kann `guix style -S inputs` für Sie übernehmen, egal ob Sie an Paketen innerhalb des eigentlichen Guix oder in einem externen Kanal arbeiten.

Die allgemeine Syntax lautet:

```
guix style [Optionen] Paket...
```

Damit wird sich `guix style` der Analyse und Umschreibung der Definition von *Paket*... annehmen. Wenn Sie kein *Paket* angeben, nimmt es sich *alle* Pakete vor. Mit der Befehlszeilenoption `--styling` oder `-S` können Sie auswählen, nach welcher Stilregel `guix style` vorgeht. Vorgegeben ist die `format`-Regel, siehe unten.

Um ganze Quelldateien auf einmal umzuformatieren, lautet die Syntax:

```
guix style --whole-file Datei...
```

Die verfügbaren Befehlszeilenoptionen folgen.

`--dry-run`
`-n` Anzeigen, welche Stellen im Quellcode verändert würden, sie jedoch nicht verändern.

`--whole-file`
`-f` Ganze Quelldateien auf einmal umformatieren. In diesem Fall werden weitere Argumente als Dateinamen aufgefasst (und eben nicht als Paketnamen) und die Befehlszeilenoption `--styling` ist ohne Wirkung.

Zum Beispiel können Sie so die Datei mit Ihrer Betriebssystemkonfiguration umformatieren (vorausgesetzt Sie haben die Schreibberechtigung auf der Datei):

```
guix style -f /etc/config.scm
```

`--styling=`*Regel*
`-S` *Regel* Die *Regel* anwenden, die eine der folgenden Stilregeln sein muss:

`format` Die angegebene Paketdefinition bzw. mehrere Paketdefinitionen formatieren. Das ist die voreingestellte Stilregel. Wenn zum Beispiel eine Paketautorin, die Guix aus einem Checkout heraus ausführt (siehe Abschnitt 22.2 [Guix vor der Installation ausführen], Seite 710), die Definition des Coreutils-Pakets formatieren wollte, würde sie das machen:

```
./pre-inst-env guix style coreutils
```

`inputs` Paketeingaben in den oben beschriebenen „neuen Stil“ umschreiben. Hiermit würden Sie die Eingaben des Pakets `oderso` in Ihrem eigenen Kanal umschreiben:

```
guix style -L ~/eigener/kanal -S inputs oderso
```

Die Umschreibung geschieht auf konservative Art: Kommentare bleiben erhalten und wenn es den Code in einem `inputs`-Feld *nicht* erfassen kann, gibt `guix style` auf. Die Befehlszeilenoption `--input-simplification` ermöglicht, genau zu bestimmen, unter welchen Voraussetzungen Eingaben vereinfacht werden sollen.

`--list-stylings`
`-l` Alle verfügbaren Stilregeln auflisten und beschreiben und sonst nichts.

`--load-path=`*Verzeichnis*
`-L` *Verzeichnis*
Das *Verzeichnis* vorne an den Suchpfad für Paketmodule anfügen (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

`--expression=`*Ausdruck*
`-e` *Ausdruck*
Das Paket umgestalten, zu dem der *Ausdruck* ausgewertet wird.

Zum Beispiel startet dies:

```
guix style -e '(@ (gnu packages gcc) gcc-5)'
```

ändert den Stil der Paketdefinition für `gcc-5`.

`--input-simplification=`*Richtlinie*
Wenn Sie die Stilregel `inputs` verwenden, als ‘`-S inputs`’, geben Sie hiermit die Richtlinie zur Vereinfachung der Paketeingaben für die Fälle an, wenn eine Eingabenbezeichnung nicht zum damit assoziierten Paketnamen passt. Die *Richtlinie* kann eine der folgenden sein:

`silent` Die Eingaben nur vereinfachen, wenn die Änderung unmerklich sind, in dem Sinne, dass das Paket nicht aufs Neue erstellt werden muss (seine Ableitung also dieselbe bleibt).

`safe` Die Eingaben nur vereinfachen, wenn dies keine Probleme mit sich bringt. Hier darf das Paket neu erstellt werden müssen, aber die Änderung daran hat keine beobachtbaren Auswirkungen.

`always` Die Eingaben selbst dann vereinfachen, wenn die Eingabebezeichnungen *nicht* zu den Paketnamen passen, obwohl das Auswirkungen haben könnte.

Die Vorgabe ist `silent`, also nur unmerkliche Vereinfachungen, die keine Neuerstellung auslösen.

10.8 `guix lint` aufrufen

Den Befehl `guix lint` gibt es, um Paketentwicklern beim Vermeiden häufiger Fehler und bei der Einhaltung eines konsistenten Code-Stils zu helfen. Er führt eine Reihe von Prüfungen auf einer angegebenen Menge von Paketen durch, um in deren Definition häufige Fehler aufzuspüren. Zu den verfügbaren *Prüfern* gehören (siehe `--list-checkers` für eine vollständige Liste):

`synopsis`
`description`
Überprüfen, ob bestimmte typografische und stilistische Regeln in Paketbeschreibungen und -zusammenfassungen eingehalten wurden.

`inputs-should-be-native`
Eingaben identifizieren, die wahrscheinlich native Eingaben sein sollten.

`source`
`home-page`
`mirror-url`
`github-url`
`source-file-name`
Die URLs für die Felder `home-page` und `source` anrufen und nicht erreichbare URLs melden. Wenn passend, wird eine `mirror://`-URL vorgeschlagen. Wenn die Quell-URL auf eine GitHub-URL weiterleitet, wird eine Empfehlung ausgegeben, direkt letztere zu verwenden. Es wird geprüft, dass der Quell-Dateiname aussagekräftig ist, dass er also z.B. nicht nur aus einer Versionsnummer besteht oder als „git-checkout“ angegeben wurde, ohne dass ein `Dateiname` deklariert wurde (siehe Abschnitt 9.2.2 [„origin“-Referenz], Seite 118).

`source-unstable-tarball`
Analysiert die `source`-URL, um zu bestimmen, ob der Tarball von GitHub automatisch generiert wurde oder zu einer Veröffentlichung gehört. Leider werden GitHubs automatisch generierte Tarballs manchmal neu generiert.

`derivation`
Prüft, ob die Ableitung der angegebenen Pakete auf allen unterstützten Systmen erfolgreich berechnet werden kann (siehe Abschnitt 9.10 [Ableitungen], Seite 167).

`profile-collisions`
: Prüft, ob die Installation der angegebenen Pakete in ein Profil zu Kollisionen führen würde. Kollisionen treten auf, wenn mehrere Pakete mit demselben Namen aber anderer Versionsnummer oder anderem Store-Dateinamen propagiert werden. Siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113, für weitere Informationen zu propagierten Eingaben.

`archival`
: Überprüft, ob der Quellcode des Pakets bei der Software Heritage (`https://www.softwareheritage.org`) archiviert ist.

  Wenn der noch nicht archivierte Quellcode aus einem Versionskontrollsystem („Version Control System“, VCS) stammt, wenn er also z.B. mit `git-fetch` bezogen wird, wird eine Anfrage an Software Heritage gestellt, diesen zu speichern („Save“), damit sie ihn irgendwann in deren Archiv aufnehmen. So wird gewährleistet, dass der Quellcode langfristig verfügbar bleibt und Guix notfalls auf Software Heritage zurückgreifen kann, falls der Quellcode bei seinem ursprünglichen Anbieter verschwindet. Der Status kürzlicher Archivierungsanfragen kann online eingesehen werden (`https://archive.softwareheritage.org/save/#requests`).

  Wenn der Quellcode in Form eines über `url-fetch` zu beziehenden Tarballs vorliegt, wird bloß eine Nachricht ausgegeben, wenn er nicht archiviert ist. Zum Zeitpunkt, wo dies geschrieben wurde, ermöglicht Software Heritage keine Anfragen, beliebige Tarballs zu archivieren; wir arbeiten an Möglichkeiten wie auch *nicht* versionskontrollierter Quellcode archiviert werden kann.

  Software Heritage beschränkt (`https://archive.softwareheritage.org/api/#rate-limiting`), wie schnell dieselbe IP-Adresse Anfragen stellen kann. Ist das Limit erreicht, gibt `guix lint` eine Mitteilung aus und der `archival`-Prüfer steht so lange still, bis die Beschränkung wieder zurückgesetzt wurde.

`cve`
: Bekannte Sicherheitslücken melden, die in den Datenbanken der „Common Vulnerabilities and Exposures“ (CVE) aus diesem und dem letzten Jahr vorkommen, wie sie von der US-amerikanischen NIST veröffentlicht werden (`https://nvd.nist.gov/vuln/data-feeds`).

  Um Informationen über eine bestimmte Sicherheitslücke angezeigt zu bekommen, besuchen Sie Webseiten wie:

  - ‘`https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-YYYY-ABCD`’
  - ‘`https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-YYYY-ABCD`’

  wobei Sie statt `CVE-YYYY-ABCD` die CVE-Kennnummer angeben – z.B. `CVE-2015-7554`.

  Paketentwickler können in ihren Paketrezepten den Namen und die Version des Pakets in der Common Platform Enumeration (CPE) (`https://nvd.nist.gov/products/cpe`) angeben, falls sich diese von dem in Guix benutzten Namen und der Version unterscheiden, zum Beispiel so:

  ```
  (package
    (name "grub")
    ;; ...
    ;; CPE bezeichnet das Paket als "grub2".
  ```

```
(properties '((cpe-name . "grub2")
              (cpe-version . "2.3"))))
```

Manche Einträge in der CVE-Datenbank geben die Version des Pakets nicht an, auf das sie sich beziehen, und würden daher bis in alle Ewigkeit Warnungen auslösen. Paketentwickler, die CVE-Warnmeldungen gefunden und geprüft haben, dass diese ignoriert werden können, können sie wie in diesem Beispiel deklarieren:

```
(package
  (name "t1lib")
  ;; ...
  ;; Diese CVEs treffen nicht mehr zu und können bedenkenlos ignoriert
  ;; werden.
  (properties `((lint-hidden-cve . ("CVE-2011-0433"
                                    "CVE-2011-1553"
                                    "CVE-2011-1554"
                                    "CVE-2011-5244")))))
```

`formatting`
: Offensichtliche Fehler bei der Formatierung von Quellcode melden, z.B. Leerraum-Zeichen am Zeilenende oder Nutzung von Tabulatorzeichen.

`input-labels`
: Eingabebezeichnungen im alten Stil melden, die nicht dem Namen des damit assoziierten Pakets entsprechen. Diese Funktionalität soll bei der Migration weg vom „alten Eingabenstil“ helfen. Siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113, um mehr Informationen über Paketeingaben und Eingabenstile zu bekommen. Siehe Abschnitt 10.7 [Aufruf von guix style], Seite 221, für Informationen, wie Sie zum neuen Stil migrieren können.

Die allgemeine Syntax lautet:

```
guix lint Optionen Pakete...
```

Wird kein Paket auf der Befehlszeile angegeben, dann werden alle Pakete geprüft, die es gibt. Als *Optionen* können null oder mehr der folgenden Befehlszeilenoptionen übergeben werden:

`--list-checkers`
`-l`
: Alle verfügbaren Prüfer für die Pakete auflisten und beschreiben.

`--checkers`
`-c`
: Nur die Prüfer aktivieren, die hiernach in einer kommagetrennten Liste aus von `--list-checkers` aufgeführten Prüfern vorkommen.

`--exclude`
`-x`
: Nur die Prüfer deaktivieren, die hiernach in einer kommagetrennten Liste aus von `--list-checkers` aufgeführten Prüfern vorkommen.

`--expression=`*Ausdruck*
`-e` *Ausdruck*
: Als Paket benutzen, wozu der *Ausdruck* ausgewertet wird.

Dies ist nützlich, um die Pakete eindeutig angeben zu können, wie in diesem Beispiel:

```
guix lint -c archival -e '(@ (gnu packages guile) guile-3.0)'
```

`--no-network`
`-n` Nur die Prüfer aktivieren, die keinen Internetzugang benötigen.

`--load-path=`*Verzeichnis*
`-L` *Verzeichnis*
Das *Verzeichnis* vorne an den Suchpfad für Paketmodule anfügen (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

Damit können Nutzer dafür sorgen, dass ihre eigenen selbstdefinierten Pakete für die Befehlszeilenwerkzeuge sichtbar sind.

10.9 `guix size` aufrufen

Der Befehl `guix size` hilft Paketentwicklern dabei, den Plattenplatzverbrauch von Paketen zu profilieren. Es ist leicht, die Auswirkungen zu unterschätzen, die das Hinzufügen zusätzlicher Abhängigkeiten zu einem Paket hat oder die das Verwenden einer einzelnen Ausgabe für ein leicht aufteilbares Paket ausmacht (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60). Das sind typische Probleme, auf die `guix size` aufmerksam machen kann.

Dem Befehl können eine oder mehrere Paketspezifikationen wie `gcc@4.8` oder `guile:debug` übergeben werden, oder ein Dateiname im Store. Betrachten Sie dieses Beispiel:

```
$ guix size coreutils
Store-Objekt                              Gesamt   Selbst
/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%
Gesamt: 78.9 MiB
```

Die hier aufgelisteten Store-Objekte bilden den *transitiven Abschluss* der Coreutils – d.h. die Coreutils und all ihre Abhängigkeiten und deren Abhängigkeiten, rekursiv –, wie sie hiervon angezeigt würden:<f

```
$ guix gc -R /gnu/store/...-coreutils-8.23
```

Hier zeigt die Ausgabe neben den Store-Objekten noch drei Spalten. Die erste Spalte namens „Gesamt“ gibt wieder, wie viele Mebibytes (MiB) der Abschluss des Store-Objekts groß ist – das heißt, dessen eigene Größe plus die Größe all seiner Abhängigkeiten. Die nächste Spalte, bezeichnet mit „Selbst“, zeigt die Größe nur dieses Objekts an. Die letzte Spalte zeigt das Verhältnis der Größe des Objekts zur Gesamtgröße aller hier aufgelisteten Objekte an.

In diesem Beispiel sehen wir, dass der Abschluss der Coreutils 79 MiB schwer ist, wovon das meiste durch libc und die Bibliotheken zur Laufzeitunterstützung von GCC ausgemacht wird. (Dass libc und die Bibliotheken vom GCC einen großen Anteil am Abschluss ausmachen, ist aber an sich noch kein Problem, weil es Bibliotheken sind, die auf dem System sowieso immer verfügbar sein müssen.)

Weil der Befehl auch Namen von Store-Dateien akzeptiert, kann man damit auch die Größe eines Erstellungsergebnisses ermitteln:

```
guix size $(guix system build config.scm)
```

Wenn das oder die Paket(e), die an `guix size` übergeben wurden, im Store verfügbar sind[1], beauftragen Sie mit `guix size` den Daemon, die Abhängigkeiten davon zu bestimmen und deren Größe im Store zu messen, ähnlich wie es mit `du -ms --apparent-size` geschehen würde (siehe Abschnitt "du invocation" in *GNU Coreutils*).

Wenn die übergebenen Pakete *nicht* im Store liegen, erstattet `guix size` Bericht mit Informationen, die aus verfügbaren Substituten herausgelesen werden (siehe Abschnitt 6.3 [Substitute], Seite 56). Dadurch kann die Plattenausnutzung von Store-Objekten profiliert werden, die gar nicht auf der Platte liegen und nur auf entfernten Rechnern vorhanden sind.

Sie können auch mehrere Paketnamen angeben:

```
$ guix size coreutils grep sed bash
Store-Objekt                              Gesamt   Selbst
/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%
...
Gesamt: 102.3 MiB
```

In diesem Beispiel sehen wir, dass die Kombination der vier Pakete insgesamt 102,3 MiB Platz verbraucht, was wesentlich weniger als die Summe der einzelnen Abschlüsse ist, weil diese viele Abhängigkeiten gemeinsam verwenden.

Wenn Sie sich das von `guix size` gelieferte Profil anschauen, fragen Sie sich vielleicht, warum ein bestimmtes Paket überhaupt darin auftaucht. Den Grund erfahren Sie, wenn Sie `guix graph --path -t references` benutzen, um sich den kürzesten Pfad zwischen zwei Paketen anzeigen zu lassen (siehe Abschnitt 10.10 [Aufruf von guix graph], Seite 228).

Die verfügbaren Befehlszeilenoptionen sind:

`--substitute-urls=`*URLs*
: Substitutinformationen von den *URLs* benutzen. Siehe [client-substitute-urls], Seite 190.

`--sort=`*Schlüssel*
: Zeilen anhand des *Schlüssels* sortieren, der eine der folgenden Alternativen sein muss:

  `self` die Größe jedes Objekts (die Vorgabe),

[1] Genauer gesagt braucht `guix size` die *nicht veredelte* Variante des angegebenen Pakets bzw. der Pakete, wie `guix build` *Paket* `--no-grafts` sie liefert. Siehe Kapitel 19 [Sicherheitsaktualisierungen], Seite 697, für Informationen über Veredelungen.

`Abschluss`
: die Gesamtgröße des Abschlusses des Objekts.

`--map-file=`*Datei*
: Eine grafische Darstellung des Plattenplatzverbrauchs als eine PNG-formatierte Karte in die *Datei* schreiben.

  Für das Beispiel oben sieht die Karte so aus:

  Diese Befehlszeilenoption setzt voraus, dass Guile-Charting (`https://wingolog.org/software/guile-charting/`) installiert und im Suchpfad für Guile-Module sichtbar ist. Falls nicht, schlägt `guix size` beim Versuch fehl, dieses Modul zu laden.

`--system=`*System*
`-s` *System* Pakete für dieses *System* betrachten – z.B. für `x86_64-linux`.

`--load-path=`*Verzeichnis*
`-L` *Verzeichnis*
: Das *Verzeichnis* vorne an den Suchpfad für Paketmodule anfügen (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

  Damit können Nutzer dafür sorgen, dass ihre eigenen selbstdefinierten Pakete für die Befehlszeilenwerkzeuge sichtbar sind.

10.10 `guix graph` aufrufen

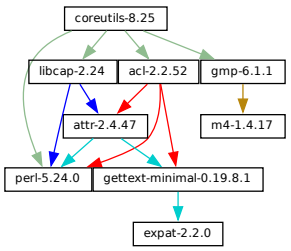
Pakete und ihre Abhängigkeiten bilden einen *Graphen*, genauer gesagt einen gerichteten azyklischen Graphen (englisch „Directed Acyclic Graph“, kurz DAG). Es kann schnell schwierig werden, ein Modell eines Paket-DAGs vor dem geistigen Auge zu behalten, weshalb der Befehl `guix graph` eine visuelle Darstellung des DAGs bietet. Das vorgegebene Verhalten von `guix graph` ist, eine DAG-Darstellung im Eingabeformat von Graphviz (`https://www.graphviz.org/`) auszugeben, damit die Ausgabe direkt an den Befehl `dot` aus Graphviz weitergeleitet werden kann. Es kann aber auch eine HTML-Seite mit eingebettetem JavaScript-Code ausgegeben werden, um ein Sehnendiagramm (englisch „Chord Diagram“) in einem Web-Browser anzuzeigen, mit Hilfe der Bibliothek d3.js (`https://d3js.org/`), oder es können Cypher-Anfragen ausgegeben werden, mit denen eine die Anfragesprache openCypher (`https://www.opencypher.org/`) unterstützende Graph-Datenbank einen Graphen konstruieren kann. Wenn Sie `--path` angeben, zeigt Guix Ihnen einfach nur den kürzesten Pfad zwischen zwei Paketen an. Die allgemeine Syntax ist:

```
guix graph Optionen Pakete...
```

Zum Beispiel erzeugt der folgende Befehl eine PDF-Datei, die den Paket-DAG für die GNU Core Utilities darstellt, welcher ihre Abhängigkeiten zur Erstellungszeit anzeigt:

```
guix graph coreutils | dot -Tpdf > dag.pdf
```

Die Ausgabe sieht so aus:



Ein netter, kleiner Graph, oder?

Vielleicht ist es Ihnen aber lieber, den Graphen interaktiv mit dem `xdot`-Programm anzuschauen (aus dem Paket `xdot`):

```
guix graph coreutils | xdot -
```

Aber es gibt mehr als eine Art von Graph! Der Graph oben ist kurz und knapp: Es ist der Graph der Paketobjekte, ohne implizite Eingaben wie GCC, libc, grep und so weiter. Oft möchte man einen knappen Graphen sehen, aber manchmal will man auch mehr Details sehen. `guix graph` unterstützt mehrere Typen von Graphen; Sie können den Detailgrad auswählen.

`package` Der vorgegebene Typ aus dem Beispiel oben. Er zeigt den DAG der Paketobjekte ohne implizite Abhängigkeiten. Er ist knapp, filtert aber viele Details heraus.

`reverse-package`
: Dies zeigt den *umgekehrten* DAG der Pakete. Zum Beispiel liefert

  ```
  guix graph --type=reverse-package ocaml
  ```

  . . . den Graphen der Pakete, die *explizit* von OCaml abhängen (wenn Sie auch an Fällen interessiert sind, bei denen OCaml eine implizite Abhängigkeit ist, siehe `reverse-bag` weiter unten).

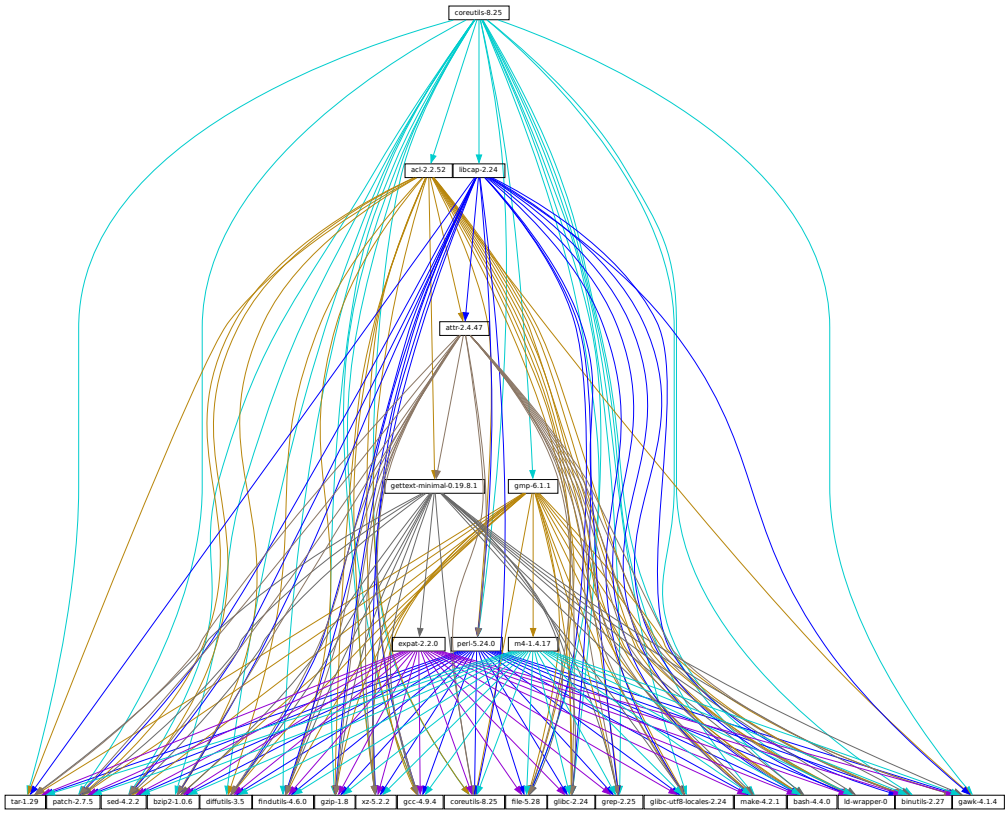
  Beachten Sie, dass für Kernpakete damit gigantische Graphen entstehen können. Wenn Sie nur die Anzahl der Pakete wissen wollen, die von einem gegebenen Paket abhängen, benutzen Sie `guix refresh --list-dependent` (siehe Abschnitt 10.6 [Aufruf von guix refresh], Seite 214).

`bag-emerged`
: Dies ist der Paket-DAG *einschließlich* impliziter Eingaben.

  Zum Beispiel liefert der folgende Befehl

  ```
  guix graph --type=bag-emerged coreutils
  ```

  . . . diesen größeren Graphen:



Am unteren Rand des Graphen sehen wir alle impliziten Eingaben des *gnu-build-system* (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130).

Beachten Sie dabei aber, dass auch hier die Abhängigkeiten dieser impliziten Eingaben – d.h. die *Bootstrap-Abhängigkeiten* (siehe Kapitel 20 [Bootstrapping], Seite 699) – nicht gezeigt werden, damit der Graph knapper bleibt.

`bag` Ähnlich wie `bag-emerged`, aber diesmal mit allen Bootstrap-Abhängigkeiten.

`bag-with-origins`
Ähnlich wie `bag`, aber auch mit den Ursprüngen und deren Abhängigkeiten.

`reverse-bag`
Dies zeigt den *umgekehrten* DAG der Pakete. Anders als `reverse-package` werden auch implizite Abhängigkeiten berücksichtigt. Zum Beispiel liefert

```
guix graph -t reverse-bag dune
```

. . . den Graphen aller Pakete, die von Dune direkt oder indirekt abhängen. Weil Dune eine *implizite* Abhängigkeit von vielen Paketen über das `dune-build-system` ist, zeigt er eine große Zahl von Paketen, während bei `reverse-package` nur sehr wenige bis gar keine zu sehen sind.

`derivation`
: Diese Darstellung ist am detailliertesten: Sie zeigt den DAG der Ableitungen (siehe Abschnitt 9.10 [Ableitungen], Seite 167) und der einfachen Store-Objekte. Verglichen mit obiger Darstellung sieht man viele zusätzliche Knoten einschließlich Erstellungs-Skripts, Patches, Guile-Module usw.

  Für diesen Typ Graph kann auch der Name einer `.drv`-Datei anstelle eines Paketnamens angegeben werden, etwa so:

```
guix graph -t derivation $(guix system build -d my-config.scm)
```

`Modul`
: Dies ist der Graph der *Paketmodule* (siehe Abschnitt 9.1 [Paketmodule], Seite 108). Zum Beispiel zeigt der folgende Befehl den Graphen für das Paketmodul an, das das `guile`-Paket definiert:

```
guix graph -t module guile | xdot -
```

Alle oben genannten Typen entsprechen *Abhängigkeiten zur Erstellungszeit*. Der folgende Graphtyp repräsentiert die *Abhängigkeiten zur Laufzeit*:

`references`
: Dies ist der Graph der *Referenzen* einer Paketausgabe, wie `guix gc --references` sie liefert (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61).

  Wenn die angegebene Paketausgabe im Store nicht verfügbar ist, versucht `guix graph`, die Abhängigkeitsinformationen aus Substituten zu holen.

  Hierbei können Sie auch einen Store-Dateinamen statt eines Paketnamens angeben. Zum Beispiel generiert der Befehl unten den Referenzgraphen Ihres Profils (der sehr groß werden kann!):

```
guix graph -t references $(readlink -f ~/.guix-profile)
```

`referrers`
: Dies ist der Graph der ein Store-Objekt *referenzierenden* Objekte, wie `guix gc --referrers` sie liefern würde (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61).

  Er basiert ausschließlich auf lokalen Informationen aus Ihrem Store. Nehmen wir zum Beispiel an, dass das aktuelle Inkscape in 10 Profilen verfügbar ist, dann wird `guix graph -t referrers inkscape` einen Graph zeigen, der bei Inkscape gewurzelt ist und Kanten zu diesen 10 Profilen hat.

  Ein solcher Graph kann dabei helfen, herauszufinden, weshalb ein Store-Objekt nicht vom Müllsammler abgeholt werden kann.

Oftmals passt der Graph des Pakets, für das Sie sich interessieren, nicht auf Ihren Bildschirm, und überhaupt möchten Sie ja nur wissen, *warum* das Paket von einem anderen abhängt, das scheinbar nichts damit zu tun hat. Die Befehlszeilenoption `--path` weist `guix graph` an, den kürzesten Pfad zwischen zwei Paketen (oder Ableitungen, Store-Objekten etc.) anzuzeigen:

```
$ 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
```

Manchmal eignet es sich, den angezeigten Graphen abzuschneiden, damit er ordentlich dargestellt werden kann. Eine Möglihkeit ist, mit der Befehlszeilenoption `--max-depth` (kurz `-M`) die maximale Tiefe des Graphen festzulegen. Im folgenden Beispiel wird nur `libreoffice` und diejenigen Knoten dargestellt, deren Distanz zu `libreoffice` höchstens 2 beträgt:

```
guix graph -M 2 libreoffice | xdot -f fdp -
```

Dabei bleibt zwar immer noch ein gewaltiges Spaghettiknäuel übrig, aber zumindest kann das `dot`-Programm daraus schnell eine halbwegs lesbare Darstellung erzeugen.

Folgendes sind die verfügbaren Befehlszeilenoptionen:

`--type=`*Typ*
`-t` *Typ*
: Eine Graph-Ausgabe dieses *Typ*s generieren. Dieser *Typ* muss einer der oben genannten Werte sein.

`--list-types`
: Die unterstützten Graph-Typen auflisten.

`--backend=`*Backend*
`-b` *Backend*
: Einen Graph mit Hilfe des ausgewählten *Backend*s generieren.

`--list-backends`
: Die unterstützten Graph-Backends auflisten.

  Derzeit sind die verfügbaren Backends Graphviz und d3.js.

`--path`
: Den kürzesten Pfad zwischen zwei Knoten anzeigen, die den mit `--type` angegebenen Typ aufweisen. Im Beispiel unten wird der kürzeste Pfad zwischen `libreoffice` und `llvm` anhand der Referenzen von `libreoffice` angezeigt:

  ```
  $ 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=`*Ausdruck*
`-e` *Ausdruck*
: Als Paket benutzen, wozu der *Ausdruck* ausgewertet wird.

  Dies ist nützlich, um genau ein bestimmtes Paket zu referenzieren, wie in diesem Beispiel:

  ```
  guix graph -e '(@@ (gnu packages commencement) gnu-make-final)'
  ```

`--system=`*System*
`-s` *System* Den Graphen für das *System* anzeigen – z.B. `i686-linux`.

Der Abhängigkeitsgraph ist größtenteils von der Systemarchitektur unabhängig, aber ein paar architekturabhängige Teile können Ihnen mit dieser Befehlszeilenoption visualisiert werden.

`--load-path=`*Verzeichnis*
`-L` *Verzeichnis*

Das *Verzeichnis* vorne an den Suchpfad für Paketmodule anfügen (siehe Abschnitt 9.1 [Paketmodule], Seite 108).

Damit können Nutzer dafür sorgen, dass ihre eigenen selbstdefinierten Pakete für die Befehlszeilenwerkzeuge sichtbar sind.

Hinzu kommt, dass `guix graph` auch all die üblichen Paketumwandlungsoptionen unterstützt (siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192). Somit ist es kein Problem, die Folgen einer den Paketgraphen umschreibenden Umwandlung wie `--with-input` zu erkennen. Zum Beispiel gibt der folgende Befehl den Graphen von `git` aus, nachdem `openssl` an allen Stellen im Graphen durch `libressl` ersetzt wurde:

```
guix graph git --with-input=openssl=libressl
```

Ihrem Vergnügen sind keine Grenzen gesetzt!

10.11 `guix publish` aufrufen

Der Zweck von `guix publish` ist, es Nutzern zu ermöglichen, ihren Store auf einfache Weise mit anderen zu teilen, die ihn dann als Substitutserver einsetzen können (siehe Abschnitt 6.3 [Substitute], Seite 56).

Wenn `guix publish` ausgeführt wird, wird dadurch ein HTTP-Server gestartet, so dass jeder mit Netzwerkzugang davon Substitute beziehen kann. Das bedeutet, dass jede Maschine, auf der Guix läuft, auch als Erstellungsfarm fungieren kann, weil die HTTP-Schnittstelle mit Cuirass, der Software, mit der die offizielle Erstellungsfarm `ci.guix.gnu.org` betrieben wird, kompatibel ist.

Um Sicherheit zu gewährleisten, wird jedes Substitut signiert, so dass Empfänger dessen Authentizität und Integrität nachprüfen können (siehe Abschnitt 6.3 [Substitute], Seite 56). Weil `guix publish` den Signierschlüssel des Systems benutzt, der nur vom Systemadministrator gelesen werden kann, muss es als der Administratornutzer „root“ gestartet werden. Mit der Befehlszeilenoption `--user` werden Administratorrechte bald nach dem Start wieder abgelegt.

Das Schlüsselpaar zum Signieren muss erzeugt werden, bevor `guix publish` gestartet wird. Dazu können Sie `guix archive --generate-key` ausführen (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74).

Wird die Befehlszeilenoption `--advertise` übergeben, teilt der Server anderen Rechnern im lokalen Netzwerk seine Verfügbarkeit mit, über Multicast-DNS (mDNS) und DNS-Service-Discovery (DNS-SD), zurzeit mittels Guile-Avahi (siehe *Using Avahi in Guile Scheme Programs*).

Die allgemeine Syntax lautet:

```
guix publish Optionen...
```

Wird `guix publish` ohne weitere Argumente ausgeführt, wird damit ein HTTP-Server gestartet, der auf Port 8080 lauscht:

```
guix publish
```

Sie können `guix publish` auch über das systemd-Protokoll zur „Socket-Aktivierung“ starten (siehe Abschnitt “Service De- and Constructors” in *The GNU Shepherd Manual*).

Sobald ein Server zum Veröffentlichen autorisiert wurde, kann der Daemon davon Substitute herunterladen. Siehe Abschnitt 6.3.3 [Substitute von anderen Servern holen], Seite 57.

Nach den Voreinstellungen komprimiert `guix publish` Archive erst dann, wenn sie angefragt werden. Dieser „dynamische“ Modus bietet sich an, weil so nichts weiter eingerichtet werden muss und er direkt verfügbar ist. Wenn Sie allerdings viele Clients bedienen wollen, empfehlen wir, dass Sie die Befehlszeilenoption `--cache` benutzen, die das Zwischenspeichern der komprimierten Archive aktiviert, bevor diese an die Clients geschickt werden – siehe unten für Details. Mit dem Befehl `guix weather` haben Sie eine praktische Methode zur Hand, zu überprüfen, was so ein Server anbietet (siehe Abschnitt 10.15 [Aufruf von guix weather], Seite 242).

Als Bonus dient `guix publish` auch als inhaltsadressierbarer Spiegelserver für Quelldateien, die in `origin`-Verbundsobjekten eingetragen sind (siehe Abschnitt 9.2.2 [„origin“-Referenz], Seite 118). Wenn wir zum Beispiel annehmen, dass `guix publish` auf `example.org` läuft, liefert folgende URL die rohe `hello-2.10.tar.gz`-Datei mit dem angegebenen SHA256-Hash als ihre Prüfsumme (dargestellt im `nix-base32`-Format, siehe Abschnitt 10.4 [Aufruf von guix hash], Seite 205):

```
http://example.org/file/hello-2.10.tar.gz/sha256/0ssi1...ndq1i
```

Offensichtlich funktionieren diese URLs nur mit solchen Dateien, die auch im Store vorliegen; in anderen Fällen werden sie 404 („Nicht gefunden“) zurückliefern.

Erstellungsprotokolle sind unter `/log`-URLs abrufbar:

```
http://example.org/log/gwspk...-guile-2.2.3
```

Ist der `guix-daemon` so eingestellt, dass er Erstellungsprotokolle komprimiert abspeichert, wie es voreingestellt ist (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18), liefern `/log`-URLs das unveränderte komprimierte Protokoll, mit einer entsprechenden `Content-Type`- und/oder `Content-Encoding`-Kopfzeile. Wir empfehlen dabei, dass Sie den `guix-daemon` mit `--log-compression=gzip` ausführen, weil Web-Browser dieses Format automatisch dekomprimieren können, was bei Bzip2-Kompression nicht der Fall ist.

Folgende Befehlszeilenoptionen stehen zur Verfügung:

`--port=`*Port*
`-p` *Port* Auf HTTP-Anfragen auf diesem *Port* lauschen.

`--listen=`*Host*
: Auf der Netzwerkschnittstelle für den angegebenen *Host*, also der angegebenen Rechneradresse, lauschen. Vorgegeben ist, Verbindungen mit jeder Schnittstelle zu akzeptieren.

`--user=`*Benutzer*
`-u` *Benutzer*
: So früh wie möglich alle über die Berechtigungen des *Benutzer*s hinausgehenden Berechtigungen ablegen – d.h. sobald der Server-Socket geöffnet und der Signierschlüssel gelesen wurde.

`--compression[=`*Methode*`[:`*Stufe*`]]`
`-C [`*Methode*`[:`*Stufe*`]]`

Daten auf der angegebenen Kompressions-*Stufe* mit der angegebenen *Methode* komprimieren. Als *Methode* kann entweder `lzip`, `zstd` oder `gzip` angegeben werden. Wird keine *Methode* angegeben, wird `gzip` benutzt.

Daten auf der angegebenen Kompressions-*Stufe* komprimieren. Wird als *Stufe* null angegeben, wird Kompression deaktiviert. Der Bereich von 1 bis 9 entspricht unterschiedlichen Kompressionsstufen: 1 ist am schnellsten, während 9 am besten komprimiert (aber den Prozessor mehr auslastet). Der Vorgabewert ist 3.

Normalerweise ist die Kompression mit `lzip` wesentlich besser als bei `gzip`, dafür wird der Prozessor geringfügig stärker ausgelastet; siehe Vergleichswerte auf dem Webauftritt von lzip (`https://nongnu.org/lzip/lzip_benchmark.html`). `lzip` erreicht jedoch nur einen niedrigen Durchsatz bei der Dekompression (in der Größenordnung von 50 MiB/s auf moderner Hardware), was einen Engpass beim Herunterladen über schnelle Netzwerkverbindungen darstellen kann.

Das Kompressionsverhältnis von `zstd` liegt zwischen dem von `lzip` und dem von `gzip`; sein größter Vorteil ist eine hohe Geschwindigkeit bei der Dekompression (`https://facebook.github.io/zstd/`).

Wenn `--cache` nicht übergeben wird, werden Daten dynamisch immer erst dann komprimiert, wenn sie abgeschickt werden; komprimierte Datenströme landen in keinem Zwischenspeicher. Um also die Auslastung der Maschine, auf der `guix publish` läuft, zu reduzieren, kann es eine gute Idee sein, eine niedrige Kompressionsstufe zu wählen, `guix publish` einen Proxy mit Zwischenspeicher (einen „Caching Proxy“) voranzuschalten, oder `--cache` zu benutzen. `--cache` zu benutzen, hat den Vorteil, dass `guix publish` damit eine `Content-Length`-HTTP-Kopfzeile seinen Antworten beifügen kann.

Wenn diese Befehlszeilenoption mehrfach angegeben wird, wird jedes Substitut mit allen ausgewählten Methoden komprimiert und jede davon wird bei Anfragen mitgeteilt. Das ist nützlich, weil Benutzer, bei denen nicht alle Kompressionsmethoden unterstützt werden, die passende wählen können.

`--cache=`*Verzeichnis*
`-c` *Verzeichnis*

Archive und Metadaten (`.narinfo`-URLs) in das *Verzeichnis* zwischenspeichern und nur solche Archive versenden, die im Zwischenspeicher vorliegen.

Wird diese Befehlszeilenoption weggelassen, dann werden Archive und Metadaten „dynamisch“ erst auf eine Anfrage hin erzeugt. Dadurch kann die verfügbare Bandbreite reduziert werden, besonders wenn Kompression aktiviert ist, weil die Operation dann durch die Prozessorleistung beschränkt sein kann. Noch ein Nachteil des voreingestellten Modus ist, dass die Länge der Archive nicht im Voraus bekannt ist, `guix publish` also keine `Content-Length`-HTTP-Kopfzeile an seine Antworten anfügt, wodurch Clients nicht wissen können, welche Datenmenge noch heruntergeladen werden muss.

Im Gegensatz dazu löst, wenn `--cache` benutzt wird, die erste Anfrage nach einem Store-Objekt (über dessen `.narinfo`-URL) den Start eines Hintergrundprozesses aus, der das Archiv in den Zwischenspeicher einlagert (auf Englisch sagen wir „bake the archive“), d.h. seine `.narinfo` wird berechnet und das Archiv, falls nötig, komprimiert. Sobald das Archiv im *Verzeichnis* zwischengespeichert wurde, werden nachfolgende Anfragen erfolgreich sein und direkt aus dem Zwischenspeicher bedient, der garantiert, dass Clients optimale Bandbreite genießen.

Die erste Anfrage nach einer `.narinfo` wird trotzdem 200 zurückliefern, wenn das angefragte Store-Objekt „klein genug“ ist, also kleiner als der Schwellwert für die Zwischenspeicherumgehung, siehe `--cache-bypass-threshold` unten. So müssen Clients nicht darauf warten, dass das Archiv eingelagert wurde. Bei größeren Store-Objekten liefert die erste `.narinfo`-Anfrage 404 zurück, was bedeutet, dass Clients warten müssen, bis das Archiv eingelagert wurde.

Der Prozess zum Einlagern wird durch Worker-Threads umgesetzt. Der Vorgabe entsprechend wird dazu pro Prozessorkern ein Thread erzeugt, aber dieses Verhalten kann angepasst werden. Siehe `--workers` weiter unten.

Wird `--ttl` verwendet, werden zwischengespeicherte Einträge automatisch gelöscht, sobald die dabei angegebene Zeit abgelaufen ist.

`--workers=`*N*
: Wird `--cache` benutzt, wird die Reservierung von *N* Worker-Threads angefragt, um Archive einzulagern.

`--ttl=`*ttl*
: `Cache-Control`-HTTP-Kopfzeilen erzeugen, die eine Time-to-live (TTL) von *ttl* signalisieren. Für *ttl* muss eine Dauer (mit dem Anfangsbuchstaben der Maßeinheit der Dauer im Englischen) angegeben werden: `5d` bedeutet 5 Tage, `1m` bedeutet 1 Monat und so weiter.

  Das ermöglicht es Guix, Substitutinformationen *ttl* lang zwischenzuspeichern. Beachten Sie allerdings, dass `guix publish` selbst *nicht* garantiert, dass die davon angebotenen Store-Objekte so lange verfügbar bleiben, wie es die *ttl* vorsieht.

  Des Weiteren können bei Nutzung von `--cache` die zwischengespeicherten Einträge gelöscht werden, wenn auf sie *ttl* lang nicht zugegriffen wurde und kein ihnen entsprechendes Objekt mehr im Store existiert.

`--negative-ttl=`*ttl*
: Eben solche `Cache-Control`-HTTP-Kopfzeilen für erfolglose (negative) Suchen erzeugen, um eine Time-to-live (TTL) zu signalisieren, wenn Store-Objekte fehlen und mit dem HTTP-Status-Code 404 geantwortet wird. Nach Vorgabe wird für negative Antworten *keine* TTL signalisiert.

  Dieser Parameter kann helfen, die Last auf dem Server und die Verzögerung zu regulieren, weil folgsame Clients angewiesen werden, für diese Zeit geduldig zu warten, wenn ein Store-Objekt fehlt.

`--cache-bypass-threshold=`*Größe*

Wird dies in Verbindung mit `--cache` angegeben, werden Store-Objekte kleiner als die *Größe* sofort verfügbar sein, obwohl sie noch nicht in den Zwischenspeicher eingelagert wurden. Die *Größe* gibt die Größe in Bytes an oder ist mit einem Suffix `M` für Megabyte und so weiter versehen. Die Vorgabe ist `10M`.

Durch diese „Zwischenspeicherumgehung“ lässt sich die Verzögerung bis zur Veröffentlichung an Clients reduzieren, auf Kosten zusätzlicher Ein-/Ausgaben und CPU-Auslastung für den Server. Je nachdem, welche Zugriffsmuster Clients haben, werden diese Store-Objekte vielleicht mehrfach zur Einlagerung vorbereitet, ehe eine Kopie davon im Zwischenspeicher verfügbar wird.

Wenn ein Anbieter nur wenige Nutzer unterhält, kann es helfen, den Schwellwert zu erhöhen, oder auch wenn garantiert werden soll, dass es auch für wenig beliebte Store-Objekte Substitute gibt.

`--nar-path=`*Pfad*

Den *Pfad* als Präfix für die URLs von „nar“-Dateien benutzen (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74).

Vorgegeben ist, dass Nars unter einer URL mit `/nar/gzip/...-coreutils-8.25` angeboten werden. Mit dieser Befehlszeilenoption können Sie den `/nar`-Teil durch den angegebenen *Pfad* ersetzen.

`--public-key=`*Datei*
`--private-key=`*Datei*

Die angegebenen *Datei*en als das Paar aus öffentlichem und privatem Schlüssel zum Signieren veröffentlichter Store-Objekte benutzen.

Die Dateien müssen demselben Schlüsselpaar entsprechen (der private Schlüssel wird zum Signieren benutzt, der öffentliche Schlüssel wird lediglich in den Metadaten der Signatur aufgeführt). Die Dateien müssen Schlüssel im kanonischen („canonical“) S-Ausdruck-Format enthalten, wie es von `guix archive --generate-key` erzeugt wird (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74). Vorgegeben ist, dass `/etc/guix/signing-key.pub` und `/etc/guix/signing-key.sec` benutzt werden.

`--repl[=`*Port*`]`
`-r [`*Port*`]` Einen Guile-REPL-Server (siehe Abschnitt “REPL Servers” in *Referenzhandbuch zu GNU Guile*) auf diesem *Port* starten (37146 ist voreingestellt). Dies kann zur Fehlersuche auf einem laufenden „`guix publish`“-Server benutzt werden.

`guix publish` auf einem „Guix System“-System zu aktivieren ist ein Einzeiler: Instanziieren Sie einfach einen `guix-publish-service-type`-Dienst im `services`-Feld Ihres `operating-system`-Objekts zur Betriebssystemdeklaration (siehe [guix-publish-service-type], Seite 295).

Falls Sie Guix aber auf einer „Fremddistribution“ laufen lassen, folgen Sie folgenden Anweisungen:

- Wenn Ihre Wirtsdistribution systemd als „init“-System benutzt:

```
# ln -s ~root/.guix-profile/lib/systemd/system/guix-publish.service \
       /etc/systemd/system/
```

```
# systemctl start guix-publish && systemctl enable guix-publish
```

- Wenn Ihre Wirts-Distribution als „init“-System Upstart verwendet:

```
# ln -s ~root/.guix-profile/lib/upstart/system/guix-publish.conf /etc/init/
# start guix-publish
```

- Verfahren Sie andernfalls auf die gleiche Art für das „init“-System, das Ihre Distribution verwendet.

10.12 `guix challenge` aufrufen

Entsprechen die von diesem Server gelieferten Binärdateien tatsächlich dem Quellcode, aus dem sie angeblich erzeugt wurden? Ist ein Paketerstellungsprozess deterministisch? Diese Fragen versucht `guix challenge` zu beantworten.

Die erste Frage ist offensichtlich wichtig: Bevor man einen Substitutserver benutzt (siehe Abschnitt 6.3 [Substitute], Seite 56), *verifiziert* man besser, dass er die richtigen Binärdateien liefert, d.h. man *fechtet sie an*. Die letzte Frage macht die erste möglich: Wenn Paketerstellungen deterministisch sind, müssten voneinander unabhängige Erstellungen genau dasselbe Ergebnis liefern, Bit für Bit; wenn ein Server mit einer anderen Binärdatei als der lokal erstellten Binärdatei antwortet, ist diese entweder beschädigt oder bösartig.

Wir wissen, dass die in `/gnu/store`-Dateinamen auftauchende Hash-Prüfsumme der Hash aller Eingaben des Prozesses ist, mit dem die Datei oder das Verzeichnis erstellt wurde – Compiler, Bibliotheken, Erstellungsskripts und so weiter (siehe Kapitel 1 [Einführung], Seite 1). Wenn wir von deterministischen Erstellungen ausgehen, sollte ein Store-Dateiname also auf genau eine Erstellungsausgabe abgebildet werden. Mit `guix challenge` prüft man, ob es tatsächlich eine eindeutige Abbildung gibt, indem die Erstellungsausgaben mehrerer unabhängiger Erstellungen jedes angegebenen Store-Objekts verglichen werden.

Die Ausgabe des Befehls sieht so aus:

```
$ guix challenge \
  --substitute-urls="https://ci.guix.gnu.org https://guix.example.org" \
  openssl git pius coreutils grep
Liste der Substitute von „https://ci.guix.gnu.org“ wird aktualisiert ... 100.0%
Liste der Substitute von „https://guix.example.org“ wird aktualisiert ... 100.0%
Inhalt von /gnu/store/...-openssl-1.0.2d verschieden:
  lokale Prüfsumme: 0725l22r5jnzazaacncwsvp9kgf42266ayyp814v7djxs7nk963q
  https://ci.guix.gnu.org/nar/...-openssl-1.0.2d: 0725l22r5jnzazaacncwsvp9kgf42266ayyp814v7djxs7nk963q
  https://guix.example.org/nar/...-openssl-1.0.2d: 1zy4fmaaqcnjrzzajkdn3f5gmjk754b43qkq47llbyak9z0qjyim
  Diese Dateien unterscheiden sich:
    /lib/libcrypto.so.1.1
    /lib/libssl.so.1.1

Inhalt von /gnu/store/...-git-2.5.0 verschieden:
  lokale Prüfsumme: 00p3bmryhjxrhpn2gxs2fy0a15lnip05l97205pgbk5ra395hyha
  https://ci.guix.gnu.org/nar/...-git-2.5.0: 069nb85bv4d4a6slrwjdy8v1cn4cwspm3kdbmyb81d6zckj3nq9f
  https://guix.example.org/nar/...-git-2.5.0: 0mdqa9w1p6cmli6976v4wi0sw9r4p5prkj7lzfd1877wk11c9c73
  Diese Datei unterscheidet sich:
    /libexec/git-core/git-fsck

Inhalt von /gnu/store/...-pius-2.1.1 verschieden:
  lokale Prüfsumme: 0k4v3m9z1zp8xzzizb7d8kjj72f9172xv078sq4wl73vnq9ig3ax
  https://ci.guix.gnu.org/nar/...-pius-2.1.1: 0k4v3m9z1zp8xzzizb7d8kjj72f9172xv078sq4wl73vnq9ig3ax
  https://guix.example.org/nar/...-pius-2.1.1: 1cy25x1a4fzq5rk0pmvc8xhwyffnqz95h2bpvqsz2mpvlbccy0gs
  Diese Datei unterscheidet sich:
```

```
    /share/man/man1/pius.1.gz

...

5 Store-Objekte wurden analysiert:
  --- 2 (40.0%) waren identisch
  --- 3 (60.0%) unterscheiden sich
  --- 0 (0.0%) blieben ergebnislos
```

In diesem Beispiel werden mit `guix challenge` alle Substitutserver zu jedem der fünf auf der Befehlszeile angegebenen Pakete angefragt. Diejenigen Store-Objekte, bei denen die Server ein anderes Ergebnis berechnet haben als die lokale Erstellung (falls vorhanden) oder wo die Server untereinander zu verschiedenen Ergebnissen gekommen sind, werden gemeldet. Daran, dass eine '`lokale Prüfsumme`' dabeisteht, erkennen Sie, dass für die Pakete eine lokale Erstellung vorgelegen hat.

Nehmen wir zum Beispiel an, `guix.example.org` gibt uns immer eine verschiedene Antwort, aber `ci.guix.gnu.org` stimmt mit lokalen Erstellungen überein, *außer* im Fall von Git. Das könnte ein Hinweis sein, dass der Erstellungsprozess von Git nichtdeterministisch ist; das bedeutet, seine Ausgabe variiert abhängig von verschiedenen Umständen, die Guix nicht vollends kontrollieren kann, obwohl es Pakete in isolierten Umgebungen erstellt (siehe Abschnitt 6.1 [Funktionalitäten], Seite 44). Zu den häufigsten Quellen von Nichtdeterminismus gehören das Einsetzen von Zeitstempeln innerhalb der Erstellungsgebnisse, das Einsetzen von Zufallszahlen und von Auflistungen eines Verzeichnisinhalts sortiert nach der Inode-Nummer. Siehe `https://reproducible-builds.org/docs/` für mehr Informationen.

Um herauszufinden, was mit dieser Git-Binärdatei nicht stimmt, ist es am leichtesten, einfach diesen Befehl auszuführen:

```
guix challenge git \
  --diff=diffoscope \
  --substitute-urls="https://ci.guix.gnu.org https://guix.example.org"
```

Dadurch wird `diffoscope` automatisch aufgerufen, um detaillierte Informationen über sich unterscheidende Dateien anzuzeigen.

Alternativ können wir so etwas machen (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74):

```
$ wget -q -O - https://ci.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
```

Dieser Befehl zeigt die Unterschiede zwischen den Dateien, die sich aus der lokalen Erstellung ergeben, und den Dateien, die sich aus der Erstellung auf `ci.guix.gnu.org` ergeben (siehe Abschnitt "Overview" in *Comparing and Merging Files*). Der Befehl `diff` funktioniert großartig für Textdateien. Wenn sich Binärdateien unterscheiden, ist Diffoscope (`https://diffoscope.org/`) die bessere Wahl: Es ist ein hilfreiches Werkzeug, das Unterschiede in allen Arten von Dateien visualisiert.

Sobald Sie mit dieser Arbeit fertig sind, können Sie erkennen, ob die Unterschiede aufgrund eines nichtdeterministischen Erstellungsprozesses oder wegen einem bösartigen Server zustande kommen. Wir geben uns Mühe, Quellen von Nichtdeterminismus in Paketen zu entfernen, damit Substitute leichter verifiziert werden können, aber natürlich ist an diesem

Prozess nicht nur Guix, sondern ein großer Teil der Freie-Software-Gemeinschaft beteiligt. In der Zwischenzeit ist `guix challenge` eines der Werkzeuge, die das Problem anzugehen helfen.

Wenn Sie ein Paket für Guix schreiben, ermutigen wir Sie, zu überprüfen, ob `ci.guix.gnu.org` und andere Substitutserver dasselbe Erstellungsergebnis bekommen, das Sie bekommen haben. Das geht so:

```
guix challenge Paket
```

Die allgemeine Syntax lautet:

```
guix challenge Optionen Argumente...
```

wobei jedes der *Argumente* eine Paketspezifikation wie `guile@2.0` oder `glibc:debug` sein muss oder alternativ der Name einer Datei im Store, wie er zum Beispiel durch `guix build` oder `guix gc --list-live` mitgeteilt wird.

Wird ein Unterschied zwischen der Hash-Prüfsumme des lokal erstellten Objekts und dem vom Server gelieferten Substitut festgestellt, oder zwischen den Substituten von unterschiedlichen Servern, dann wird der Befehl dies wie im obigen Beispiel anzeigen und mit dem Exit-Code 2 terminieren (andere Exit-Codes außer null stehen für andere Arten von Fehlern).

Die eine, wichtige Befehlszeilenoption ist:

`--substitute-urls=`*URLs*
: Die *URLs* als durch Leerraumzeichen getrennte Liste von Substitut-Quell-URLs benutzen. mit denen verglichen wird.

`--diff=`*Modus*
: Wenn sich Dateien unterscheiden, diese Unterschiede entsprechend dem *Modus* anzeigen. Dieser kann einer der folgenden sein:

  `simple` (die Vorgabe)
  : Zeige die Liste sich unterscheidender Dateien.

  `diffoscope`
  Befehl
  : Diffoscope (`https://diffoscope.org/`) aufrufen und ihm zwei Verzeichnisse mit verschiedenem Inhalt übergeben.

    Wenn der *Befehl* ein absoluter Dateiname ist, wird der *Befehl* anstelle von Diffoscope ausgeführt.

  `none`
  : Keine näheren Details zu Unterschieden anzeigen.

  Solange kein `--diff=none` angegeben wird, werden durch `guix challenge` also Store-Objekte von den festgelegten Substitutservern heruntergeladen, damit diese verglichen werden können.

`--verbose`
`-v`
: Details auch zu Übereinstimmungen (deren Inhalt identisch ist) ausgeben, zusätzlich zu Informationen über Unterschiede.

10.13 `guix copy` aufrufen

Der Befehl `guix copy` kopiert Objekte aus dem Store einer Maschine in den Store einer anderen Maschine mittels einer Secure-Shell-Verbindung (kurz SSH-Verbindung)[2]. Zum Beispiel kopiert der folgende Befehl das Paket `coreutils`, das Profil des Benutzers und all deren Abhängigkeiten auf den anderen *Rechner*, dazu meldet sich Guix als *Benutzer* an:

```
guix copy --to=Benutzer@Rechner \
          coreutils $(readlink -f ~/.guix-profile)
```

Wenn manche der zu kopierenden Objekte schon auf dem anderen *Rechner* vorliegen, werden sie tatsächlich *nicht* übertragen.

Der folgende Befehl bezieht `libreoffice` und `gimp` von dem *Rechner*, vorausgesetzt sie sind dort verfügbar:

```
guix copy --from=host libreoffice gimp
```

Die SSH-Verbindung wird mit dem Guile-SSH-Client hergestellt, der mit OpenSSH kompatibel ist: Er berücksichtigt `~/.ssh/known_hosts` und `~/.ssh/config` und verwendet den SSH-Agenten zur Authentifizierung.

Der Schlüssel, mit dem gesendete Objekte signiert sind, muss von der entfernten Maschine akzeptiert werden. Ebenso muss der Schlüssel, mit dem die Objekte signiert sind, die Sie von der entfernten Maschine empfangen, in Ihrer Datei `/etc/guix/acl` eingetragen sein, damit Ihr Daemon sie akzeptiert. Siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74, für mehr Informationen über die Authentifizierung von Store-Objekten.

Die allgemeine Syntax lautet:

```
guix copy [--to=Spezifikation|--from=Spezifikation] Objekte...
```

Sie müssen immer eine der folgenden Befehlszeilenoptionen angeben:

`--to=`*Spezifikation*
`--from=`*Spezifikation*
: Gibt den Rechner (den „Host“) an, an den oder von dem gesendet bzw. empfangen wird. Die *Spezifikation* muss eine SSH-Spezifikation sein wie `example.org`, `charlie@example.org` oder `charlie@example.org:2222`.

Die *Objekte* können entweder Paketnamen wie `gimp` oder Store-Objekte wie `/gnu/store/...-idutils-4.6` sein.

Wenn ein zu sendendes Paket mit Namen angegeben wird, wird es erst erstellt, falls es nicht im Store vorliegt, außer `--dry-run` wurde angegeben wurde. Alle gemeinsamen Erstellungsoptionen werden unterstützt (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189).

10.14 `guix container` aufrufen

Anmerkung: Dieses Werkzeug ist noch experimentell, Stand Version 1.4.0. Die Schnittstelle wird sich in Zukunft grundlegend verändern.

[2] Dieser Befehl steht nur dann zur Verfügung, wenn Guile-SSH gefunden werden kann. Siehe Abschnitt 2.2 [Voraussetzungen], Seite 8, für Details.

Der Zweck von `guix container` ist, in einer isolierten Umgebung (gemeinhin als „Container“ bezeichnet) laufende Prozesse zu manipulieren, die typischerweise durch die Befehle `guix shell` (siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86) und `guix system container` (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619) erzeugt werden.

Die allgemeine Syntax lautet:

```
guix container Aktion Optionen...
```

Mit *Aktion* wird die Operation angegeben, die in der isolierten Umgebung durchgeführt werden soll, und mit *Optionen* werden die kontextabhängigen Argumente an die Aktion angegeben.

Folgende Aktionen sind verfügbar:

`exec` Führt einen Befehl im Kontext der laufenden isolierten Umgebung aus.

Die Syntax ist:

```
guix container exec PID Programm Argumente...
```

PID gibt die Prozess-ID der laufenden isolierten Umgebung an. Als *Programm* muss eine ausführbare Datei im Wurzeldateisystem der isolierten Umgebung angegeben werden. Die *Argumente* sind die zusätzlichen Befehlszeilenoptionen, die an das *Programm* übergeben werden.

Der folgende Befehl startet eine interaktive Login-Shell innerhalb einer isolierten Guix-Systemumgebung, gestartet durch `guix system container`, dessen Prozess-ID 9001 ist:

```
guix container exec 9001 /run/current-system/profile/bin/bash --login
```

Beachten Sie, dass die *PID* nicht der Elternprozess der isolierten Umgebung sein darf, sondern PID 1 in der isolierten Umgebung oder einer seiner Kindprozesse sein muss.

10.15 `guix weather` aufrufen

Manchmal werden Sie schlecht gelaunt sein, weil es zu wenige Substitute gibt und die Pakete bei Ihnen selbst erstellt werden müssen (siehe Abschnitt 6.3 [Substitute], Seite 56). Der Befehl `guix weather` zeigt einen Bericht über die Verfügbarkeit von Substituten auf den angegebenen Servern an, damit Sie sich eine Vorstellung davon machen können, wie es heute um Ihre Laune bestellt sein wird. Manchmal bekommt man als Nutzer so hilfreiche Informationen, aber in erster Linie nützt der Befehl den Leuten, die `guix publish` benutzen (siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233).

Hier ist ein Beispiel für einen Aufruf davon:

```
$ guix weather --substitute-urls=https://guix.example.org
5.872 Paketableitungen für x86_64-linux berechnen ...
Nach 6.128 Store-Objekten von https://guix.example.org suchen ...
updating list of substitutes from 'https://guix.example.org'... 100.0%
https://guix.example.org
  43,4% Substitute verfügbar (2.658 von 6.128)
  7.032,5 MiB an Nars (komprimiert)
  19.824,2 MiB auf der Platte (unkomprimiert)
  0,030 Sekunden pro Anfrage (182,9 Sekunden insgesamt)
```

```
33,5 Anfragen pro Sekunde

9,8% (342 von 3.470) der fehlenden Objekte sind in der Warteschlange
Mindestens 867 Erstellungen in der Warteschlange
    x86_64-linux: 518 (59,7%)
    i686-linux: 221 (25,5%)
    aarch64-linux: 128 (14,8%)
Erstellungsgeschwindigkeit: 23,41 Erstellungen pro Stunde
    x86_64-linux: 11,16 Erstellungen pro Stunde
    i686-linux: 6,03 Erstellungen pro Stunde
    aarch64-linux: 6,41 Erstellungen pro Stunde
```

Wie Sie sehen können, wird der Anteil unter allen Paketen angezeigt, für die auf dem Server Substitute verfügbar sind – unabhängig davon, ob Substitute aktiviert sind, und unabhängig davon, ob der Signierschlüssel des Servers autorisiert ist. Es wird auch über die Größe der komprimierten Archive (die „Nars“) berichtet, die vom Server angeboten werden, sowie über die Größe, die die zugehörigen Store-Objekte im Store belegen würden (unter der Annahme, dass Deduplizierung abgeschaltet ist) und über den Durchsatz des Servers. Der zweite Teil sind Statistiken zur Kontinuierlichen Integration (englisch „Continuous Integration“, kurz CI), wenn der Server dies unterstützt. Des Weiteren kann `guix weather`, wenn es mit der Befehlszeilenoption `--coverage` aufgerufen wird, „wichtige“ Paketsubstitute, die auf dem Server fehlen, auflisten (siehe unten).

Dazu werden mit `guix weather` Anfragen über HTTP(S) zu Metadaten (*Narinfos*) für alle relevanten Store-Objekte gestellt. Wie `guix challenge` werden die Signaturen auf den Substituten ignoriert, was harmlos ist, weil der Befehl nur Statistiken sammelt und keine Substitute installieren kann.

Die allgemeine Syntax lautet:

```
guix weather Optionen... [Pakete...]
```

Wenn keine *Pakete* angegeben werden, prüft `guix weather` für *alle* Pakete bzw. für die Pakete mit `--manifest` angegebenen Manifest, ob Substitute zur Verfügung stehen. Ansonsten wird es nur für die angegebenen Pakete geprüft. Es ist auch möglich, die Suche mit `--system` auf bestimmte Systemtypen einzuschränken. Der Rückgabewert von `guix weather` ist *nicht* null, wenn weniger als 100% Substitute verfügbar sind.

Die verfügbaren Befehlszeilenoptionen folgen.

`--substitute-urls=`*URLs*
: *URLs* ist eine leerzeichengetrennte Liste anzufragender Substitutserver-URLs. Wird diese Befehlszeilenoption weggelassen, wird die vorgegebene Menge an Substitutservern angefragt.

`--system=`*System*
`-s` *System*
: Substitute für das *System* anfragen – z.B. für `aarch64-linux`. Diese Befehlszeilenoption kann mehrmals angegeben werden, wodurch `guix weather` die Substitute für mehrere Systemtypen anfragt.

`--manifest=`*Datei*
: Anstatt die Substitute für alle Pakete anzufragen, werden nur die in der *Datei* angegebenen Pakete erfragt. Die *Datei* muss ein *Manifest* enthalten, wie bei der

Befehlszeilenoption `-m` von `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

Wenn diese Befehlszeilenoption mehrmals wiederholt angegeben wird, werden die Manifeste aneinandergehängt.

`--coverage[=`*Anzahl*`]`
`-c [`*Anzahl*`]`

Einen Bericht über die Substitutabdeckung für Pakete ausgeben, d.h. Pakete mit mindestens *Anzahl*-vielen Abhängigen (voreingestellt mindestens null) anzeigen, für die keine Substitute verfügbar sind. Die abhängigen Pakete werden selbst nicht aufgeführt: Wenn *b* von *a* abhängt und Substitute für *a* fehlen, wird nur *a* aufgeführt, obwohl dann in der Regel auch die Substitute für *b* fehlen. Das Ergebnis sieht so aus:

```
$ guix weather --substitute-urls=https://ci.guix.gnu.org https://bordeaux.gu
8.983 Paketableitungen für x86_64-linux berechnen ...
Nach 9.343 Store-Objekten von https://ci.guix.gnu.org https://bordeaux.guix.
Liste der Substitute von „https://ci.guix.gnu.org https://bordeaux.guix.gnu.
https://ci.guix.gnu.org https://bordeaux.guix.gnu.org
  64.7% Substitute verfügbar (6.047 von 9.343)
...
2502 Pakete fehlen auf „https://ci.guix.gnu.org https://bordeaux.guix.gnu.or
    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
    ...
```

Was man hier in diesem Beispiel sehen kann, ist, dass es für `kcoreaddons` und vermutlich die 58 Pakete, die davon abhängen, auf `ci.guix.gnu.org` keine Substitute gibt; Gleiches gilt für `qgpgme` und die 46 Pakete, die davon abhängen.

Wenn Sie ein Guix-Entwickler sind oder sich um diese Erstellungsfarm kümmern, wollen Sie sich diese Pakete vielleicht genauer anschauen. Es kann sein, dass sie schlicht nie erfolgreich erstellt werden können.

`--display-missing`

Eine Liste derjenigen Store-Objekte anzeigen, für die *keine* Substitute verfügbar sind.

10.16 `guix processes` aufrufen

Der Befehl `guix processes` kann sich für Entwickler und Systemadministratoren als nützlich erweisen, besonders auf Maschinen mit mehreren Nutzern und auf Erstellungsfarmen. Damit werden die aktuellen Sitzungen (also Verbindungen zum Daemon) sowie Informationen über die beteiligten Prozesse aufgelistet[3]. Hier ist ein Beispiel für die davon gelieferten Informationen:

```
$ sudo guix processes
SessionPID: 19002
```

[3] Entfernte Sitzungen, wenn `guix-daemon` mit `--listen` unter Angabe eines TCP-Endpunkts gestartet wurde, werden *nicht* aufgelistet.

```
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
```

In diesem Beispiel sehen wir, dass `guix-daemon` drei Clients hat: `guix environment`, `guix publish` und das Werkzeug Cuirass zur Kontinuierlichen Integration. Deren Prozesskennung (PID) ist jeweils im `ClientPID`-Feld zu sehen. Das Feld `SessionPID` zeigt die PID des `guix-daemon`-Unterprozesses dieser bestimmten Sitzung.

Das Feld `LockHeld` zeigt an, welche Store-Objekte derzeit durch die Sitzung gesperrt sind, d.h. welche Store-Objekte zurzeit erstellt oder substituiert werden (das `LockHeld`-Feld wird nicht angezeigt, wenn `guix processes` nicht als Administratornutzer root ausgeführt wird). Letztlich sehen wir an den Feldern `ChildPID` und `ChildCommand` oben, dass diese drei Erstellungen hier ausgelagert (englisch „offloaded“) werden (siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13).

Die Ausgabe ist im Recutils-Format, damit wir den praktischen `recsel`-Befehl benutzen können, um uns interessierende Sitzungen auszuwählen (siehe Abschnitt “Selection Expressions” in *GNU recutils manual*). Zum Beispiel zeigt dieser Befehl die Befehlszeile und PID des Clients an, der die Erstellung des Perl-Pakets ausgelöst hat:

```
$ sudo guix processes | \
    recsel -p ClientPID,ClientCommand -e 'LockHeld ~ "perl"'
ClientPID: 19419
ClientCommand: cuirass --cache-directory /var/cache/cuirass ...
```

Weitere Befehlszeilenoptionen folgen.

`--format=`*Format*
`-f` *Format* Die Ausgabe im angegebenen *Format* generieren, was eines der Folgenden sein muss:

`recutils` Diese Option entspricht der Vorgabe. Sitzungen werden als „Session“-Datensätze im recutils-Format ausgegeben; dabei steht jeweils ein Feld `ChildProcess` für jeden Kindprozess.

`normalized`

Normalisiert die ausgegebenen Datensätze nach Typ (als „Record Sets“, siehe Abschnitt “Record Sets” in *Handbuch von GNU recutils*). Dadurch, dass die Datensatztypen in der Ausgabe stehen, wird eine Verknüpfung („Join“) zwischen den Datensatztypen möglich. Im folgenden Beispiel würde die PID jedes Kindprozesses (mit Typ `ChildProcess`) mit der PID der ihn erzeugt habenden Sitzung (Typ `Session`) angezeigt, vorausgesetzt die Sitzung wurde mit `guix build` gestartet.

```
$ 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
```

11 Fremde Architekturen

Es ist möglich, für Rechner mit einer anderen CPU-Architektur als der eigenen Pakete zu erstellen (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45), Bündel zu erstellen (siehe Abschnitt 8.3 [Aufruf von guix pack], Seite 99) oder auch vollständige Systeme aufzusetzen (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

GNU Guix unterstützt zwei unterschiedliche Mechanismen, um Software für fremde Architekturen bereitzustellen:

1. Das traditionelle Vorgehen mit Cross-Kompilierung (`https://de.wikipedia.org/wiki/Cross-Compiler`).
2. Nativ zu erstellen. Dieser Mechanismus bedeutet, dass zum Erstellen für das fremde Zielsystem der Befehlssatz des Zielrechners ausgeführt wird. Gängigerweise setzt man hierzu einen Emulator ein wie das QEMU-Programm.

11.1 Cross-Kompilieren

Für die Befehle, die Cross-Kompilierung unterstützen, werden die Befehlszeilenoptionen `--list-targets` und `--target` angeboten.

Die Befehlszeilenoption `--list-targets` listet alle unterstützten Zielsysteme auf, die Sie als Argument für `--target` angeben können.

```
$ guix build --list-targets
Die verfügbaren Zielsysteme sind:

   - aarch64-linux-gnu
   - arm-linux-gnueabihf
   - i586-pc-gnu
   - i686-linux-gnu
   - i686-w64-mingw32
   - mips64el-linux-gnu
   - powerpc-linux-gnu
   - powerpc64le-linux-gnu
   - riscv64-linux-gnu
   - x86_64-linux-gnu
   - x86_64-w64-mingw32
```

Zielsysteme werden als GNU-Tripel angegeben (siehe Abschnitt “Specifying Target Triplets” in *Autoconf*).

Diese Tripel werden an GCC und die anderen benutzten Compiler übergeben, die an der Erstellung eines Pakets, Systemabbilds oder sonstigen GNU-Guix-Ausgabe beteiligt sein können.

```
$ 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 ...
```

Der große Vorteil beim Cross-Kompilieren liegt darin, dass keine Leistungseinbußen beim Kompilieren zu befürchten sind, im Gegensatz zur Emulation mit QEMU. Jedoch besteht ein höheres Risiko, dass manche Pakete nicht erfolgreich cross-kompiliert werden können, weil sich nur wenige Leute mit diesem Mechanismus lange befassen.

11.2 Native Erstellungen

Die Befehle, die es erlauben, mit einem gewählten System Erstellungen durchzuführen, haben die Befehlszeilenoptionen `--list-systems` und `--system`.

Die Befehlszeilenoption `--list-systems` listet alle unterstützten Systeme auf, mit denen erstellt werden kann, wenn man sie als Argument für `--system` angibt.

```
$ guix build --list-systems
Die verfügbaren Systeme sind:

   - 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 ...
```

Im obigen Beispiel ist das aktuelle System *x86_64-linux*. Wir erstellen das *hello*-Paket aber für das System *i686-linux*.

Das geht, weil der *i686*-Befehlssatz der CPU eine Teilmenge des *x86_64*-Befehlssatzes ist, also Binärdateien für *i686* auf einem *x86_64*-System laufen können.

Wenn wir aber im Rahmen desselben Beispiels das System *aarch64-linux* wählen und `guix build --system=aarch64-linux hello` auch wirklich Ableitungen erstellen muss, klappt das vielleicht erst nach einem weiteren Schritt.

Denn Binärdateien mit dem Zielsystem *aarch64-linux* können auf einem *x86_64-linux*-System erst mal *nicht* ausgeführt werden. Deshalb wird nach einer Emulationsschicht gesucht. Der GNU-Guix-Daemon kann den binfmt_misc-Mechanismus (`https://en.wikipedia.org/wiki/Binfmt_misc`) des Linux-Kernels dafür einsetzen. Zusammengefasst würde der Linux-Kernel die Ausführung einer Binärdatei für eine fremde Plattform wie *aarch64-linux* auslagern auf ein Programm der Anwendungsebene („User Space“); normalerweise lagert man so auf einen Emulator aus.

Es gibt einen Dienst, der QEMU als Hintergrundprogramm für den `binfmt_misc`-Mechanismus registriert (siehe Abschnitt 12.9.29 [Virtualisierungsdienste], Seite 534). Auf Debian-basierten Fremddistributionen ist deren Alternative das Paket `qemu-user-static`.

Wenn der `binfmt_misc`-Mechanismus falsch eingerichtet ist, schlägt die Erstellung folgendermaßen fehl:

```
$ 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'...
```

Aber wenn der `binfmt_misc`-Mechanismus richtig mit QEMU verbunden wurde, kann man diese Meldung erwarten:

```
$ guix build --system=armhf-linux hello --check
/gnu/store/13xz4nghg39wpymivlwghy08yzj97hlj-hello-2.12
```

Der größte Vorteil, den man davon hat, nativ zu erstellen statt zu cross-kompilieren, ist, dass die Erstellung bei mehr Paketen klappt. Doch es hat seinen Preis: Mit QEMU als Hintergrundprogramm zu kompilieren ist *viel langsamer* als Cross-Kompilierung, weil jede Anweisung emuliert werden muss.

Die Verfügbarkeit von Substituten für die Architektur, die man bei `--system` angibt, lindert dieses Problem. Eine andere Möglichkeit ist, GNU Guix auf einer Maschine zu installieren, deren CPU den Befehlssatz tatsächlich unterstützt, für den man erstellen möchte. Diese Maschine kann man über den Mechanismus zum Auslagern von Erstellungen einbeziehen (siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13).

12 Systemkonfiguration

Guix System unterstützt einen Mechanismus zur konsistenten Konfiguration des gesamten Systems. Damit meinen wir, dass alle Aspekte der globalen Systemkonfiguration an einem Ort stehen, d.h. die zur Verfügung gestellten Systemdienste, die Zeitzone und Einstellungen zur Locale (also die Anpassung an regionale Gepflogenheiten und Sprachen) sowie Benutzerkonten. Sie alle werden an derselben Stelle deklariert. So eine *Systemkonfiguration* kann *instanziiert*, also umgesetzt, werden.

Einer der Vorteile, die ganze Systemkonfiguration unter die Kontrolle von Guix zu stellen, ist, dass so transaktionelle Systemaktualisierungen möglich werden und dass diese rückgängig gemacht werden können, wenn das aktualisierte System nicht richtig funktioniert (siehe Abschnitt 6.1 [Funktionalitäten], Seite 44). Ein anderer Vorteil ist, dass dieselbe Systemkonfiguration leicht auf einer anderen Maschine oder zu einem späteren Zeitpunkt benutzt werden kann, ohne dazu eine weitere Schicht administrativer Werkzeuge über den systemeigenen Werkzeugen einsetzen zu müssen.

In diesem Abschnitt wird dieser Mechanismus beschrieben. Zunächst betrachten wir ihn aus der Perspektive eines Administrators. Dabei wird erklärt, wie das System konfiguriert und instanziiert werden kann. Dann folgt eine Demonstration, wie der Mechanismus erweitert werden kann, etwa um neue Systemdienste zu unterstützen.

12.1 Das Konfigurationssystem nutzen

Das Betriebssystem können Sie konfigurieren, indem Sie eine `operating-system`-Deklaration in einer Datei speichern, die Sie dann dem Befehl `guix system` übergeben (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619). Eine einfache Konfiguration mit den vorgegebenen Systemdiensten und dem vorgegebenen Linux-Libre als Kernel und mit einer initialen RAM-Disk und einem Bootloader sieht so aus:

```
;; 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.
(services (append (list (service dhcp-client-service-type)
                        (service openssh-service-type
                                 (openssh-configuration
                                  (openssh openssh-sans-x)
                                  (port-number 2222))))
                  %base-services)))
```

Dieses Beispiel sollte selbsterklärend sein. Manche der Felder oben, wie etwa `host-name` und `bootloader`, müssen angegeben werden. Andere sind optional, wie etwa `packages` und `services`, sind optional; werden sie nicht angegeben, nehmen sie einen Vorgabewert an.

Im Folgenden werden die Effekte von einigen der wichtigsten Feldern erläutert (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258, für Details zu allen verfügbaren Feldern), dann wird beschrieben, wie man das Betriebssystem mit `guix system` *instanziieren* kann.

Bootloader

Das `bootloader`-Feld beschreibt, mit welcher Methode Ihr System „gebootet“ werden soll. Maschinen, die auf Intel-Prozessoren basieren, können im alten „Legacy“-BIOS-Modus ge-

bootet werden, wie es im obigen Beispiel der Fall wäre. Neuere Maschinen benutzen stattdessen das *Unified Extensible Firmware Interface* (UEFI) zum Booten. In diesem Fall sollte das `bootloader`-Feld in etwa so aussehen:

```
(bootloader-configuration
  (bootloader grub-efi-bootloader)
  (targets '("/boot/efi")))
```

Siehe den Abschnitt Abschnitt 12.14 [Bootloader-Konfiguration], Seite 613, für weitere Informationen zu den verfügbaren Konfigurationsoptionen.

global sichtbare Pakete

Im Feld `packages` werden Pakete aufgeführt, die auf dem System für alle Benutzerkonten global sichtbar sein sollen, d.h. in der `PATH`-Umgebungsvariablen jedes Nutzers, zusätzlich zu den nutzereigenen Profilen (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Die Variable `%base-packages` bietet alle Werkzeuge, die man für grundlegende Nutzer- und Administratortätigkeiten erwarten würde, einschließlich der GNU Core Utilities, der GNU Networking Utilities, des leichtgewichtigen Texteditors `mg`, `find`, `grep` und so weiter. Obiges Beispiel fügt zu diesen noch das Programm GNU Screen hinzu, welches aus dem Modul `(gnu packages screen)` genommen wird (siehe Abschnitt 9.1 [Paketmodule], Seite 108). Die Syntax `(list package output)` kann benutzt werden, um eine bestimmte Ausgabe eines Pakets auszuwählen:

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

(operating-system
  ;; ...
  (packages (cons (list isc-bind "utils")
                  %base-packages)))
```

Sich auf Pakete anhand ihres Variablennamens zu beziehen, wie oben bei `isc-bind`, hat den Vorteil, dass der Name eindeutig ist; Tippfehler werden direkt als „unbound variables“ gemeldet. Der Nachteil ist, dass man wissen muss, in welchem Modul ein Paket definiert wird, um die Zeile mit `use-package-modules` entsprechend zu ergänzen. Um dies zu vermeiden, kann man auch die Prozedur `specification->package` aus dem Modul `(gnu packages)` aufrufen, welche das einem angegebenen Namen oder Name-Versions-Paar zu Grunde liegende Paket liefert:

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

(operating-system
  ;; ...
  (packages (append (map specification->package
                         '("tcpdump" "htop" "gnupg@2.0"))
                    %base-packages)))
```

Systemdienste

Das Feld `services` listet *Systemdienste* auf, die zur Verfügung stehen sollen, wenn das System startet (siehe Abschnitt 12.9 [Dienste], Seite 280). Die `operating-system`-Deklaration

oben legt fest, dass wir neben den grundlegenden Basis-Diensten auch wollen, dass der OpenSSH-Secure-Shell-Daemon auf Port 2222 lauscht (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314). Intern sorgt der `openssh-service-type` dafür, dass `sshd` mit den richtigen Befehlszeilenoptionen aufgerufen wird, je nach Systemkonfiguration werden auch für dessen Betrieb nötige Konfigurationsdateien erstellt (siehe Abschnitt 12.18 [Dienste definieren], Seite 635).

Gelegentlich werden Sie die Basis-Dienste nicht einfach so, wie sie sind, benutzen, sondern anpassen wollen. Benutzen Sie `modify-services` (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639), um die Liste der Basis-Dienste zu modifizieren.

Wenn Sie zum Beispiel `guix-daemon` und Mingetty (das Programm, womit Sie sich auf der Konsole anmelden) in der `%base-services`-Liste modifizieren möchten (siehe Abschnitt 12.9.1 [Basisdienste], Seite 281), schreiben Sie das Folgende in Ihre Betriebssystemdeklaration:

```
(define %my-services
  ;; Meine ganz eigene Liste von Diensten.
  (modify-services %base-services
    (guix-service-type config =>
                       (guix-configuration
                        (inherit config)
                        ;; Substitute von example.org herunterladen.
                        (substitute-urls
                          (list "https://example.org/guix"
                                "https://ci.guix.gnu.org"))))
    (mingetty-service-type config =>
                           (mingetty-configuration
                            (inherit config)
                            ;; Automatisch als "gast" anmelden.
                            (auto-login "gast")))))

(operating-system
  ;; ...
  (services %my-services))
```

Dadurch ändert sich die Konfiguration – d.h. die Dienst-Parameter – der `guix-service-type`-Instanz und die aller `mingetty-service-type`-Instanzen in der `%base-services`-Liste (siehe Abschnitt "Automatisch an virtueller Konsole anmelden" in *GNU Guix Cookbook* für eine Anleitung, wie man damit ein Benutzerkonto automatisch auf nur einem TTY anmelden ließe). Das funktioniert so: Zunächst arrangieren wir, dass die ursprüngliche Konfiguration an den Bezeichner `config` im *Rumpf* gebunden wird, dann schreiben wir den *Rumpf*, damit er zur gewünschten Konfiguration ausgewertet wird. Beachten Sie insbesondere, wie wir mit `inherit` eine neue Konfiguration erzeugen, die dieselben Werte wie die alte Konfiguration hat, aber mit ein paar Modifikationen.

Die Konfiguration für typische Nutzung auf Heim- und Arbeitsrechnern, mit einer verschlüsselten Partition für das Wurzeldateisystem, darauf einer Swap-Datei, einem X11-Anzeigeserver, GNOME und Xfce (Benutzer können im Anmeldebildschirm auswählen, welche dieser Arbeitsumgebungen sie möchten, indem sie die Taste `F1` drücken), Netz-

werkverwaltung, Verwaltungswerkzeugen für den Energieverbrauch, und Weiteres, würde so aussehen:

```
;; 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 certs 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 HTTPS access
                   nss-certs
                   ;; 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))
```

Ein grafisches System mit einer Auswahl an leichtgewichtigen Fenster-Managern statt voll ausgestatteten Arbeitsumgebungen würde so aussehen:

```
;; 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 certs 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
                     ;; for HTTPS access
                     nss-certs)
                    %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))
```

Dieses Beispiel bezieht sich auf das Dateisystem hinter `/boot/efi` über dessen UUID, `1234-ABCD`. Schreiben Sie statt dieser UUID die richtige UUID für Ihr System, wie sie der Befehl `blkid` liefert.

Im Abschnitt Abschnitt 12.9.9 [Desktop-Dienste], Seite 366, finden Sie eine genaue Liste der unter `%desktop-services` angebotenen Dienste. Der Abschnitt Abschnitt 12.11 [X.509-Zertifikate], Seite 606, hat Hintergrundinformationen über das `nss-certs`-Paket, das hier benutzt wird.

Beachten Sie, dass `%desktop-services` nur eine Liste von die Dienste repräsentierenden service-Objekten ist. Wenn Sie Dienste daraus entfernen möchten, können Sie dazu die Prozeduren zum Filtern von Listen benutzen (siehe Abschnitt “SRFI-1 Filtering and Partitioning” in *Referenzhandbuch zu GNU Guile*). Beispielsweise liefert der folgende Ausdruck eine Liste mit allen Diensten von `%desktop-services` außer dem Avahi-Dienst.

```
(remove (lambda (service)
          (eq? (service-kind service) avahi-service-type))
        %desktop-services)
```

Alternativ können Sie das Makro `modify-services` benutzen:

```
(modify-services %desktop-services
  (delete avahi-service-type))
```

Das System instanziieren

Angenommen, Sie haben die `operating-system`-Deklaration in einer Datei `my-system-config.scm` gespeichert, dann instanziiert der Befehl `guix system reconfigure my-system-config.scm` diese Konfiguration und macht sie zum voreingestellten GRUB-Boot-Eintrag (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

> **Anmerkung:** Unsere Empfehlung ist, dass Sie die Datei `my-system-config.scm` sicher mit einem Versionsverwaltungssystem aufbewahren, so dass Sie die Änderungen daran leicht im Blick haben.

Der normale Weg, die Systemkonfiguration nachträglich zu ändern, ist, die Datei zu aktualisieren und `guix system reconfigure` erneut auszuführen. Man sollte nie die Dateien in `/etc` bearbeiten oder den Systemzustand mit Befehlen wie `useradd` oder `grub-install` verändern. Tatsächlich müssen Sie das ausdrücklich vermeiden, sonst verfällt nicht nur Ihre Garantie, sondern Sie können Ihr System auch nicht mehr auf eine alte Version des Systems zurücksetzen, falls das jemals notwendig wird.

Zurücksetzen bezieht sich hierbei darauf, dass jedes Mal, wenn Sie `guix system reconfigure` ausführen, eine neue *Generation* des Systems erzeugt wird – ohne vorherige Generationen zu verändern. Alte Systemgenerationen bekommen einen Eintrag im Boot-Menü des Bootloaders, womit Sie alte Generationen beim Starten des Rechners auswählen können, wenn mit der neuesten Generation etwas nicht stimmt. Eine beruhigende Vorstellung, oder? Der Befehl `guix system list-generations` führt die auf der Platte verfügbaren Systemgenerationen auf. Es ist auch möglich, das System mit den Befehlen `guix system roll-back` und `guix system switch-generation` zurückzusetzen.

Obwohl der Befehl `guix system reconfigure` vorherige Generationen nicht verändern wird, müssen Sie achtgeben, dass wenn die momentan aktuelle Generation nicht die neueste ist (z.B. nach einem Aufruf von `guix system roll-back`), weil `guix system reconfigure` alle neueren Generationen überschreibt (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

Die Programmierschnittstelle

Auf der Ebene von Scheme wird der Großteil der `operating-system`-Deklaration mit der folgenden monadischen Prozedur instanziiert (siehe Abschnitt 9.11 [Die Store-Monade], Seite 170):

`operating-system-derivation` *os* [Monadische Prozedur]

Liefert eine Ableitung, mit der ein `operating-system`-Objekt *os* erstellt wird (siehe Abschnitt 9.10 [Ableitungen], Seite 167).

Die Ausgabe der Ableitung ist ein einzelnes Verzeichnis mit Verweisen auf alle Pakete, Konfigurationsdateien und andere unterstützenden Dateien, die nötig sind, um *os* zu instanziieren.

Diese Prozedur wird vom Modul `(gnu system)` angeboten. Zusammen mit `(gnu services)` (siehe Abschnitt 12.9 [Dienste], Seite 280) deckt dieses Modul den Kern von „Guix System“ ab. Schauen Sie es sich mal an!

12.2 `operating-system`-Referenz

Dieser Abschnitt fasst alle Optionen zusammen, die für `operating-system`-Deklarationen zur Verfügung stehen (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250).

`operating-system` [Datentyp]

Der die Betriebssystemkonfiguration repräsentierende Datentyp. Damit meinen wir die globale Konfiguration des Systems und nicht die, die sich nur auf einzelne Nutzer bezieht (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250).

`kernel` (Vorgabe: `linux-libre`)
: Das Paket für den zu nutzenden Betriebssystem-Kernel als „package“-Objekt[1].

`hurd` (Vorgabe: `#f`)
: Das Paketobjekt derjenigen Hurd, die der Kernel starten soll. Wenn dieses Feld gesetzt ist, wird ein GNU/Hurd-Betriebssystem erzeugt. In diesem Fall muss als `kernel` das `gnumach`-Paket (das ist der Microkernel, auf dem Hurd läuft) ausgewählt sein.

  Warnung: Diese Funktionalität ist experimentell und wird nur für Disk-Images unterstützt.

`kernel-loadable-modules` (Vorgabe: '())
: Eine Liste von Objekten (normalerweise Pakete), aus denen Kernel-Module geladen werden können, zum Beispiel `(list ddcci-driver-linux)`.

`kernel-arguments` (Vorgabe: `%default-kernel-arguments`)
: Eine Liste aus Zeichenketten oder G-Ausdrücken, die für zusätzliche Argumente an den Kernel stehen, die ihm auf seiner Befehlszeile übergeben werden – wie z.B. `("console=ttyS0")`.

`bootloader`
: Das Konfigurationsobjekt für den Bootloader, mit dem das System gestartet wird. Siehe Abschnitt 12.14 [Bootloader-Konfiguration], Seite 613.

`label` Diese Bezeichnung (eine Zeichenkette) wird für den Menüeintrag im Bootloader verwendet. Die Vorgabe ist eine Bezeichnung, die den Namen des Kernels und seine Version enthält.

`keyboard-layout` (Vorgabe: `#f`)
: Dieses Feld gibt an, welche Tastaturbelegung auf der Konsole benutzt werden soll. Es kann entweder auf `#f` gesetzt sein, damit die voreingestellte Tastaturbelegung benutzt wird (in der Regel ist diese „US English“), oder ein `<keyboard-layout>`-Verbundsobjekt sein. Siehe Abschnitt 12.7 [Tastaturbelegung], Seite 276, für weitere Informationen.

  Diese Tastaturbelegung wird benutzt, sobald der Kernel gebootet wurde. Diese Tastaturbelegung wird zum Beispiel auch verwendet, wenn Sie eine Passphrase eintippen, falls sich Ihr Wurzeldateisystem auf einem mit `luks-device-mapping` zugeordneten Gerät befindet (siehe Abschnitt 12.4 [Zugeordnete Geräte], Seite 269).

  Anmerkung: Damit wird *nicht* angegeben, welche Tastaturbelegung der Bootloader benutzt, und auch nicht, welche der grafische Anzeigeserver verwendet. Siehe Abschnitt 12.14 [Bootloader-Konfiguration], Seite 613, für Informationen darüber, wie Sie die Tastaturbelegung des

[1] Derzeit wird nur der Kernel Linux-libre vollständig unterstützt. Die Nutzung von GNU mach mit GNU Hurd ist experimentell und steht nur zur Erstellung eines Disk-Image für virtuelle Maschinen bereit.

Bootloaders angeben können. Siehe Abschnitt 12.9.7 [X Window], Seite 342, für Informationen darüber, wie Sie die Tastaturbelegung angeben können, die das X-Fenstersystem verwendet.

`initrd-modules` (Vorgabe: `%base-initrd-modules`)
: Die Liste der Linux-Kernel-Module, die in der initialen RAM-Disk zur Verfügung stehen sollen. Siehe Abschnitt 12.13 [Initiale RAM-Disk], Seite 609.

`initrd` (Vorgabe: `base-initrd`)
: Eine Prozedur, die eine initiale RAM-Disk für den Linux-Kernel liefert. Dieses Feld gibt es, damit auch sehr systemnahe Anpassungen vorgenommen werden können, aber für die normale Nutzung sollten Sie es kaum brauchen. Siehe Abschnitt 12.13 [Initiale RAM-Disk], Seite 609.

`firmware` (Vorgabe: `%base-firmware`)
: Eine Liste der Firmware-Pakete, die vom Betriebssystem-Kernel geladen werden können.

  Vorgegeben ist, dass für Atheros- und Broadcom-basierte WLAN-Geräte nötige Firmware geladen werden kann (genauer jeweils die Linux-libre-Module `ath9k` und `b43-open`). Siehe den Abschnitt Abschnitt 3.2 [Hardware-Überlegungen], Seite 27, für mehr Informationen zu unterstützter Hardware.

`host-name`
: Der Rechnername.

`hosts-file`
: Ein dateiartiges Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), das für `/etc/hosts` benutzt werden soll (siehe Abschnitt "Host Names" in *Referenzhandbuch der GNU-C-Bibliothek*). Der Vorgabewert ist eine Datei mit Einträgen für `localhost` und *host-name*.

`mapped-devices` (Vorgabe: `'()`)
: Eine Liste zugeordneter Geräte („mapped devices"). Siehe Abschnitt 12.4 [Zugeordnete Geräte], Seite 269.

`file-systems`
: Eine Liste von Dateisystemen. Siehe Abschnitt 12.3 [Dateisysteme], Seite 263.

`swap-devices` (Vorgabe: `'()`)
: Eine Liste der Swap-Speicher. Siehe Abschnitt 12.5 [Swap-Speicher], Seite 271.

`users` (Vorgabe: `%base-user-accounts`)
`groups` (Vorgabe: `%base-groups`)
: Liste der Benutzerkonten und Benutzergruppen. Siehe Abschnitt 12.6 [Benutzerkonten], Seite 273.

Wenn in der `users`-Liste kein Benutzerkonto mit der UID-Kennung 0 aufgeführt wird, wird automatisch für den Administrator ein „root“-Benutzerkonto mit UID-Kennung 0 hinzugefügt.

`skeletons` (Vorgabe: `(default-skeletons)`)
Eine Liste von Tupeln aus je einem Ziel-Dateinamen und einem dateiähnlichen Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175). Diese Objekte werden als Skeleton-Dateien im Persönlichen Verzeichnis („Home“-Verzeichnis) jedes neuen Benutzerkontos angelegt.

Ein gültiger Wert könnte zum Beispiel so aussehen:

```
`((".bashrc" ,(plain-file "bashrc" "echo Hallo\n"))
  (".guile" ,(plain-file "guile"
                         "(use-modules (ice-9 readline))
                          (activate-readline)")))
```

`issue` (Vorgabe: `%default-issue`)
Eine Zeichenkette, die als Inhalt der Datei `/etc/issue` verwendet werden soll, der jedes Mal angezeigt wird, wenn sich ein Nutzer auf einer Textkonsole anmeldet.

`packages` (Vorgabe: `%base-packages`)
Eine Liste von Paketen, die ins globale Profil installiert werden sollen, welches unter `/run/current-system/profile` zu finden ist. Jedes Element ist entweder eine Paketvariable oder ein Tupel aus einem Paket und dessen gewünschter Ausgabe. Hier ist ein Beispiel:

```
(cons* git                       ; die Standardausgabe "out"
       (list git "send-email") ; eine andere Ausgabe von git
       %base-packages)           ; die normale Paketmenge
```

Die vorgegebene Paketmenge umfasst zum Kern des Systems gehörende Werkzeuge („core utilities“). Es ist empfehlenswert, nicht zum Kern gehörende Werkzeuge („non-core“) stattdessen in Nutzerprofile zu installieren (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

`timezone` (Vorgabe: `"Etc/UTC"`)
Eine Zeichenkette, die die Zeitzone bezeichnet, wie z.B. `"Europe/Berlin"`.

Mit dem Befehl `tzselect` können Sie herausfinden, welche Zeichenkette der Zeitzone Ihrer Region entspricht. Wenn Sie eine ungültige Zeichenkette angeben, schlägt `guix system` fehl.

`locale` (Vorgabe: `"en_US.utf8"`)
Der Name der als Voreinstellung zu verwendenden Locale (siehe Abschnitt “Locale Names” in *Referenzhandbuch der GNU-C-Bibliothek*). Siehe Abschnitt 12.8 [Locales], Seite 278, für weitere Informationen.

`locale-definitions` (Vorgabe: `%default-locale-definitions`)
Die Liste der Locale-Definitionen, die kompiliert werden sollen und dann im laufenden System benutzt werden können. Siehe Abschnitt 12.8 [Locales], Seite 278.

`locale-libcs` (Vorgabe: `(list glibc)`)
: Die Liste der GNU-libc-Pakete, deren Locale-Daten und -Werkzeuge zum Erzeugen der Locale-Definitionen verwendet werden sollen. Siehe Abschnitt 12.8 [Locales], Seite 278, für eine Erläuterung der Kompatibilitätsauswirkungen, deretwegen man diese Option benutzen wollen könnte.

`name-service-switch` (Vorgabe: `%default-nss`)
: Die Konfiguration des Name Service Switch (NSS) der libc – ein `<name-service-switch>`-Objekt. Siehe Abschnitt 12.12 [Name Service Switch], Seite 607, für Details.

`services` (Vorgabe: `%base-services`)
: Eine Liste von „service“-Objekten, die die Systemdienste repräsentieren. Siehe Abschnitt 12.9 [Dienste], Seite 280.

`essential-services` (Vorgabe: ...)
: Die Liste „essenzieller Dienste“ – d.h. Dinge wie Instanzen von `system-service-type` und `host-name-service-type` (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639), die aus der Betriebssystemdefinition an sich abgeleitet werden. Als normaler Benutzer sollten Sie dieses Feld *niemals* ändern müssen.

`pam-services` (Vorgabe: `(base-pam-services)`)
: Dienste für *Pluggable Authentication Modules* (PAM) von Linux.

`setuid-programs` (Vorgabe: `%setuid-programs`)
: Eine Liste von `<setuid-program>`-Objekten. Siehe Abschnitt 12.10 [Setuid-Programme], Seite 604.

`sudoers-file` (Vorgabe: `%sudoers-specification`)
: Der Inhalt der Datei `/etc/sudoers` als ein dateiähnliches Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

  Diese Datei gibt an, welche Nutzer den Befehl `sudo` benutzen dürfen, was sie damit tun und welche Berechtigungen sie so erhalten können. Die Vorgabe ist, dass nur der Administratornutzer `root` und Mitglieder der Benutzergruppe `wheel` den `sudo`-Befehl verwenden dürfen.

`this-operating-system` [Scheme-Syntax]
: Wenn dies im *lexikalischen Geltungsbereich* der Definition eines Feldes im Betriebssystem steht, bezieht sich dieser Bezeichner auf das Betriebssystem, das gerade definiert wird.

  Das folgende Beispiel zeigt, wie man auf das Betriebssystem, das gerade definiert wird, verweist, während man die Definition des `label`-Felds schreibt:

```
(use-modules (gnu) (guix))

(operating-system
  ;; ...
  (label (package-full-name
          (operating-system-kernel this-operating-system))))
```

Es ist ein Fehler, außerhalb einer Betriebssystemdefinition auf `this-operating-system` zu verweisen.

12.3 Dateisysteme

Die Liste der Dateisysteme, die eingebunden werden sollen, steht im `file-systems`-Feld der Betriebssystemdeklaration (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250). Jedes Dateisystem wird mit der `file-system`-Form deklariert, etwa so:

```
(file-system
  (mount-point "/home")
  (device "/dev/sda3")
  (type "ext4"))
```

Wie immer müssen manche Felder angegeben werden – die, die im Beispiel oben stehen –, während andere optional sind. Die Felder werden nun beschrieben.

`file-system` [Datentyp]

Objekte dieses Typs repräsentieren einzubindende Dateisysteme. Sie weisen folgende Komponenten auf:

`type` Eine Zeichenkette, die den Typ des Dateisystems spezifiziert, z.B. `"ext4"`.

`mount-point`
Der Einhängepunkt, d.h. der Pfad, an dem das Dateisystem eingebunden werden soll.

`device` Hiermit wird die „Quelle“ des Dateisystems bezeichnet. Sie kann eines von drei Dingen sein: die Bezeichnung („Labels“) eines Dateisystems, die UUID-Kennung des Dateisystems oder der Name eines `/dev`-Knotens. Mit Bezeichnungen und UUIDs können Sie Dateisysteme benennen, ohne den Gerätenamen festzuschreiben[2].

Dateisystem-Bezeichnungen („Labels“) werden mit der Prozedur `file-system-label` erzeugt und UUID-Kennungen werden mit `uuid` erzeugt, während Knoten in `/dev` mit ihrem Pfad als einfache Zeichenketten aufgeführt werden. Hier ist ein Beispiel, wie wir ein Dateisystem anhand seiner Bezeichnung aufführen, wie sie vom Befehl `e2label` angezeigt wird:

```
(file-system
  (mount-point "/home")
  (type "ext4")
  (device (file-system-label "my-home")))
```

UUID-Kennungen werden mit der `uuid`-Form von ihrer Darstellung als Zeichenkette (wie sie vom Befehl `tune2fs -l` angezeigt wird) konvertiert[3] wie hier:

[2] Beachten Sie: Obwohl es verführerisch ist, mit `/dev/disk/by-uuid` und ähnlichen Gerätenamen dasselbe Resultat bekommen zu wollen, raten wir davon ab: Diese speziellen Geräte werden erst vom udev-Daemon erzeugt und sind, wenn die Geräte eingebunden werden, vielleicht noch nicht verfügbar.

[3] Die `uuid`-Form nimmt 16-Byte-UUIDs entgegen, wie sie in RFC 4122 (`https://tools.ietf.org/html/rfc4122`) definiert sind. Diese Form der UUID wird unter anderem von der ext2-Familie von Dateisystemen verwendet, sie unterscheidet sich jedoch zum Beispiel von den „UUID“ genannten Kennungen, wie man sie bei FAT-Dateisystemen findet.

```
(file-system
  (mount-point "/home")
  (type "ext4")
  (device (uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")))
```

Wenn die Quelle eines Dateisystems ein zugeordnetes Gerät (siehe Abschnitt 12.4 [Zugeordnete Geräte], Seite 269) ist, *muss* sich das `device`-Feld auf den zugeordneten Gerätenamen beziehen – z.B. `"/dev/mapper/root-partition"`. Das ist nötig, damit das System weiß, dass das Einbinden des Dateisystems davon abhängt, die entsprechende Gerätezuordnung hergestellt zu haben.

`flags` (Vorgabe: `'()`)

Eine Liste von Symbolen, die Einbinde-Schalter („mount flags“) bezeichnen. Erkannt werden unter anderem `read-only` (nur lesbar), `bind-mount` (Verzeichniseinbindung), `no-dev` (Zugang zu besonderen Dateien verweigern), `no-suid` (setuid- und setgid-Bits ignorieren), `no-atime` (Dateizugriffs-Zeitstempel *nicht* aktualisieren), `no-diratime` (das Gleiche ausschließlich für Verzeichnisse), `strict-atime` (Dateizugriffs-Zeitstempel immer aktualisieren), `lazy-time` (Zeitstempel nur auf zwischengespeicherten Datei-Inodes im Arbeitsspeicher aktualisieren), `no-exec` (Programmausführungen verweigern) und `shared` (für Mehrfacheinhängungen). Siehe Abschnitt “Mount-Unmount-Remount” in *Referenzhandbuch der GNU-C-Bibliothek* für mehr Informationen zu diesen Einbinde-Schaltern.

`options` (Vorgabe: `#f`)

Entweder `#f` oder eine Zeichenkette mit Einbinde-Optionen („mount options“), die an den Dateisystemtreiber übergeben werden. Siehe Abschnitt “Mount-Unmount-Remount” in *Referenzhandbuch der GNU-C-Bibliothek* für Details.

Führen Sie `man 8 mount` aus, um die Einbinde-Optionen verschiedener Dateisysteme zu sehen. Aber aufgepasst: Wenn dort von „vom Dateisystem unabhängigen Einhängeoptionen“ die Rede ist, sind eigentlich Flags gemeint; sie gehören in das oben beschriebene `flags`-Feld.

Die Prozeduren `file-system-options->alist` und `alist->file-system-options` aus `(gnu system file-systems)` können benutzt werden, um als assoziative Liste dargestellte Dateisystemoptionen in eine Darstellung als Zeichenkette umzuwandeln und umgekehrt.

`mount?` (Vorgabe: `#t`)

Dieser Wert zeigt an, ob das Dateisystem automatisch eingebunden werden soll, wenn das System gestartet wird. Ist der Wert `#f`, dann erhält das Dateisystem nur einen Eintrag in der Datei `/etc/fstab` (welche vom `mount`-Befehl zum Einbinden gelesen wird), es wird aber nicht automatisch eingebunden.

`needed-for-boot?` (Vorgabe: `#f`)

Dieser boolesche Wert gibt an, ob das Dateisystem zum Hochfahren des Systems notwendig ist. In diesem Fall wird das Dateisystem eingebunden,

wenn die initiale RAM-Disk (initrd) geladen wird. Für zum Beispiel das Wurzeldateisystem ist dies ohnehin immer der Fall.

`check?` (Vorgabe: `#t`)
: Dieser boolesche Wert sagt aus, ob das Dateisystem vor dem Einbinden auf Fehler hin geprüft werden soll. Feineinstellungen, wie und wann geprüft wird, sind mit den folgenden Optionen möglich.

`skip-check-if-clean?` (Vorgabe: `#t`)
: Wenn es wahr ist, zeigt dieser Boolesche Ausdruck an, ob eine durch `check?` ausgelöste Dateisystemüberprüfung direkt abbrechen darf, wenn das Dateisystem als in Ordnung („clean“) markiert ist, also zuvor korrekt ausgehangen wurde, so dass es keine Fehler enthalten dürfte.

  Wenn Sie es auf falsch setzen, wird eine vollständige Konsistenzprüfung bei jedem Start erzwungen, wenn `check?` auf wahr gesetzt ist. Das kann sehr viel Zeit in Anspruch nehmen. Auf gesunden Systemen lassen Sie es besser eingeschaltet, sonst kann die Verlässlichkeit sogar abnehmen!

  Andererseits speichern Dateisysteme wie `fat` *nicht*, ob der Rechner ordentlich heruntergefahren wurde, deswegen wird diese Option für sie ignoriert.

`repair` (Vorgabe: `'preen`)
: Wenn durch `check?` Fehler festgestellt wurden, kann es versuchen, das Dateisystem zu reparieren, und das System danach normal starten. Mit dieser Option legen Sie fest, wann und wie repariert werden soll.

  Wenn es falsch ist, wird das Dateisystem möglichst unverändert gelassen. Beim Überprüfen mancher Dateisysteme wie `jfs` können trotzdem Schreibzugriffe auf das Gerät stattfinden, um die Aufzeichnungen über Operationen (das „Journal“) zu wiederholen. Es wird *keine* Reparatur versucht.

  Wenn es `#t` ist, wird versucht, alle gefundenen Fehler zu beheben. Auf alle Rückfragen wird mit „yes“ geantwortet. Dadurch werden die meisten Fehler behoben, aber es kann schiefgehen.

  Wenn es `'preen` ist, werden nur solche Fehler behoben, wo auch ohne menschliche Aufsicht nichts Schlimmes passieren kann. Was das genau heißt, bleibt den Entwicklern des jeweiligen Dateisystems überlassen. Es kann auf dasselbe hinauslaufen wie keine oder alle Fehler zu beheben.

`create-mount-point?` (Vorgabe: `#f`)
: Steht dies auf wahr, wird der Einhängepunkt vor dem Einbinden erstellt, wenn er noch nicht existiert.

`mount-may-fail?` (Vorgabe: `#f`)
: Wenn dies auf wahr steht, bedeutet es, dass das Einbinden dieses Dateisystems scheitern kann, dies aber nicht als Fehler aufgefasst werden soll. Das braucht man in besonderen Fällen, zum Beispiel wird es für `efivarfs` benutzt, einem Dateisystem, das nur auf EFI-/UEFI-Systemen eingebunden werden kann.

`dependencies` (Vorgabe: `'()`)

Dies ist eine Liste von `<file-system>`- oder `<mapped-device>`-Objekten, die Dateisysteme repräsentieren, die vor diesem Dateisystem eingebunden oder zugeordnet werden müssen (und nach diesem ausgehängt oder geschlossen werden müssen).

Betrachten Sie zum Beispiel eine Hierarchie von Einbindungen: `/sys/fs/cgroup` ist eine Abhängigkeit von `/sys/fs/cgroup/cpu` und `/sys/fs/cgroup/memory`.

Ein weiteres Beispiel ist ein Dateisystem, was von einem zugeordneten Gerät abhängt, zum Beispiel zur Verschlüsselung einer Partition (siehe Abschnitt 12.4 [Zugeordnete Geräte], Seite 269).

`file-system-label` *Zeichenkette* [Scheme-Prozedur]

Diese Prozedur kapselt die *Zeichenkette* in einer opaken Dateisystembezeichnung:

```
(file-system-label "home")
⇒ #<file-system-label "home">
```

Mit Dateisystembezeichnungen werden Dateisysteme anhand ihrer Bezeichnung („Label“) statt ihres Gerätenamens („Device Name“) identifiziert. Siehe die Beispiele oben.

Das Modul `(gnu system file-systems)` exportiert die folgenden nützlichen Variablen.

`%base-file-systems` [Scheme-Variable]

Hiermit werden essenzielle Dateisysteme bezeichnet, die für normale Systeme unverzichtbar sind, wie zum Beispiel `%pseudo-terminal-file-system` und `%immutable-store` (siehe unten). Betriebssystemdeklaration sollten auf jeden Fall mindestens diese enthalten.

`%pseudo-terminal-file-system` [Scheme-Variable]

Das als `/dev/pts` einzubindende Dateisystem. Es unterstützt über `openpty` und ähnliche Funktionen erstellte *Pseudo-Terminals* (siehe Abschnitt “Pseudo-Terminals” in *Referenzhandbuch der GNU-C-Bibliothek*). Pseudo-Terminals werden von Terminal-Emulatoren wie `xterm` benutzt.

`%shared-memory-file-system` [Scheme-Variable]

Dieses Dateisystem wird als `/dev/shm` eingebunden, um Speicher zwischen Prozessen teilen zu können (siehe Abschnitt “Memory-mapped I/O” in *Referenzhandbuch der GNU-C-Bibliothek*).

`%immutable-store` [Scheme-Variable]

Dieses Dateisystem vollzieht eine Verzeichniseinbindung („bind mount“) des `/gnu/store`, um ihn für alle Nutzer einschließlich des Administratornutzers `root` nur lesbar zu machen, d.h. Schreibrechte zu entziehen. Dadurch kann als `root` ausgeführte Software, oder der Systemadministrator, nicht aus Versehen den Store modifizieren.

Der Daemon kann weiterhin in den Store schreiben, indem er ihn selbst mit Schreibrechten in seinem eigenen „Namensraum“ einbindet.

`%binary-format-file-system` [Scheme-Variable]
: Das `binfmt_misc`-Dateisystem, durch das beliebige Dateitypen als ausführbare Dateien auf der Anwendungsebene (dem User Space) zugänglich gemacht werden können. Es setzt voraus, dass das Kernel-Modul `binfmt.ko` geladen wurde.

`%fuse-control-file-system` [Scheme-Variable]
: Das `fusectl`-Dateisystem, womit „unprivilegierte“ Nutzer ohne besondere Berechtigungen im User Space FUSE-Dateisysteme einbinden und aushängen können. Dazu muss das Kernel-Modul `fuse.ko` geladen sein.

Das Modul `(gnu system uuid)` stellt Werkzeug zur Verfügung, um mit eindeutigen Identifikatoren für Dateisysteme umzugehen (sogenannten „Unique Identifiers“, UUIDs).

`uuid` *Zeichenkette* [*Typ*] [Scheme-Prozedur]
: Liefert eine eindeutige UUID (Unique Identifier) als opakes Objekt des angegebenen *Typ*s (ein Symbol), indem die *Zeichenkette* verarbeitet wird:

```
(uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")
⇒ #<<uuid> type: dce bv: ...>

(uuid "1234-ABCD" 'fat)
⇒ #<<uuid> type: fat bv: ...>
```

: Als *Typ* kann entweder `dce`, `iso9660`, `fat`, `ntfs` oder eines der üblichen Synonyme dafür angegeben werden.

: UUIDs bieten eine andere Möglichkeit, sich in der Betriebssystemkonfiguration ohne Mehrdeutigkeiten auf eines der Dateisysteme zu beziehen. Siehe die Beispiele oben.

12.3.1 Btrfs-Dateisystem

Das Btrfs-Dateisystem bietet besondere Funktionalitäten, wie z.B. Unterlaufwerke („Subvolumes“), die eine detailliertere Erklärung verdienen. Im folgenden Abschnitt wird versucht, grundlegende sowie komplexe Anwendungsmöglichkeiten eines Btrfs-Dateisystem für Guix System abzudecken.

Im einfachsten Fall kann ein Btrfs-Dateisystem durch einen Ausdruck wie hier beschrieben werden:

```
(file-system
  (mount-point "/home")
  (type "btrfs")
  (device (file-system-label "my-home")))
```

Nun folgt ein komplexeres Beispiel, bei dem ein Btrfs-Unterlaufwerk namens `rootfs` benutzt wird. Dessen Eltern-Btrfs-Dateisystem wird mit `my-btrfs-pool` bezeichnet und befindet sich auf einem verschlüsselten Gerät (daher die Abhängigkeit von `mapped-devices`):

```
(file-system
  (device (file-system-label "my-btrfs-pool"))
  (mount-point "/")
  (type "btrfs")
  (options "subvol=rootfs")
  (dependencies mapped-devices))
```

Manche Bootloader, wie zum Beispiel GRUB, binden von einer Btrfs-Partition zuerst beim frühen Boot („early boot“) nur die oberste Ebene ein und verlassen sich darauf, dass ihre Konfiguration den korrekten Pfad samt Unterlaufwerk innerhalb dieser obersten Ebene enthält. Auf diese Weise arbeitende Bootloader erzeugen ihre Konfiguration normalerweise auf einem laufenden System, auf dem die Btrfs-Partitionen bereits eingebunden sind und die Informationen über die Unterlaufwerke zur Verfügung stehen. Zum Beispiel liest `grub-mkconfig`, der bei GRUB mitgelieferte Befehl zur Erzeugung von Konfigurationsdateien, aus `/proc/self/mountinfo`, um festzustellen, was auf oberster Ebene der Pfad zum Unterlaufwerk ist.

Guix System hingegen erzeugt eine Bootloader-Konfiguration mit der Betriebssystemkonfiguration als einzige Eingabe. Daher muss der Name des Unterlaufwerks, auf dem sich `/gnu/store` befindet (falls Sie eines benutzen) aus derselben Betriebssystemkonfiguration kommen. Um das besser zu veranschaulichen, betrachten Sie ein Unterlaufwerk namens „rootfs“, das die Daten des Wurzeldateisystems speichert. In einer solchen Situation würde der GRUB-Bootloader nur die oberste Ebene der Wurzel-Btrfs-Partition sehen, z.B.:

```
/                  (oberste Ebene)
 rootfs          (Unterlaufwerk als Verzeichnis)
     gnu         (normales Verzeichnis)
         store   (normales Verzeichnis)
[...]
```

Deswegen muss der Name des Unterlaufwerks dem `/gnu/store`-Pfad des Kernels, der initrd und jeder anderen Datei vorangestellt werden, die die GRUB-Konfiguration referenziert und während des frühen Boots gefunden werden können muss.

Das nächste Beispiel zeigt eine verschachtelte Hierarchie aus Unterlaufwerken und Verzeichnissen:

```
/                  (oberste Ebene)
 rootfs          (Unterlaufwerk)
     gnu         (normales Verzeichnis)
         store   (Unterlaufwerk)
[...]
```

Dieses Szenario würde ohne Einbinden des „store“-Unterlaufwerks funktionieren. „rootfs“ genügt, weil der Name des Unterlaufwerks dem dafür vorgesehenen Einhängepunkt in der Dateisystemhierarchie entspricht. Alternativ könnte man das „store“-Unterlaufwerk durch Festlegen der `subvol`-Option auf entweder `/rootfs/gnu/store` oder `rootfs/gnu/store` verwenden.

Abschließend folgt ein ausgeklügelteres Beispiel verschachtelter Unterlaufwerke:

```
/                         (oberste Ebene)
 root-snapshots          (Unterlaufwerk)
     root-current        (Unterlaufwerk)
         guix-store      (Unterlaufwerk)
[...]
```

Hier stimmt das „guix-store“-Unterlaufwerk nicht mit dem vorgesehenen Einhängepunkt überein, daher muss es eingebunden werden. Das Unterlaufwerk muss vollständig spezifiziert werden, indem sein Dateiname an die `subvol`-Option übergeben wird. Eine Möglichkeit

wäre, das „guix-store“-Unterlaufwerk als `/gnu/store` über eine solche Dateisystemdeklaration einzubinden:

```
(file-system
  (device (file-system-label "btrfs-pool-1"))
  (mount-point "/gnu/store")
  (type "btrfs")
  (options "subvol=root-snapshots/root-current/guix-store,\
compress-force=zstd,space_cache=v2"))
```

12.4 Zugeordnete Geräte

Der Linux-Kernel unterstützt das Konzept der *Gerätezuordnung*: Ein blockorientiertes Gerät wie eine Festplattenpartition kann einem neuen Gerät *zugeordnet* werden, gewöhnlich unter `/dev/mapper/`, wobei das neue Gerät durchlaufende Daten zusätzlicher Verarbeitung unterzogen werden[4]. Ein typisches Beispiel ist eine Gerätezuordnung zur Verschlüsselung: Jeder Schreibzugriff auf das zugeordnete Gerät wird transparent verschlüsselt und jeder Lesezugriff ebenso entschlüsselt. Guix erweitert dieses Konzept, indem es darunter jedes Gerät und jede Menge von Geräten versteht, die auf irgendeine Weise *umgewandelt* wird, um ein neues Gerät zu bilden; zum Beispiel entstehen auch RAID-Geräte aus einem *Verbund* mehrerer anderer Geräte, wie etwa Festplatten oder Partition zu einem einzelnen Gerät, das sich wie eine Partition verhält.

Zugeordnete Geräte werden mittels einer `mapped-device`-Form deklariert, die wie folgt definiert ist; Beispiele folgen weiter unten.

`mapped-device` [Datentyp]

Objekte dieses Typs repräsentieren Gerätezuordnungen, die gemacht werden, wenn das System hochfährt.

`source` Es handelt sich entweder um eine Zeichenkette, die den Namen eines zuzuordnenden blockorientierten Geräts angibt, wie `"/dev/sda3"`, oder um eine Liste solcher Zeichenketten, sofern mehrere Geräts zu einem neuen Gerät verbunden werden. Im Fall von LVM ist es eine Zeichenkette, die den Namen der zuzuordnenden Datenträgergruppe (Volume Group) angibt.

`target` Diese Zeichenkette gibt den Namen des neuen zugeordneten Geräts an. Bei Kernel-Zuordnern, wie verschlüsselten Geräten vom Typ `luks-device-mapping`, wird durch Angabe von `"my-partition"` ein Gerät `"/dev/mapper/my-partition"` erzeugt. Bei RAID-Geräten vom Typ `raid-device-mapping` muss der Gerätename als voller Pfad wie zum Beispiel `"/dev/md0"` angegeben werden. Logische Datenträger von LVM („LVM logical volumes“) vom Typ `lvm-device-mapping` müssen angegeben werden als `"DATENTRÄGERGRUPPENNAME-LOGISCHERDATENTRÄGERNAME"`.

[4] Beachten Sie, dass mit GNU Hurd kein Unterschied zwischen dem Konzept eines „zugeordneten Geräts“ und dem eines Dateisystems besteht: Dort werden bei beiden Ein- und Ausgabeoperationen auf eine Datei in Operationen auf dessen Hintergrundspeicher *übersetzt*. Hurd implementiert zugeordnete Geräte genau wie Dateisysteme mit dem generischen *Übersetzer*-Mechanismus (siehe Abschnitt “Translators” in *Referenzhandbuch von GNU Hurd*).

`targets` Diese Liste von Zeichenketten gibt die Namen der neuen zugeordneten Geräte an, wenn es mehrere davon gibt. Das Format ist mit *target* identisch.

`type` Dies muss ein `mapped-device-kind`-Objekt sein, das angibt, wie die Quelle *source* dem Ziel *target* zugeordnet wird.

`luks-device-mapping` [Scheme-Variable]
Hiermit wird ein blockorientiertes Gerät mit LUKS verschlüsselt, mit Hilfe des Befehls `cryptsetup` aus dem gleichnamigen Paket. Dazu wird das Linux-Kernel-Modul `dm-crypt` vorausgesetzt.

`raid-device-mapping` [Scheme-Variable]
Dies definiert ein RAID-Gerät, das mit dem Befehl `mdadm` aus dem gleichnamigen Paket als Verbund zusammengestellt wird. Es setzt voraus, dass das Linux-Kernel-Modul für das entsprechende RAID-Level geladen ist, z.B. `raid456` für RAID-4, RAID-5 oder RAID-6, oder `raid10` für RAID-10.

`lvm-device-mapping` [Scheme-Variable]
Hiermit wird ein oder mehrere logische Datenträger („Logical Volumes“) für den Logical Volume Manager (LVM) (`https://www.sourceware.org/lvm2/`) für Linux definiert. Die Datenträgergruppe („Volume Group“) wird durch den Befehl `vgchange` aus dem `lvm2`-Paket aktiviert.

Das folgende Beispiel gibt eine Zuordnung von `/dev/sda3` auf `/dev/mapper/home` mit LUKS an – dem Linux Unified Key Setup (`https://gitlab.com/cryptsetup/cryptsetup`), einem Standardmechanismus zur Plattenverschlüsselung. Das Gerät `/dev/mapper/home` kann dann als `device` einer `file-system`-Deklaration benutzt werden (siehe Abschnitt 12.3 [Dateisysteme], Seite 263).

```
(mapped-device
  (source "/dev/sda3")
  (target "home")
  (type luks-device-mapping))
```

Um nicht davon abhängig zu sein, wie Ihre Geräte nummeriert werden, können Sie auch die LUKS-UUID (*unique identifier*, d.h. den eindeutigen Bezeichner) des Quellgeräts auf der Befehlszeile ermitteln:

```
cryptsetup luksUUID /dev/sda3
```

und wie folgt benutzen:

```
(mapped-device
  (source (uuid "cb67fc72-0d54-4c88-9d4b-b225f30b0f44"))
  (target "home")
  (type luks-device-mapping))
```

Es ist auch wünschenswert, Swap-Speicher zu verschlüsseln, da in den Swap-Speicher sensible Daten ausgelagert werden können. Eine Möglichkeit ist, eine Swap-Datei auf einem mit LUKS-Verschlüsselung zugeordneten Dateisystem zu verwenden. Dann wird die Swap-Datei verschlüsselt, weil das ganze Gerät verschlüsselt wird. Ein Beispiel finden Sie im Abschnitt Abschnitt 12.5 [Swap-Speicher], Seite 271, oder im Abschnitt Abschnitt 3.4 [Disk Partitioning], Seite 29.

Ein RAID-Gerät als Verbund der Partitionen `/dev/sda1` und `/dev/sdb1` kann wie folgt deklariert werden:

```
(mapped-device
  (source (list "/dev/sda1" "/dev/sdb1"))
  (target "/dev/md0")
  (type raid-device-mapping))
```

Das Gerät `/dev/md0` kann als `device` in einer `file-system`-Deklaration dienen (siehe Abschnitt 12.3 [Dateisysteme], Seite 263). Beachten Sie, dass das RAID-Level dabei nicht angegeben werden muss; es wird während der initialen Erstellung und Formatierung des RAID-Geräts festgelegt und später automatisch bestimmt.

Logische Datenträger von LVM namens „alpha“ und „beta“ aus der Datenträgergruppe (Volume Group) „vg0“ können wie folgt deklariert werden:

```
(mapped-device
  (source "vg0")
  (targets (list "vg0-alpha" "vg0-beta"))
  (type lvm-device-mapping))
```

Die Geräte `/dev/mapper/vg0-alpha` und `/dev/mapper/vg0-beta` können dann im `device`-Feld einer `file-system`-Deklaration verwendet werden (siehe Abschnitt 12.3 [Dateisysteme], Seite 263).

12.5 Swap-Speicher

Swap-Speicher, wie man ihn oft nennt, ist ein Bereich auf der Platte, wohin Speicherseiten verdrängt werden können. Als Seitenaustausch (englisch „Paging“) bezeichnet man das Verfahren, wie der für die Speicherverwaltung zuständige Prozess (d.h. der Linux-Kernel oder der „Default Pager“ in Hurd) entscheiden kann, manche Speicherseiten aus dem Arbeitsspeicher (RAM), die einem laufenden Prozess zugewiesen sind, ohne gerade benutzt zu werden, stattdessen auf der Platte zu speichern. Dadurch wird Arbeitsspeicher frei, wodurch mehr wertvoller schneller Speicher verfügbar wird; die Daten darin werden in den Swap-Speicher ausgelagert. Wenn das Programm auf eben diese Speicherseite zuzugreifen versucht, lädt der Speicherverwaltungsprozess die Daten zurück in den Speicher, damit das Programm sie benutzen kann.

Häufig begegnet man der falschen Vorstellung, Swap nütze nur dann etwas, wenn das System wenig Arbeitsspeicher zur Verfügung hat. Doch benutzen Kernels oft allen Arbeitsspeicher als Zwischenspeicher für Plattenzugriffe, um Ein- und Ausgaben zu beschleunigen. Deswegen steht durch den Austausch ungenutzter Teile des Arbeitsspeichers mehr RAM für diese Art von Caching zur Verfügung.

Wenn Sie genauer wissen wollen, wie Arbeitsspeicher aus Sicht eines monolithisch aufgebauten Kernels verwaltet wird, siehe Abschnitt “Memory Concepts” in *Referenzhandbuch der GNU-C-Bibliothek*.

Der Linux-Kernel unterstützt Swap-Partitionen und Swap-Dateien: Erstere reservieren eine ganze Plattenpartition für den Seitenaustausch, wohingegen Zweitere dafür eine Datei aus dem Dateisystem einsetzen (der Dateisystemtreiber muss sie unterstützen). Auf vergleichbaren Systemen haben beide die gleiche Leistung, also sollte man sich für das entscheiden, was es einem leichter macht. Partitionen sind „einfacher“ und brauchen keine

Unterstützung durch das Dateisystem, aber man muss sie schon beim Formatieren der Platte zuweisen (außer man nutzt logische Datenträger). Dateien hingegen kann man jederzeit anlegen oder löschen.

Vorsicht, Swap-Speicher wird beim Herunterfahren nicht genullt. Sensible Daten (wie Passwörter) können sich darin befinden, wenn deren Speicherseiten verdrängt wurden. Daher sollten Sie in Betracht ziehen, Ihren Swap-Speicher auf ein verschlüsseltes Gerät zu legen (siehe Abschnitt 12.4 [Zugeordnete Geräte], Seite 269).

`swap-space` [Datentyp]

Objekte dieses Typs repräsentieren Swap-Speicher. Sie weisen folgende Komponenten auf:

`target` Welches Gerät oder welche Datei verwendet werden soll, angegeben entweder über die UUID, über ein `file-system-label`-Objekt mit der Bezeichnung oder über eine Zeichenkette wie in der Definition eines `file-system`-Objekts für ein Dateisystem (siehe Abschnitt 12.3 [Dateisysteme], Seite 263).

`dependencies` (Vorgabe: `'()`)
Eine Liste von `file-system`-Objekten oder `mapped-device`-Objekten, die vorausgesetzt werden, damit der Speicher verfügbar ist. Achtung: Genau wie bei `file-system`-Objekten gilt auch für die Abhängigkeiten im `dependencies`-Feld, dass zum Hochfahren des Systems notwendige Abhängigkeiten, die beim Start der Anwendungsebene („User Space“) eingebunden werden, nicht durch Shepherd verwaltet werden, sondern weggefiltert werden.

`priority` (Vorgabe: `#f`)
Wird nur beim Linux-Kernel unterstützt. Entweder `#f`, damit *keine* Priorität festgelegt wird, oder eine ganze Zahl zwischen 0 und 32767. Der Kernel wird erst den Swap-Speicher mit der höheren Priorität für den Seitenaustausch benutzen und bei gleicher Priorität im Rundlauf wechseln („Round-Robin-Verfahren“). Swap-Speicher ohne festgelegte Priorität wird später als priorisierter verwendet, in der angegebenen Reihenfolge ohne Round Robin.

`discard?` (Vorgabe: `#f`)
Wird nur beim Linux-Kernel unterstützt. Wenn es wahr ist, benachrichtigt der Kernel die Steuereinheit (Controller) der Platte, welche Seiten verworfen wurden, zum Beispiel mit der TRIM-Operation auf SSD-Speicher.

Hier sind einige Beispiele:

```
(swap-space (target (uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")))
```

Die Swap-Partition mit der angegebenen UUID verwenden. Sie können die UUID einer Linux-Swap-Partition erfahren, indem Sie `swaplabel` *`Gerät`* ausführen, wobei *Gerät* der Dateiname unter `/dev` für die Partition ist.

```
(swap-space
  (target (file-system-label "swap"))
```

```
  (dependencies mapped-devices))
```

Die Swap-Partition mit der Bezeichnung `swap` verwenden. Die Bezeichnung können Sie finden, nachdem all die zugeordneten Geräte aus *mapped-devices* geöffnet wurden. Auch hier können Sie mittels `swaplabel`-Befehls die Bezeichnung einer Linux-Swap-Partition einsehen und ändern.

Nun folgt ein anspruchsvolleres Beispiel. Sie sehen auch den entsprechenden Teil zu `file-systems` aus der Deklaration eines `operating-system`.

```
(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)))))
```

Die Datei `/btrfs/swapfile` als Swap-Speicher benutzen, die abhängig ist von dem als `/btrfs` eingebundenen Dateisystem. Sie sehen, wie wir mit Guiles `filter`-Prozedur das Dateisystem auf elegante Weise auswählen!

12.6 Benutzerkonten

Benutzerkonten und Gruppen werden allein durch die `operating-system`-Deklaration des Betriebssystems verwaltet. Sie werden mit den `user-account-` und `user-group`-Formen angegeben:

```
(user-account
  (name "alice")
  (group "users")
  (supplementary-groups '("wheel"   ;zur sudo-Nutzung usw. berechtigen
                          "audio"   ;Soundkarte
                          "video"   ;Videogeräte wie Webcams
                          "cdrom")) ;die gute alte CD-ROM
  (comment "Bobs Schwester"))
```

Hier sehen Sie ein Benutzerkonto, das eine andere Shell und ein geändertes Persönliches Verzeichnis benutzt (die Vorgabe wäre `"/home/bob"`):

```
(user-account
  (name "bob")
  (group "users")
  (comment "Alices Bruder")
```

```
(shell (file-append zsh "/bin/zsh"))
(home-directory "/home/robert"))
```

Beim Hochfahren oder nach Abschluss von `guix system reconfigure` stellt das System sicher, dass nur die in der `operating-system`-Deklaration angegebenen Benutzerkonten und Gruppen existieren, mit genau den angegebenen Eigenschaften. Daher gehen durch direkten Aufruf von Befehlen wie `useradd` erwirkte Erstellungen oder Modifikationen von Konten oder Gruppen verloren, sobald rekonfiguriert oder neugestartet wird. So wird sichergestellt, dass das System genau so funktioniert, wie es deklariert wurde.

`user-account` [Datentyp]

Objekte dieses Typs repräsentieren Benutzerkonten. Darin können folgende Komponenten aufgeführt werden:

`name` Der Name des Benutzerkontos.

`group` Dies ist der Name (als Zeichenkette) oder die Bezeichnung (als Zahl) der Benutzergruppe, zu der dieses Konto gehört.

`supplementary-groups` (Vorgabe: `'()`)
Dies kann optional als Liste von Gruppennamen angegeben werden, zu denen dieses Konto auch gehört.

`uid` (Vorgabe: `#f`)
Dies ist entweder der Benutzeridentifikator dieses Kontos (seine „User ID“) als Zahl oder `#f`. Bei Letzterem wird vom System automatisch eine Zahl gewählt, wenn das Benutzerkonto erstellt wird.

`comment` (Vorgabe: `""`)
Ein Kommentar zu dem Konto, wie etwa der vollständige Name des Kontoinhabers.

Beachten Sie, dass Benutzer den für ihr Benutzerkonto hinterlegten echten Namen beliebig ändern können, *außer* es handelt sich um „System“-Benutzerkonten. Den in `/etc/passwd` gespeicherten Namen können sie mit dem Befehl `chfn` ändern. Wenn sie das tun, hat der gewählte Name Vorrang vor dem vom Systemadministrator auserkorenen Namen. Rekonfigurieren ändert den Namen *nicht*.

`home-directory`
Der Name des Persönlichen Verzeichnisses („Home“-Verzeichnis) für dieses Konto.

`create-home-directory?` (Vorgabe: `#t`)
Zeigt an, ob das Persönliche Verzeichnis für das Konto automatisch erstellt werden soll, falls es noch nicht existiert.

`shell` (Vorgabe: Bash)
Ein G-Ausdruck, der den Dateinamen des Programms angibt, das dem Benutzer als Shell dienen soll (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175). Auf die Programmdatei der Bash-Shell würden Sie zum Beispiel so verweisen:

```
(file-append bash "/bin/bash")
```

... und so auf die Programmdatei von Zsh:

```
(file-append zsh "/bin/zsh")
```

`system?` (Vorgabe: `#f`)
: Dieser boolesche Wert zeigt an, ob das Konto ein „System“-Benutzerkonto ist. Systemkonten werden manchmal anders behandelt, zum Beispiel werden sie auf grafischen Anmeldebildschirmen nicht aufgeführt.

`password` (Vorgabe: `#f`)
: Normalerweise lassen Sie dieses Feld auf `#f` und initialisieren Benutzerpasswörter als `root` mit dem `passwd`-Befehl. Die Benutzer lässt man ihr eigenes Passwort dann mit `passwd` ändern. Mit `passwd` festgelegte Passwörter bleiben natürlich beim Neustarten und beim Rekonfigurieren erhalten.

  Wenn Sie aber *doch* ein anfängliches Passwort für ein Konto voreinstellen möchten, muss dieses Feld hier das verschlüsselte Passwort als Zeichenkette enthalten. Sie können dazu die Prozedur `crypt` benutzen.

```
(user-account
  (name "charlie")
  (group "users")

  ;; Ein mit SHA-512 gehashtes initiales Passwort.
  (password (crypt "InitialPassword!" "$6$abc")))
```

  Anmerkung: Der Hash dieses initialen Passworts wird in einer Datei im `/gnu/store` abgelegt, auf die alle Benutzer Lesezugriff haben, daher ist Vorsicht geboten, wenn Sie diese Methode verwenden.

  Siehe Abschnitt “Passphrase Storage” in *Referenzhandbuch der GNU-C-Bibliothek* für weitere Informationen über Passwortverschlüsselung und Abschnitt “Encryption” in *Referenzhandbuch zu GNU Guile* für Informationen über die Prozedur `crypt` in Guile.

Benutzergruppen-Deklarationen sind noch einfacher aufgebaut:

```
(user-group (name "students"))
```

`user-group` [Datentyp]
: Dieser Typ gibt, nun ja, eine Benutzergruppe an. Es gibt darin nur ein paar Felder:

  `name` Der Name der Gruppe.

  `id` (Vorgabe: `#f`)
  : Der Gruppenbezeichner (eine Zahl). Wird er als `#f` angegeben, wird automatisch eine neue Zahl reserviert, wenn die Gruppe erstellt wird.

  `system?` (Vorgabe: `#f`)
  : Dieser boolesche Wert gibt an, ob es sich um eine „System“-Gruppe handelt. Systemgruppen sind solche mit einer kleinen Zahl als Bezeichner.

`password` (Vorgabe: `#f`)
: Wie, Benutzergruppen können ein Passwort haben? Nun ja, anscheinend schon. Wenn es nicht auf `#f` steht, gibt dieses Feld das Passwort der Gruppe an.

Um Ihnen das Leben zu erleichtern, gibt es eine Variable, worin alle grundlegenden Benutzergruppen aufgeführt sind, die man erwarten könnte:

`%base-groups` [Scheme-Variable]
: Die Liste von Basis-Benutzergruppen, von denen Benutzer und/oder Pakete erwarten könnten, dass sie auf dem System existieren. Dazu gehören Gruppen wie „root“, „wheel“ und „users“, sowie Gruppen, um den Zugriff auf bestimmte Geräte einzuschränken, wie „audio“, „disk“ und „cdrom“.

`%base-user-accounts` [Scheme-Variable]
: Diese Liste enthält Basis-Systembenutzerkonten, von denen Programme erwarten können, dass sie auf einem GNU/Linux-System existieren, wie das Konto „nobody“.

  Beachten Sie, dass das Konto „root“ für den Administratornutzer nicht dazugehört. Es ist ein Sonderfall und wird automatisch erzeugt, egal ob es spezifiziert wurde oder nicht.

12.7 Tastaturbelegung

Um anzugeben, was jede Taste auf Ihrer Tastatur tut, müssen Sie angeben, welche *Tastaturbelegung* das Betriebssystem benutzen soll. Wenn nichts angegeben wird, ist die „US English“-QWERTY-Tastaturbelegung für PC-Tastaturen mit 105 Tasten voreingestellt. Allerdings bevorzugen Deutsch sprechende Nutzer meistens die deutsche QWERTZ-Tastaturbelegung, Französisch sprechende haben lieber die AZERTY-Belegung und so weiter; Hacker wollen vielleicht Dvorak oder Bépo als Tastaturbelegung benutzen oder sogar eigene Anpassungen bei manchen Tasten vornehmen. Dieser Abschnitt erklärt, wie das geht.

Die Informationen über Ihre Tastaturbelegung werden an drei Stellen gebraucht:

- Der *Bootloader* muss auslesen können, welche Tastaturbelegung Sie benutzen möchten (siehe Abschnitt 12.14 [Bootloader-Konfiguration], Seite 613). Das ist praktisch, wenn Sie zum Beispiel die Passphrase Ihrer verschlüsselten Wurzelpartition mit der richtigen Tastaturbelegung eintippen wollen.
- Der *Kernel des Betriebssystems*, Linux, braucht die Information, damit die Konsole richtig eingestellt ist (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258).
- Der *grafische Anzeigeserver*, meistens ist das Xorg, hat auch seine eigene Konfiguration der Tastaturbelegung (siehe Abschnitt 12.9.7 [X Window], Seite 342).

Mit Guix können Sie alle drei Komponenten separat konfigurieren, aber zum Glück können Sie damit auch dieselbe Konfiguration der Tastaturbelegung für alle drei benutzen.

Tastaturbelegungen werden durch Verbundsobjekte repräsentiert, die mit der Prozedur `keyboard-layout` aus dem Modul `(gnu system keyboard)` angelegt werden. Entsprechend der „X-Keyboard“-Erweiterung (XKB) verfügt jede Tastaturbelegung über vier Attribute: einen Namen (oft ist das ein Sprachkürzel wie „fi“ für Finnisch oder „jp“ für Japanisch),

ein optionaler Variantenname, ein optionaler Tastaturmodellname und eine möglicherweise leere Liste zusätzlicher Optionen. In den meisten Fällen interessiert Sie nur der Name der Tastaturbelegung.

`keyboard-layout` *`Name`* [*`Variante`*] [*#:model*] [*#:options '()*] [Scheme-Prozedur]
Liefert eine neue Tastaturbelegung mit dem angegebenen *Name*n in der *Variante*.

Der *Name* muss eine Zeichenkette wie `"fr"` sein und die *Variante* eine Zeichenkette wie `"bepo"` oder `"nodeadkeys"`. Siehe das Paket `xkeyboard-config` für Informationen, welche Optionen gültig sind.

Hier sind ein paar Beispiele:

```
;; Die deutsche QWERTZ-Belegung. Hierbei nehmen wir
;; ein Standard-"pc105"-Tastaturmodell an.
(keyboard-layout "de")

;; Die Bépo-Variante der französischen Belegung.
(keyboard-layout "fr" "bepo")

;; Die katalanische Tastaturbelegung.
(keyboard-layout "es" "cat")

;; Arabische Tastaturbelegung. "Alt-Umschalt" wechselt auf US-Amerikanisch.
(keyboard-layout "ar,us" #:options '("grp:alt_shift_toggle"))

;; Die lateinamerikanisch-spanische Tastaturbelegung. Des Weiteren
;; wird die Feststelltaste (auf Englisch "Caps Lock") als eine
;; weitere Steuerungstaste (auf Englisch "Ctrl") festgelegt und
;; die Menütaste soll als eine "Compose"-Taste herhalten, mit der
;; Buchstaben mit Diakritika geschrieben werden können.
(keyboard-layout "latam"
                 #:options '("ctrl:nocaps" "compose:menu"))

;; Die russische Tastaturbelegung für eine ThinkPad-Tastatur.
(keyboard-layout "ru" #:model "thinkpad")

;; Die Tastaturbelegung "US international", d.h. die US-Belegung
;; mit Tottasten zur Eingabe von Buchstaben mit Diakritika. Hier
;; wird die Belegung für eine Apple-MacBook-Tastatur gewählt.
(keyboard-layout "us" "intl" #:model "macbook78")
```

Im Verzeichnis `share/X11/xkb` des `xkeyboard-config`-Pakets finden Sie eine vollständige Liste der unterstützten Tastaturbelegungen, Varianten und Modelle.

Sagen wir, Sie würden gerne die türkische Tastaturbelegung für Ihr gesamtes System – Bootloader, Konsole und Xorg – verwenden. Dann würde Ihre Systemkonfiguration so aussehen:

```
;; Die türkische Tastaturbelegung für Bootloader, Konsole und Xorg
```

```
;; benutzen.

(operating-system
  ;; ...
  (keyboard-layout (keyboard-layout "tr"))  ;für die Konsole
  (bootloader (bootloader-configuration
               (bootloader grub-efi-bootloader)
               (targets '("/boot/efi"))
               (keyboard-layout keyboard-layout))) ;für GRUB
  (services (cons (set-xorg-configuration
                   (xorg-configuration             ;für Xorg
                     (keyboard-layout keyboard-layout)))
                  %desktop-services)))
```

Im obigen Beispiel beziehen wir uns für GRUB und Xorg einfach auf das `keyboard-layout`-Feld, was wir darüber definiert haben, wir könnten aber auch eine andere Tastaturbelegung angeben. Die Prozedur `set-xorg-configuration` kommuniziert an die grafische Anmeldeverwaltung (d.h. nach Vorgabe an GDM), welche Xorg-Konfiguration verwendet werden soll.

Wir haben uns bisher damit auseinandergesetzt, wie die *Voreinstellung* für die Tastaturbelegung ausgewählt werden kann, die das System annimmt, wenn es startet, aber zur Laufzeit kann sie geändert werden:

- Wenn Sie GNOME benutzen, können Sie in den Einstellungen dazu einen Eintrag „Region und Sprache“ finden, in dem Sie eine oder mehrere Tastaturbelegungen auswählen können.
- Unter Xorg können Sie den Befehl `setxkbmap` (aus dem gleichnamigen Paket) zum Anpassen der momentan aktiven Tastaturbelegung benutzen. Zum Beispiel würden Sie so die Belegung auf US Dvorak wechseln:

  ```
  setxkbmap us dvorak
  ```

- Mit dem Befehl `loadkeys` ändern Sie die für die Linux-Konsole geltende Tastaturbelegung. Allerdings ist zu beachten, dass `loadkeys` *nicht* die Kategorisierung der Tastaturbelegungen von XKB benutzt. Der Befehl, um die französische Bépo-Belegung zu laden, wäre folgender:

  ```
  loadkeys fr-bepo
  ```

12.8 Locales

Eine *Locale* legt die kulturellen Konventionen einer bestimmten Sprache und Region auf der Welt fest (siehe Abschnitt “Locales” in *Referenzhandbuch der GNU-C-Bibliothek*). Jede Locale hat einen Namen, der typischerweise von der Form *`Sprache_Gebiet.Kodierung`* – z.B. benennt `fr_LU.utf8` die Locale für französische Sprache mit den kulturellen Konventionen aus Luxemburg unter Verwendung der UTF-8-Kodierung.

Normalerweise werden Sie eine standardmäßig zu verwendende Locale für die Maschine vorgeben wollen, indem Sie das `locale`-Feld der `operating-system`-Deklaration verwenden (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258).

Die ausgewählte Locale wird automatisch zu den dem System bekannten *Locale-Definitionen* hinzugefügt, falls nötig, und ihre Kodierung wird aus dem Namen hergeleitet –

z.B. wird angenommen, dass `bo_CN.utf8` als Kodierung `UTF-8` verwendet. Zusätzliche Locale-Definitionen können im Feld `locale-definitions` vom `operating-system` festgelegt werden – das ist zum Beispiel dann nützlich, wenn die Kodierung nicht aus dem Locale-Namen hergeleitet werden konnte. Die vorgegebene Menge an Locale-Definitionen enthält manche weit verbreiteten Locales, aber um Platz zu sparen, nicht alle verfügbaren Locales.

Um zum Beispiel die nordfriesische Locale für Deutschland hinzuzufügen, könnte der Wert des Feldes wie folgt aussehen:

```
(cons (locale-definition
        (name "fy_DE.utf8") (source "fy_DE"))
      %default-locale-definitions)
```

Um Platz zu sparen, könnte man auch wollen, dass `locale-definitions` nur die tatsächlich benutzen Locales aufführt, wie etwa:

```
(list (locale-definition
        (name "ja_JP.eucjp") (source "ja_JP")
        (charset "EUC-JP")))
```

Die kompilierten Locale-Definitionen sind unter `/run/current-system/locale/X.Y` verfügbar, wobei `X.Y` die Version von libc bezeichnet. Dies entspricht dem Pfad, an dem eine von Guix ausgelieferte GNU libc standardmäßig nach Locale-Daten sucht. Er kann überschrieben werden durch die Umgebungsvariable `LOCPATH` (siehe [locales-and-locpath], Seite 23).

Die `locale-definition`-Form wird vom Modul `(gnu system locale)` zur Verfügung gestellt. Details folgen unten.

`locale-definition` [Datentyp]

Dies ist der Datentyp einer Locale-Definition.

`name` Der Name der Locale. Siehe Abschnitt “Locale Names” in *Referenzhandbuch der GNU-C-Bibliothek* für mehr Informationen zu Locale-Namen.

`source` Der Name der Quelle der Locale. Typischerweise ist das der Teil *`Sprache_Gebiet`* des Locale-Namens.

`charset` (Vorgabe: `"UTF-8"`)
Der „Zeichensatz“ oder das „Code set“, d.h. die Kodierung dieser Locale, wie die IANA sie definiert (`https://www.iana.org/assignments/character-sets`).

`%default-locale-definitions` [Scheme-Variable]

Eine Liste häufig benutzter UTF-8-Locales, die als Vorgabewert des `locale-definitions`-Feldes in `operating-system`-Deklarationen benutzt wird.

Diese Locale-Definitionen benutzen das *normalisierte Codeset* für den Teil des Namens, der nach dem Punkt steht (siehe Abschnitt “Using gettextized software” in *Referenzhandbuch der GNU-C-Bibliothek*). Zum Beispiel ist `uk_UA.utf8` enthalten, dagegen ist etwa `uk_UA.UTF-8` darin *nicht* enthalten.

12.8.1 Kompatibilität der Locale-Daten

`operating-system`-Deklarationen verfügen über ein `locale-libcs`-Feld, um die GNU libc-Pakete anzugeben, die zum Kompilieren von Locale-Deklarationen verwendet werden sollen (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258). „Was interessiert mich das?“, könnten Sie fragen. Naja, leider ist das binäre Format der Locale-Daten von einer libc-Version auf die nächste manchmal nicht miteinander kompatibel.

Zum Beispiel kann ein an die libc-Version 2.21 gebundenes Programm keine mit libc 2.22 erzeugten Locale-Daten lesen; schlimmer noch, das Programm *terminiert*, statt einfach die inkompatiblen Locale-Daten zu ignorieren[5]. Ähnlich kann ein an libc 2.22 gebundenes Programm die meisten, aber nicht alle, Locale-Daten von libc 2.21 lesen (Daten zu `LC_COLLATE` sind aber zum Beispiel inkompatibel); somit schlagen Aufrufe von `setlocale` vielleicht fehl, aber das Programm läuft weiter.

Das „Problem“ mit Guix ist, dass Nutzer viel Freiheit genießen: Sie können wählen, ob und wann sie die Software in ihren Profilen aktualisieren und benutzen vielleicht eine andere libc-Version als sie der Systemadministrator benutzt hat, um die systemweiten Locale-Daten zu erstellen.

Glücklicherweise können „unprivilegierte“ Nutzer ohne zusätzliche Berechtigungen dann zumindest ihre eigenen Locale-Daten installieren und `GUIX_LOCPATH` entsprechend definieren (siehe [locales-and-locpath], Seite 23).

Trotzdem ist es am besten, wenn die systemweiten Locale-Daten unter `/run/current-system/locale` für alle libc-Versionen erstellt werden, die auf dem System noch benutzt werden, damit alle Programme auf sie zugreifen können – was auf einem Mehrbenutzersystem ganz besonders wichtig ist. Dazu kann der Administrator des Systems mehrere libc-Pakete im `locale-libcs`-Feld vom `operating-system` angeben:

```
(use-package-modules base)

(operating-system
  ;; ...
  (locale-libcs (list glibc-2.21 (canonical-package glibc))))
```

Mit diesem Beispiel ergäbe sich ein System, was Locale-Definitionen sowohl für libc 2.21 als auch die aktuelle Version von libc in `/run/current-system/locale` hat.

12.9 Dienste

Ein wichtiger Bestandteil des Schreibens einer `operating-system`-Deklaration ist das Auflisten der *Systemdienste* und ihrer Konfiguration (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250). Systemdienste sind typischerweise im Hintergrund laufende Daemon-Programme, die beim Hochfahren des Systems gestartet werden, oder andere Aktionen, die zu dieser Zeit durchgeführt werden müssen – wie das Konfigurieren des Netzwerkzugangs.

Guix hat eine weit gefasste Definition, was ein „Dienst“ ist (siehe Abschnitt 12.18.1 [Dienstkompositionen], Seite 635), aber viele Dienste sind solche, die von GNU Shepherd

[5] Versionen 2.23 von GNU libc und neuere werden inkompatible Locale-Daten nur mehr überspringen, was schon einmal eine Verbesserung ist.

verwaltet werden (siehe Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644). Auf einem laufenden System kann der `herd`-Befehl benutzt werden, um verfügbare Dienste aufzulisten, ihren Status anzuzeigen, sie zu starten und zu stoppen oder andere angebotene Operationen durchzuführen (siehe Abschnitt "Jump Start" in *The GNU Shepherd Manual*). Zum Beispiel:

```
# herd status
```

Dieser Befehl, durchgeführt als `root`, listet die momentan definierten Dienste auf. Der Befehl `herd doc` fasst kurz zusammen, was ein gegebener Dienst ist und welche Aktionen mit ihm assoziiert sind:

```
# 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.
```

Die Unterbefehle `start`, `stop` und `restart` haben die Wirkung, die man erwarten würde. Zum Beispiel kann mit folgenden Befehlen der nscd-Dienst angehalten und der Xorg-Anzeigeserver neu gestartet werden:

```
# herd stop nscd
Service nscd has been stopped.
# herd restart xorg-server
Service xorg-server has been stopped.
Service xorg-server has been started.
```

Für einige Dienste wird mit `herd configuration` der Name der Konfigurationsdatei des Dienstes ausgegeben. Das ist sinnvoll, wenn Sie die benutzte Konfigurationsdatei untersuchen möchten:

```
# herd configuration sshd
/gnu/store/...-sshd_config
```

Die folgenden Abschnitte dokumentieren die verfügbaren Dienste, die in einer `operating-system`-Deklaration benutzt werden können, angefangen mit den Diensten im Kern des Systems („core services“)

12.9.1 Basisdienste

Das Modul `(gnu services base)` stellt Definitionen für Basis-Dienste zur Verfügung, von denen man erwartet, dass das System sie anbietet. Im Folgenden sind die von diesem Modul exportierten Dienste aufgeführt.

`%base-services` [Scheme-Variable]

Diese Variable enthält eine Liste von Basis-Diensten, die man auf einem System vorzufinden erwartet (siehe Abschnitt 12.18.2 [Diensttypen und Dienste], Seite 637, für weitere Informationen zu Dienstobjekten): ein Anmeldungsdienst (mingetty) auf jeder Konsole (jedem „tty“), syslogd, den Name Service Cache Daemon (nscd) von libc, die udev-Geräteverwaltung und weitere.

Dies ist der Vorgabewert für das `services`-Feld für die Dienste von `operating-system`-Deklarationen. Normalerweise werden Sie, wenn Sie ein Betriebssystem anpassen, Dienste an die `%base-services`-Liste anhängen, wie hier gezeigt:

```
(append (list (service avahi-service-type)
```

```
                (service openssh-service-type))
          %base-services)
```

`special-files-service-type` [Scheme-Variable]

Dieser Dienst richtet „besondere Dateien“ wie `/bin/sh` ein; eine Instanz des Dienstes ist Teil der `%base-services`.

Der mit `special-files-service-type`-Diensten assoziierte Wert muss eine Liste von Tupeln sein, deren erstes Element eine „besondere Datei“ und deren zweites Element deren Zielpfad ist. Der Vorgabewert ist:

```
`(("/bin/sh" ,(file-append bash "/bin/sh"))
  ("/usr/bin/env" ,(file-append coreutils "/bin/env")))
```

Wenn Sie zum Beispiel auch `/bin/bash` zu Ihrem System hinzufügen möchten, können Sie den Wert ändern auf:

```
`(("/bin/sh" ,(file-append bash "/bin/sh"))
  ("/usr/bin/env" ,(file-append coreutils "/bin/env"))
  ("/bin/bash" ,(file-append bash "/bin/bash")))
```

Da dieser Dienst Teil der `%base-services` ist, können Sie `modify-services` benutzen, um die Liste besonderer Dateien abzuändern (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639). Die leichte Alternative, um eine besondere Datei hinzuzufügen, ist über die Prozedur `extra-special-file` (siehe unten).

`extra-special-file` *Datei Ziel* [Scheme-Prozedur]

Das *Ziel* als „besondere Datei“ *Datei* verwenden.

Beispielsweise können Sie die folgenden Zeilen in das `services`-Feld Ihrer Betriebssystemdeklaration einfügen für eine symbolische Verknüpfung `/usr/bin/env`:

```
(extra-special-file "/usr/bin/env"
                    (file-append coreutils "/bin/env"))
```

`host-name-service` *Name* [Scheme-Prozedur]

Liefert einen Dienst, der den Rechnernamen (den „Host“-Namen des Rechners) als *Name* festlegt.

`console-font-service-type` [Scheme-Variable]

Installiert die angegebenen Schriftarten auf den festgelegten TTYs (auf dem Linux-Kernel werden Schriftarten für jede virtuelle Konsole einzeln festgelegt). Als Wert nimmt dieser Dienst eine Liste von Paaren aus TTY und Schriftart. Als Schriftart kann der Name einer vom `kbd`-Paket zur Verfügung gestellten Schriftart oder ein beliebiges gültiges Argument für `setfont` dienen. Ein Beispiel:

```
`(("tty1" . "LatGrkCyr-8x16")
  ("tty2" . ,(file-append
              font-tamzen
              "/share/kbd/consolefonts/TamzenForPowerline10x20.psf"))
  ("tty3" . ,(file-append
              font-terminus
              "/share/consolefonts/ter-132n"))) ; für HiDPI
```

`login-service` *Konfiguration* [Scheme-Prozedur]
Liefert einen Dienst, der die Benutzeranmeldung möglich macht. Dazu verwendet er die angegebene *Konfiguration*, ein `<login-configuration>`-Objekt, das unter anderem die beim Anmelden angezeigte Mitteilung des Tages („Message of the Day“) festlegt.

`login-configuration` [Datentyp]
Dies ist der Datentyp, der die Anmeldekonfiguration repräsentiert.

`motd` Ein dateiartiges Objekt, das die „Message of the Day“ enthält.

`allow-empty-passwords?` (Vorgabe: `#t`)
Leere Passwörter standardmäßig zulassen, damit sich neue Anwender anmelden können, direkt nachdem das Benutzerkonto „root“ für den Administrator angelegt wurde.

`mingetty-service` *Konfiguration* [Scheme-Prozedur]
Liefert einen Dienst, der mingetty nach den Vorgaben der *Konfiguration* ausführt, einem `<mingetty-configuration>`-Objekt, das unter anderem die Konsole (das „tty“) festlegt, auf der mingetty laufen soll.

`mingetty-configuration` [Datentyp]
Dieser Datentyp repräsentiert die Konfiguration von Mingetty, der vorgegebenen Implementierung zur Anmeldung auf einer virtuellen Konsole.

`tty` Der Name der Konsole, auf der diese Mingetty-Instanz läuft – z.B. `"tty1"`.

`auto-login` (Vorgabe: `#f`)
Steht dieses Feld auf wahr, muss es eine Zeichenkette sein, die den Benutzernamen angibt, als der man vom System automatisch angemeldet wird. Ist es `#f`, so muss zur Anmeldung ein Benutzername und ein Passwort eingegeben werden.

`login-program` (Vorgabe: `#f`)
Dies muss entweder `#f` sein, dann wird das voreingestellte Anmeldeprogramm benutzt (`login` aus dem Shadow-Werkzeugsatz) oder der Name des Anmeldeprogramms als G-Ausdruck.

`login-pause?` (Vorgabe: `#f`)
Ist es auf `#t` gesetzt, sorgt es in Verbindung mit *auto-login* dafür, dass der Benutzer eine Taste drücken muss, ehe eine Login-Shell gestartet wird.

`clear-on-logout?` (Vorgabe: `#t`)
Ist dies auf `#t` gesetzt, wird der Bildschirm nach der Abmeldung *nicht* gelöscht.

`mingetty` (Vorgabe: *mingetty*)
Welches Mingetty-Paket benutzt werden soll.

`agetty-service` *Konfiguration* [Scheme-Prozedur]
Liefert einen Dienst, um agetty entsprechend der *Konfiguration* auszuführen, welche ein `<agetty-configuration>`-Objekt sein muss, das unter anderem festlegt, auf welchem tty es laufen soll.

`agetty-configuration` [Datentyp]

Dies ist der Datentyp, der die Konfiguration von agetty repräsentiert, was Anmeldungen auf einer virtuellen oder seriellen Konsole implementiert. Siehe die Handbuchseite `agetty(8)` für mehr Informationen.

`tty` Der Name der Konsole, auf der diese Instanz von agetty läuft, als Zeichenkette – z.B. `"ttyS0"`. Dieses Argument ist optional, sein Vorgabewert ist eine vernünftige Wahl unter den seriellen Schnittstellen, auf deren Benutzung der Linux-Kernel eingestellt ist.

Hierzu wird, wenn in der Kernel-Befehlszeile ein Wert für eine Option namens `agetty.tty` festgelegt wurde, der Gerätename daraus für agetty extrahiert und benutzt.

Andernfalls wird agetty, falls auf der Kernel-Befehlszeile eine Option `console` mit einem tty vorkommt, den daraus extrahierten Gerätenamen der seriellen Schnittstelle benutzen.

In beiden Fällen wird agetty nichts an den anderen Einstellungen für serielle Geräte verändern (Baud-Rate etc.), in der Hoffnung, dass Linux sie auf die korrekten Werte festgelegt hat.

`baud-rate` (Vorgabe: `#f`)
Eine Zeichenkette, die aus einer kommagetrennten Liste von einer oder mehreren Baud-Raten besteht, absteigend sortiert.

`term` (Vorgabe: `#f`)
Eine Zeichenkette, die den Wert enthält, der für die Umgebungsvariable `TERM` benutzt werden soll.

`eight-bits?` (Vorgabe: `#f`)
Steht dies auf `#t`, wird angenommen, dass das tty 8-Bit-korrekt ist, so dass die Paritätserkennung abgeschaltet wird.

`auto-login` (Vorgabe: `#f`)
Wird hier ein Anmeldename als eine Zeichenkette übergeben, wird der angegebene Nutzer automatisch angemeldet, ohne nach einem Anmeldenamen oder Passwort zu fragen.

`no-reset?` (Vorgabe: `#f`)
Steht dies auf `#t`, werden die Cflags des Terminals (d.h. dessen Steuermodi) nicht zurückgesetzt.

`host` (Vorgabe: `#f`)
Dies akzeptiert eine Zeichenkette mit dem einzutragenden Anmeldungs-Rechnernamen („login_host“), der in die Datei `/var/run/utmpx` geschrieben wird.

`remote?` (Vorgabe: `#f`)
Ist dies auf `#t` gesetzt, wird in Verbindung mit *host* eine Befehlszeilenoption `-r` für einen falschen Rechnernamen („Fakehost“) in der Befehlszeile des mit *login-program* angegebenen Anmeldeprogramms übergeben.

`flow-control?` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, wird Hardware-Flusssteuerung (RTS/CTS) aktiviert.

`no-issue?` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, wird der Inhalt der Datei `/etc/issue` *nicht* angezeigt, bevor die Anmeldeaufforderung zu sehen ist.

`init-string` (Vorgabe: `#f`)
: Dies akzeptiert eine Zeichenkette, die zum tty oder zum Modem zuerst vor allem anderen gesendet wird. Es kann benutzt werden, um ein Modem zu initialisieren.

`no-clear?` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, wird agetty den Bildschirm *nicht* löschen, bevor es die Anmeldeaufforderung anzeigt.

`login-program` (Vorgabe: (file-append shadow "/bin/login"))
: Hier muss entweder ein G-Ausdruck mit dem Namen eines Anmeldeprogramms übergeben werden, oder dieses Feld wird nicht gesetzt, so dass als Vorgabewert das Programm `login` aus dem Shadow-Werkzeugsatz verwendet wird.

`local-line` (Vorgabe: `#f`)
: Steuert den Leitungsschalter CLOCAL. Hierfür wird eines von drei Symbolen als Argument akzeptiert, `'auto`, `'always` oder `'never`. Für `#f` wählt agetty als Vorgabewert `'auto`.

`extract-baud?` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, so wird agetty angewiesen, die Baud-Rate aus den Statusmeldungen mancher Arten von Modem abzulesen.

`skip-login?` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, wird der Benutzer nicht aufgefordert, einen Anmeldenamen einzugeben. Dies kann zusammen mit dem *login-program*-Feld benutzt werden, um nicht standardkonforme Anmeldesysteme zu benutzen.

`no-newline?` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, wird *kein* Zeilenumbruch ausgegeben, bevor die Datei `/etc/issue` ausgegeben wird.

`login-options` (Vorgabe: `#f`)
: Dieses Feld akzeptiert eine Zeichenkette mit den Befehlszeilenoptionen für das Anmeldeprogramm. Beachten Sie, dass bei einem selbst gewählten *login-program* ein böswilliger Nutzer versuchen könnte, als Anmeldenamen etwas mit eingebetteten Befehlszeilenoptionen anzugeben, die vom Anmeldeprogramm interpretiert werden könnten.

`login-pause` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, wird auf das Drücken einer beliebigen Taste gewartet, bevor die Anmeldeaufforderung angezeigt wird. Hiermit kann in

Verbindung mit *auto-login* weniger Speicher verbraucht werden, indem man Shells erst erzeugt, wenn sie benötigt werden.

`chroot` (Vorgabe: `#f`)
: Wechselt die Wurzel des Dateisystems auf das angegebene Verzeichnis. Dieses Feld akzeptiert einen Verzeichnispfad als Zeichenkette.

`hangup?` (Vorgabe: `#f`)
: Mit dem Linux-Systemaufruf `vhangup` auf dem angegebenen Terminal virtuell auflegen.

`keep-baud?` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, wird versucht, die bestehende Baud-Rate beizubehalten. Die Baud-Raten aus dem Feld *baud-rate* werden benutzt, wenn agetty ein `BREAK`-Zeichen empfängt.

`timeout` (Vorgabe: `#f`)
: Ist dies auf einen ganzzahligen Wert gesetzt, wird terminiert, falls kein Benutzername innerhalb von *timeout* Sekunden eingelesen werden konnte.

`detect-case?` (Vorgabe: `#f`)
: Ist dies auf `#t` gesetzt, wird Unterstützung für die Erkennung von Terminals aktiviert, die nur Großschreibung beherrschen. Mit dieser Einstellung wird, wenn ein Anmeldename nur aus Großbuchstaben besteht, dieser als Anzeichen dafür aufgefasst, dass das Terminal nur Großbuchstaben beherrscht, und einige Umwandlungen von Groß- in Kleinbuchstaben aktiviert. Beachten Sie, dass dabei *keine* Unicode-Zeichen unterstützt werden.

`wait-cr?` (Vorgabe: `#f`)
: Wenn dies auf `#t` gesetzt ist, wird gewartet, bis der Benutzer oder das Modem einen Wagenrücklauf („Carriage Return“) oder einen Zeilenvorschub („Linefeed“) absendet, ehe `/etc/issue` oder eine Anmeldeaufforderung angezeigt wird. Dies wird typischerweise zusammen mit dem Feld *init-string* benutzt.

`no-hints?` (Vorgabe: `#f`)
: Ist es auf `#t` gesetzt, werden *keine* Hinweise zu den Feststelltasten Num-Taste, Umschaltsperre („Caps Lock“) und Rollen-Taste („Scroll Lock“) angezeigt.

`no-hostname?` (Vorgabe: `#f`)
: Das vorgegebene Verhalten ist, den Rechnernamen auszugeben. Ist dieses Feld auf `#t` gesetzt, wird überhaupt kein Rechnername angezeigt.

`long-hostname?` (Vorgabe: `#f`)
: Das vorgegebene Verhalten ist, den Rechnernamen nur bis zu seinem ersten Punkt anzuzeigen. Ist dieses Feld auf `#t` gesetzt, wird der vollständige Rechnername (der „Fully Qualified Hostname“), wie ihn `gethostname` oder `getaddrinfo` liefern, angezeigt.

`erase-characters` (Vorgabe: `#f`)
: Dieses Feld akzeptiert eine Zeichenkette aus Zeichen, die auch als Rücktaste (zum Löschen) interpretiert werden sollen, wenn der Benutzer seinen Anmeldenamen eintippt.

`kill-characters` (Vorgabe: `#f`)
: Dieses Feld akzeptiert eine Zeichenkette aus Zeichen, deren Eingabe als „ignoriere alle vorherigen Zeichen“ interpretiert werden soll (auch „Kill“-Zeichen genannt), wenn der Benutzer seinen Anmeldenamen eintippt.

`chdir` (Vorgabe: `#f`)
: Dieses Feld akzeptiert eine Zeichenkette, die einen Verzeichnispfad angibt, zu dem vor der Anmeldung gewechselt wird.

`delay` (Vorgabe: `#f`)
: Dieses Feld akzeptiert eine ganze Zahl mit der Anzahl Sekunden, die gewartet werden soll, bis ein tty geöffnet und die Anmeldeaufforderung angezeigt wird.

`nice` (Vorgabe: `#f`)
: Dieses Feld akzeptiert eine ganze Zahl mit dem „nice“-Wert, mit dem das Anmeldeprogramm ausgeführt werden soll.

`extra-options` (Vorgabe: `'()`)
: Dieses Feld ist ein „Notausstieg“, mit dem Nutzer beliebige Befehlszeilenoptionen direkt an `agetty` übergeben können. Diese müssen hier als eine Liste von Zeichenketten angegeben werden.

`shepherd-requirement` (Vorgabe: `'()`)
: Mit dieser Option können zusätzliche Shepherd-Anforderungen für die einzelnen `'term-`*-Shepherd-Dienste festgelegt werden (zum Beispiel `'syslogd`).

`kmscon-service-type` *Konfiguration* [Scheme-Prozedur]
: Liefert einen Dienst, um kmscon (`https://www.freedesktop.org/wiki/Software/kmscon`) entsprechend der *Konfiguration* auszuführen. Diese ist ein `<kmscon-configuration>`-Objekt, das unter anderem angibt, auf welchem tty es ausgeführt werden soll.

`kmscon-configuration` [Datentyp]
: Dieser Datentyp repräsentiert die Konfiguration von Kmscon, die das Anmelden auf virtuellen Konsolen ermöglicht.

`virtual-terminal`
: Der Name der Konsole, auf der diese Kmscon läuft – z.B. `"tty1"`.

`login-program` (Vorgabe: `#~(string-append #$shadow "/bin/login")`)
: Ein G-Ausdruck, der den Namen des Anmeldeprogramms angibt. Als Vorgabe wird das Anmeldeprogramm `login` aus dem Shadow-Werkzeugsatz verwendet.

`login-arguments` (Vorgabe: `'("-p")`)
: Eine Liste der Argumente, die an `login` übergeben werden sollen.

`auto-login` (Vorgabe: `#f`)
: Wird hier ein Anmeldename als eine Zeichenkette übergeben, wird der angegebene Nutzer automatisch angemeldet, ohne nach einem Anmeldenamen oder Passwort zu fragen.

`hardware-acceleration?` (Vorgabe: #f)
: Ob Hardware-Beschleunigung verwendet werden soll.

`font-engine` (Vorgabe: `"pango"`)
: Welcher Schriftartentreiber in Kmscon benutzt wird.

`font-size` (Vorgabe: `12`)
: Welche Schriftgröße in Kmscon benutzt wird.

`keyboard-layout` (Vorgabe: `#f`)
: Wenn es auf `#f` gesetzt ist, benutzt Kmscon die voreingestellte Tastaturbelegung, also normalerweise US English („QWERTY“) für eine PC-Tastatur mit 105 Tasten.

  Andernfalls muss hier ein `keyboard-layout`-Objekt stehen, das angibt, welche Tastaturbelegung aktiv sein soll. Siehe Abschnitt 12.7 [Tastaturbelegung], Seite 276, für mehr Informationen, wie die Tastaturbelegung angegeben werden kann.

`kmscon` (Vorgabe: *kmscon*)
: Das Kmscon-Paket, das benutzt werden soll.

`nscd-service` [*Konfiguration*] [*#:glibc glibc*] [*#:name-services '()*] *Liefert einen Dienst, der den Name Service Cache Daemon* (nscd) von libc mit der angegebenen *Konfiguration* ausführt – diese muss ein `<nscd-configuration>`-Objekt sein. Siehe Abschnitt 12.12 [Name Service Switch], Seite 607, für ein Beispiel. [Scheme-Prozedur]

Der Einfachheit halber bietet der Shepherd-Dienst für nscd die folgenden Aktionen an:

`invalidate`
: Dies macht den angegebenen Zwischenspeicher ungültig. Wenn Sie zum Beispiel:

  ```
  herd invalidate nscd hosts
  ```

  ausführen, wird der Zwischenspeicher für die Auflösung von Rechnernamen (von „Host“-Namen) des nscd ungültig.

`statistics`
: Wenn Sie `herd statistics nscd` ausführen, werden Ihnen Informationen angezeigt, welche Ihnen Informationen über den nscd-Zustand und die Zwischenspeicher angezeigt.

`%nscd-default-configuration` [Scheme-Variable]
Dies ist der vorgegebene Wert für die `<nscd-configuration>` (siehe unten), die `nscd-service` benutzt. Die Konfiguration benutzt die Zwischenspeicher, die in `%nscd-default-caches` definiert sind; siehe unten.

`nscd-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des Name Service Caching Daemon (kurz „nscd“).

`name-services` (Vorgabe: `'()`)
: Liste von Paketen, die *Namensdienste* bezeichnen, die für den nscd sichtbar sein müssen, z.B. `(list nss-mdns)`.

`glibc` (Vorgabe: *glibc*)
: Ein Paket-Objekt, das die GNU-C-Bibliothek angibt, woraus der `nscd`-Befehl genommen werden soll.

`log-file` (Vorgabe: `"/var/log/nscd.log"`)
: Name der nscd-Protokolldatei. Hierhin werden Ausgaben zur Fehlersuche geschrieben, falls `debug-level` echt positiv ist.

`debug-level` (Vorgabe: `0`)
: Eine ganze Zahl, die den Detailgrad der Ausgabe zur Fehlersuche angibt. Größere Zahlen bewirken eine ausführlichere Ausgabe.

`caches` (Vorgabe: `%nscd-default-caches`)
: Liste der `<nscd-cache>`-Objekte, die repräsentieren, was alles zwischengespeichert werden soll; siehe unten.

`nscd-cache` [Datentyp]

Ein Datentyp, der eine Zwischenspeicher-Datenbank von nscd mitsamt ihren Parametern definiert.

`Datenbank`
: Dies ist ein Symbol, was den Namen der Datenbank repräsentiert, die zwischengespeichert werden soll. Gültige Werte sind `passwd`, `group`, `hosts` und `services`, womit jeweils die entsprechende NSS-Datenbank bezeichnet wird (siehe Abschnitt “NSS Basics” in *Referenzhandbuch der GNU-C-Bibliothek*).

`positive-time-to-live`
`negative-time-to-live` (Vorgabe: `20`)
: Eine Zahl, die für die Anzahl an Sekunden steht, die ein erfolgreiches (positives) oder erfolgloses (negatives) Nachschlageresultat im Zwischenspeicher verbleibt.

`check-files?` (Vorgabe: `#t`)
: Ob auf Änderungen an den der *database* entsprechenden Dateien reagiert werden soll.

  Wenn *database* zum Beispiel `hosts` ist, wird, wenn dieses Feld gesetzt ist, nscd Änderungen an `/etc/hosts` beobachten und berücksichtigen.

`persistent?` (Vorgabe: `#t`)
: Ob der Zwischenspeicher dauerhaft auf der Platte gespeichert werden soll.

`shared?` (Vorgabe: `#t`)
: Ob der Zwischenspeicher zwischen den Nutzern geteilt werden soll.

`max-database-size` (Vorgabe: 32 MiB)
: Die Maximalgröße des Datenbank-Zwischenspeichers in Bytes.

`%nscd-default-caches` [Scheme-Variable]
: Liste von `<nscd-cache>`-Objekten, die von der vorgegebenen `nscd-configuration` benutzt werden (siehe oben).

  Damit wird dauerhaftes und aggressives Zwischenspeichern beim Nachschlagen von Dienst- und Rechnernamen („Host“-Namen) aktiviert. Letzteres verbessert die Leistungsfähigkeit beim Nachschlagen von Rechnernamen, sorgt für mehr Widerstandsfähigkeit gegenüber unverlässlichen Namens-Servern und bietet außerdem einen besseren Datenschutz – oftmals befindet sich das Ergebnis einer Anfrage nach einem Rechnernamen bereits im lokalen Zwischenspeicher und externe Namens-Server müssen nicht miteinbezogen werden.

`syslog-configuration` [Datentyp]
: Dieser Datentyp repräsentiert die Konfiguration des syslog-Daemons.

  `syslogd` (Vorgabe: `#~(string-append #$inetutils "/libexec/syslogd")`)
  : Welcher Syslog-Daemon benutzt werden soll.

  `config-file` (Vorgabe: `%default-syslog.conf`)
  : Die zu benutzende syslog-Konfigurationsdatei.

`syslog-service` *`Konfiguration`* [Scheme-Prozedur]
: Liefert einen Dienst, der einen syslog-Daemon entsprechend der *Konfiguration* ausführt.

  Siehe Abschnitt “syslogd invocation” in *GNU Inetutils* für weitere Informationen über die Syntax der Konfiguration.

`guix-service-type` [Scheme-Variable]
: Dies ist der Typ für den Dienst, der den Erstellungs-Daemon `guix-daemon` ausführt (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18). Als Wert muss ein `guix-configuration`-Verbundsobjekt verwendet werden, wie unten beschrieben.

`guix-configuration` [Datentyp]
: Dieser Datentyp repräsentiert die Konfiguration des Erstellungs-Daemons von Guix. Siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18, für weitere Informationen.

  `guix` (Vorgabe: *guix*)
  : Das zu verwendende Guix-Paket.

  `build-group` (Vorgabe: `"guixbuild"`)
  : Der Name der Gruppe, zu der die Erstellungs-Benutzerkonten gehören.

  `build-accounts` (Vorgabe: `10`)
  : Die Anzahl zu erzeugender Erstellungs-Benutzerkonten.

  `authorize-key?` (Vorgabe: `#t`)
  : Ob die unter `authorized-keys` aufgelisteten Substitutschlüssel autorisiert werden sollen – vorgegeben ist, den von `ci.guix.gnu.org` und `bordeaux.guix.gnu.org` zu autorisieren (siehe Abschnitt 6.3 [Substitute], Seite 56).

Wenn `authorize-key?` wahr ist, kann `/etc/guix/acl` durch einen Aufruf von `guix archive --authorize` *nicht* verändert werden. Sie müssen stattdessen die `guix-configuration` wie gewünscht anpassen und das System rekonfigurieren. Dadurch wird sichergestellt, dass die Betriebssystemkonfigurationsdatei eigenständig ist.

> **Anmerkung:** Wenn Sie ein System mit auf wahr gesetztem `authorize-key?` starten oder dahin rekonfigurieren, wird eine Sicherungskopie der bestehenden `/etc/guix/acl` als `/etc/guix/acl.bak` angelegt, wenn festgestellt wurde, dass jemand die Datei von Hand verändert hatte. Das passiert, um die Migration von früheren Versionen zu erleichtern, als eine direkte Modifikation der Datei `/etc/guix/acl`, „in place“, noch möglich war.

`authorized-keys` (Vorgabe: `%default-authorized-guix-keys`)
: Die Liste der Dateien mit autorisierten Schlüsseln, d.h. eine Liste von Zeichenketten als G-Ausdrücke (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74). Der vorgegebene Inhalt ist der Schlüssel von `ci.guix.gnu.org` und `bordeaux.guix.gnu.org` (siehe Abschnitt 6.3 [Substitute], Seite 56). Siehe im Folgenden `substitute-urls` für ein Beispiel, wie Sie sie ändern können.

`use-substitutes?` (Vorgabe: `#t`)
: Ob Substitute benutzt werden sollen.

`substitute-urls` (Vorgabe: `%default-substitute-urls`)
: Die Liste der URLs, auf denen nach Substituten gesucht wird, wenn nicht anders angegeben.

  Wenn Sie zum Beispiel gerne Substitute von `guix.example.org` zusätzlich zu `ci.guix.gnu.org` laden würden, müssen Sie zwei Dinge tun: Erstens `guix.example.org` zu den `substitute-urls` hinzufügen und zweitens dessen Signierschlüssel autorisieren, nachdem Sie ihn hinreichend geprüft haben (siehe Abschnitt 6.3.2 [Substitut-Server autorisieren], Seite 56). Mit der Konfiguration unten wird das erledigt:

```
(guix-configuration
  (substitute-urls
   (append (list "https://guix.example.org")
           %default-substitute-urls))
  (authorized-keys
   (append (list (local-file "./guix.example.org-key.pub"))
           %default-authorized-guix-keys)))
```

  In diesem Beispiel wird angenommen, dass die Datei `./guix.example.org-key.pub` den öffentlichen Schlüssel enthält, mit dem auf `guix.example.org` Substitute signiert werden.

`generate-substitute-key?` (Vorgabe: `#t`)
: Ob ein *Schlüsselpaar für Substitute* als `/etc/guix/signing-key.pub` und `/etc/guix/signing-key.sec` erzeugt werden soll, wenn noch keines da ist.

Mit dem Schlüsselpaar werden Store-Objekte exportiert, z.B. bei `guix publish` (siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233) oder `guix archive` (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74). Es zu erzeugen dauert einige Sekunden, wenn genug Entropie vorrätig ist, und ist auch nur einmal nötig, aber z.B. auf virtuellen Maschinen, die so etwas *nicht* brauchen, schalten Sie es vielleicht lieber aus, wenn die zusätzliche Zeit beim Systemstart ein Problem darstellt.

`max-silent-time` (Vorgabe: `0`)
`timeout` (Vorgabe: `0`)
: Die Anzahl an Sekunden, die jeweils nichts in die Ausgabe geschrieben werden darf bzw. die es insgesamt dauern darf, bis ein Erstellungsprozess abgebrochen wird. Beim Wert null wird nie abgebrochen.

`log-compression` (Vorgabe: `'gzip`)
: Die für Erstellungsprotokolle zu benutzende Kompressionsmethode – entweder `gzip`, `bzip2` oder `none`.

`discover?` (Vorgabe: `#f`)
: Ob im lokalen Netzwerk laufende Substitutserver mit mDNS und DNS-SD ermittelt werden sollen oder nicht.

`extra-options` (Vorgabe: `'()`)
: Eine Liste zusätzlicher Befehlszeilenoptionen zu `guix-daemon`.

`log-file` (Vorgabe: `"/var/log/guix-daemon.log"`)
: Die Datei, in die die Standardausgabe und die Standardfehlerausgabe von `guix-daemon` geschrieben werden.

`http-proxy` (Vorgabe: `#f`)
: Die URL des für das Herunterladen von Ableitungen mit fester Ausgabe und von Substituten zu verwendenden HTTP- und HTTPS-Proxys.

  Sie können den für den Daemon benutzten Proxy auch zur Laufzeit ändern, indem Sie die `set-http-proxy`-Aktion aufrufen, wodurch er neu gestartet wird.

  ```
  herd set-http-proxy guix-daemon http://localhost:8118
  ```

  Um die Proxy-Einstellungen zu löschen, führen Sie dies aus:

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

`tmpdir` (Vorgabe: `#f`)
: Ein Verzeichnispfad, der angibt, wo `guix-daemon` seine Erstellungen durchführt.

`guix-extension` [Datentyp]

Dieser Datentyp repräsentiert die Parameter des Erstellungs-Daemons von Guix, die erweiterbar sind. Ein Objekt dieses Typs kann als Diensterweiterung für Guix verwendet werden. Siehe Abschnitt 12.18.1 [Dienstkompositionen], Seite 635, für weitere Informationen.

`authorized-keys` (Vorgabe: `'()`)
: Eine Liste dateiartiger Objekte, wobei jedes Listenelement einen öffentlichen Schlüssel enthält.

`substitute-urls` (Vorgabe: `'()`)
: Eine Liste von Zeichenketten, die jeweils eine URL mit Substituten enthalten.

`chroot-directories` (Vorgabe: `'()`)
: Eine Liste von dateiartigen Objekten oder Zeichenketten, die Verzeichnisse anzeigen, die im Erstellungs-Daemon zusätzlich nutzbar sind.

`udev-service` [*#:udev* `eudev` *#:rules* `'()`] [Scheme-Prozedur]
Führt *udev* aus, was zur Laufzeit Gerätedateien ins Verzeichnis `/dev` einfügt. udev-Regeln können über die *rules*-Variable als eine Liste von Dateien übergeben werden. Die Prozeduren `udev-rule`, `udev-rules-service` und `file->udev-rule` aus `(gnu services base)` vereinfachen die Erstellung einer solchen Regeldatei.

The `herd rules udev` command, as root, returns the name of the directory containing all the active udev rules.

`udev-rule` [*Dateiname Inhalt*] [Scheme-Prozedur]
Liefert eine udev-Regeldatei mit dem angegebenen *Dateinam*en, in der die vom Literal *Inhalt* definierten Regeln stehen.

Im folgenden Beispiel wird eine Regel für ein USB-Gerät definiert und in der Datei `90-usb-ding.rules` gespeichert. Mit der Regel wird ein Skript ausgeführt, sobald ein USB-Gerät mit der angegebenen Produktkennung erkannt wird.

```
(define %beispiel-udev-rule
  (udev-rule
    "90-usb-ding.rules"
    (string-append "ACTION==\"add\", SUBSYSTEM==\"usb\", "
                   "ATTR{product}==\"Beispiel\", "
                   "RUN+=\"/pfad/zum/skript\"")))
```

`udev-rules-service` [*Name Regeln*] [*#:groups Gruppen*] [Scheme-Prozedur]
Liefert einen Dienst, der den Dienst vom Typ `udev-service-type` um die *Regeln* erweitert und den Dienst vom Typ `account-service-type` um die *Gruppen* in Form von Systembenutzergruppen. Dazu wird ein Diensttyp *name*`-udev-rules` für den einmaligen Gebrauch erzeugt, den der zurückgelieferte Dienst instanziiert.

Hier zeigen wir, wie damit `udev-service-type` um die vorher definierte Regel `%beispiel-udev-rule` erweitert werden kann.

```
(operating-system
 ;; ...
 (services
   (cons (udev-rules-service 'usb-ding %beispiel-udev-rule)
         %desktop-services)))
```

`file->udev-rule` [*Dateiname Datei*] [Scheme-Prozedur]
Liefert eine udev-Datei mit dem angegebenen *Dateinam*en, in der alle in der *Datei*, einem dateiartigen Objekt, definierten Regeln stehen.

Folgendes Beispiel stellt dar, wie wir eine bestehende Regeldatei verwenden können.

```
(use-modules (guix download)     ;für url-fetch
```

```
                (guix packages)     ;für origin
                ...)

        (define %android-udev-rules
          (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"))))))
```

Zusätzlich können Guix-Paketdefinitionen unter den *rules* aufgeführt werden, um die udev-Regeln um diejenigen Definitionen zu ergänzen, die im Unterverzeichnis `lib/udev/rules.d` des jeweiligen Pakets aufgeführt sind. Statt des bisherigen Beispiels zu *file->udev-rule* hätten wir also auch das Paket *android-udev-rules* benutzen können, das in Guix im Modul `(gnu packages android)` vorhanden ist.

Das folgende Beispiel zeit, wie dieses Paket `android-udev-rules` benutzt werden kann, damit das „Android-Tool“ `adb` Geräte erkennen kann, ohne dafür Administratorrechte vorauszusetzen. Man sieht hier auch, wie die Benutzergruppe `adbusers` erstellt werden kann, die existieren muss, damit die im Paket `android-udev-rules` definierten Regeln richtig funktionieren. Um so eine Benutzergruppe zu erzeugen, müssen wir sie sowohl unter den `supplementary-groups` unserer `user-account`-Deklaration aufführen als auch sie im `groups`-Feld der `udev-rules-service`-Prozedur aufführen.

```
(use-modules (gnu packages android)  ;für android-udev-rules
             (gnu system shadow)     ;für user-group
             ...)

(operating-system
  ;; ...
  (users (cons (user-account
                ;; ...
                (supplementary-groups
                 '("adbusers"   ;für adb
                   "wheel" "netdev" "audio" "video")))))
  ;; ...
  (services
    (cons (udev-rules-service 'android android-udev-rules
                              #:groups '("adbusers"))
          %desktop-services)))
```

`urandom-seed-service-type` [Scheme-Variable]

Etwas Entropie in der Datei `%random-seed-file` aufsparen, die als Startwert (als sogenannter „Seed“) für `/dev/urandom` dienen kann, nachdem das System neu gest-

artet wurde. Es wird auch versucht, `/dev/urandom` beim Hochfahren mit Werten aus `/dev/hwrng` zu starten, falls `/dev/hwrng` existiert und lesbar ist.

`%random-seed-file` [Scheme-Variable]
Der Name der Datei, in der einige zufällige Bytes vom *urandom-seed-service* abgespeichert werden, um sie nach einem Neustart von dort als Startwert für `/dev/urandom` auslesen zu können. Als Vorgabe wird `/var/lib/random-seed` verwendet.

`gpm-service-type` [Scheme-Variable]
Dieser Typ wird für den Dienst verwendet, der GPM ausführt, den *General-Purpose Mouse Daemon*, welcher zur Linux-Konsole Mausunterstützung hinzufügt. GPM ermöglicht es seinen Benutzern, auch in der Konsole die Maus zu benutzen und damit etwa Text auszuwählen, zu kopieren und einzufügen.

Der Wert für Dienste dieses Typs muss eine `gpm-configuration` sein (siehe unten). Dieser Dienst gehört *nicht* zu den `%base-services`.

`gpm-configuration` [Datentyp]
Repräsentiert die Konfiguration von GPM.

`options` (Vorgabe: `%default-gpm-options`)
Befehlszeilenoptionen, die an `gpm` übergeben werden. Die vorgegebenen Optionen weisen `gpm` an, auf Maus-Ereignisse auf der Datei `/dev/input/mice` zu lauschen. Siehe Abschnitt "Command Line" in *gpm manual* für weitere Informationen.

`gpm` (Vorgabe: `gpm`)
Das GPM-Paket, was benutzt werden soll.

`guix-publish-service-type` [Scheme-Variable]
Dies ist der Diensttyp für `guix publish` (siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233). Sein Wert muss ein `guix-publish-configuration`-Objekt sein, wie im Folgenden beschrieben.

Hierbei wird angenommen, dass `/etc/guix` bereits ein mit `guix archive --generate-key` erzeugtes Schlüsselpaar zum Signieren enthält (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74). Falls nicht, wird der Dienst beim Starten fehlschlagen.

`guix-publish-configuration` [Datentyp]
Der Datentyp, der die Konfiguration des „`guix publish`"-Dienstes repräsentiert.

`guix` (Vorgabe: `guix`)
Das zu verwendende Guix-Paket.

`port` (Vorgabe: `80`)
Der TCP-Port, auf dem auf Verbindungen gelauscht werden soll.

`host` (Vorgabe: `"localhost"`)
Unter welcher Rechneradresse (welchem „Host", also welcher Netzwerkschnittstelle) auf Verbindungen gelauscht wird. Benutzen Sie `"0.0.0.0"`, wenn auf allen verfügbaren Netzwerkschnittstellen gelauscht werden soll.

`advertise?` (Vorgabe: `#f`)
: Steht dies auf wahr, wird anderen Rechnern im lokalen Netzwerk über das Protokoll DNS-SD unter Verwendung von Avahi mitgeteilt, dass dieser Dienst zur Verfügung steht.

  Dadurch können in der Nähe befindliche Guix-Maschinen mit eingeschalteter Ermittlung (siehe oben die `guix-configuration`) diese Instanz von `guix publish` entdecken und Substitute darüber beziehen.

`compression` (Vorgabe: `'(("gzip" 3) ("zstd" 3))`)
: Dies ist eine Liste von Tupeln aus Kompressionsmethode und -stufe, die zur Kompression von Substituten benutzt werden. Um zum Beispiel alle Substitute mit *beiden*, sowohl lzip auf Stufe 7 und gzip auf Stufe 9, zu komprimieren, schreiben Sie:

  ```
  '(("lzip" 7) ("gzip" 9))
  ```

  Auf Stufe 9 ist das Kompressionsverhältnis am besten, auf Kosten von hoher Prozessorauslastung, während auf Stufe 1 eine schnelle Kompression erreicht wird. Siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233, für weitere Informationen zu den verfügbaren Kompressionsmethoden und ihren jeweiligen Vor- und Nachteilen.

  Wird eine leere Liste angegeben, wird Kompression abgeschaltet.

`nar-path` (Vorgabe: `"nar"`)
: Der URL-Pfad, unter dem „Nars“ zum Herunterladen angeboten werden. Siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233, für Details.

`cache` (Vorgabe: `#f`)
: Wenn dies `#f` ist, werden Archive nicht zwischengespeichert, sondern erst bei einer Anfrage erzeugt. Andernfalls sollte dies der Name eines Verzeichnisses sein – z.B. `"/var/cache/guix/publish"` –, in das `guix publish` fertige Archive und Metadaten zwischenspeichern soll. Siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233, für weitere Informationen über die jeweiligen Vor- und Nachteile.

`workers` (Vorgabe: `#f`)
: Ist dies eine ganze Zahl, gibt es die Anzahl der Worker-Threads an, die zum Zwischenspeichern benutzt werden; ist es `#f`, werden so viele benutzt, wie es Prozessoren gibt. Siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233, für mehr Informationen.

`cache-bypass-threshold` (Vorgabe: 10 MiB)
: Wenn `cache` wahr ist, ist dies die Maximalgröße in Bytes, die ein Store-Objekt haben darf, damit `guix publish` den Zwischenspeicher umgehen darf, falls eine Suche darin mit negativem Ergebnis ausfällt („Cache Miss“). Siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233, für weitere Informationen.

`ttl` (Vorgabe: `#f`)
: Wenn dies eine ganze Zahl ist, bezeichnet sie die *Time-to-live* als die Anzahl der Sekunden, die heruntergeladene veröffentlichte Archive zwi-

schengespeichert werden dürfen. Siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233, für mehr Informationen.

`negative-ttl` (Vorgabe: `#f`)
: Wenn dies eine ganze Zahl ist, bezeichnet sie die *Time-to-live* für erfolglose (negative) Suchen, als Anzahl der Sekunden. Siehe Abschnitt 10.11 [Aufruf von guix publish], Seite 233, für mehr Informationen.

`rngd-service` [*#:rng-tools* `rng-tools`] [*#:device* `"/dev/hwrng"`] *Liefert einen Dienst, der das* [Scheme-Prozedur]
`rngd`-Programm aus den *rng-tools* benutzt, um das mit *device* bezeichnete Gerät zum Entropie-Pool des Kernels hinzuzufügen. Dieser Dienst wird fehlschlagen, falls das mit *device* bezeichnete Gerät nicht existiert.

`pam-limits-service` [*#:limits* `'()`] [Scheme-Prozedur]
Liefert einen Dienst, der eine Konfigurationsdatei für das `pam_limits`-Modul (`http://linux-pam.org/Linux-PAM-html/sag-pam_limits.html`) installiert. Diese Prozedur nimmt optional eine Liste von `pam-limits-entry`-Werten entgegen, die benutzt werden können, um `ulimit`-Limits und `nice`-Prioritäten für Benutzersitzungen festzulegen.

Die folgenden Limit-Definitionen setzen zwei harte und weiche Limits für alle Anmeldesitzungen für Benutzer in der `realtime`-Gruppe.

```
(pam-limits-service
 (list
  (pam-limits-entry "@realtime" 'both 'rtprio 99)
  (pam-limits-entry "@realtime" 'both 'memlock 'unlimited)))
```

Der erste Eintrag erhöht die maximale Echtzeit-Priorität für unprivilegierte Prozesse ohne zusätzliche Berechtigungen; der zweite Eintrag hebt jegliche Einschränkungen des maximalen Adressbereichs auf, der im Speicher reserviert werden darf. Diese Einstellungen werden in dieser Form oft für Echtzeit-Audio-Systeme verwendet.

Ein weiteres nützliches Beispiel stellt das Erhöhen der Begrenzung dar, wie viele geöffnete Dateideskriptoren auf einmal benutzt werden können:

```
(pam-limits-service
 (list
  (pam-limits-entry "*" 'both 'nofile 100000)))
```

Im Beispiel oben steht das Sternchen dafür, dass die Beschränkung für alle Benutzer gelten soll. Es ist wichtig, dass Sie darauf achten, dass der Wert *nicht* größer als der Höchstwert des Systems ist, der in der Datei `/proc/sys/fs/file-max` zu finden ist, denn sonst könnten sich Benutzer *nicht* mehr anmelden. Weitere Informationen über Schranken im Pluggable Authentication Module (PAM) bekommen Sie, wenn Sie die Handbuchseite im `linux-pam`-Paket lesen.

`greetd-service-type` [Scheme-Variable]
`greetd` (`https://git.sr.ht/~kennylevinsen/greetd`) ist ein minimaler und flexibler Daemon zur Anmeldeverwaltung, der *nicht* voraussetzt, dass zu startende Programme außergewöhnliche Anforderungen erfüllen müssen.

Wenn Sie etwas von der Shell in einer virtuellen Konsole aufrufen können, dann kann greetd das auch aufrufen. Wenn dem Programm ein einfaches JSON-basiertes Protokoll zur Interprozesskommunikation beigebracht werden kann, kann es für die Anmeldung verwendet werden als „Greeter“.

`greetd-service-type` steuert die Infrastruktur bei, um Nutzer anzumelden. Dazu gehört:

- `greetd`-PAM-Dienst
- Eine besondere Variation von `pam-mount`, mit der das `XDG_RUNTIME_DIR`-Verzeichnis eingebunden wird

Hier ist ein Beispiel, wie Sie von `mingetty-service-type` auf `greetd-service-type` umsteigen können und wie verschiedene Terminals eingerichtet werden könnten:

```
(append
 (modify-services %base-services
   ;; greetd-service-type bringt uns den PAM-Dienst von "greetd"
   (delete login-service-type)
   ;; und er kann mingetty-service-type ersetzen
   (delete mingetty-service-type))
 (list
  (service greetd-service-type
           (greetd-configuration
            (terminals
             (list
              ;; wir haben die Wahl, welches Terminal anfangs aktiv sein soll
              (greetd-terminal-configuration (terminal-vt "1") (terminal-switch
              ;; wir bestimmen, ob in der Umgebung XDG_RUNTIME_DIR gesetzt wird
              ;; und können sogar eigene Umgebungsvariable vorgeben
              (greetd-terminal-configuration
               (terminal-vt "2")
               (default-session-command
                 (greetd-agreety-session
                  (extra-env '(("MEINE_VAR" . "1")))
                  (xdg-env? #f))))
              ;; wir können eine andere Shell als wie vorgegeben bash benutzen
              (greetd-terminal-configuration
               (terminal-vt "3")
               (default-session-command
                 (greetd-agreety-session (command (file-append zsh "/bin/zsh"))
              ;; wir können jeden ausführbaren Befehl zum Greeter machen
              (greetd-terminal-configuration
               (terminal-vt "4")
               (default-session-command (program-file "nichts-tun-greeter" #~(e
              (greetd-terminal-configuration (terminal-vt "5"))
              (greetd-terminal-configuration (terminal-vt "6"))))))
  ;; mingetty-service-type kann auch parallel benutzt werden;
  ;; wenn man das will, sollte man (delete login-service-type)
  ;; oben weglassen
```

```
#| (service mingetty-service-type (mingetty-configuration (tty "tty8"))) |#)|
```

`greetd-configuration` [Datentyp]

Das Verbundsobjekt mit der Konfiguration des `greetd-service-type`.

`motd` Ein dateiartiges Objekt, das die „Message of the Day“ enthält.

`allow-empty-passwords?` (Vorgabe: `#t`)
Leere Passwörter standardmäßig zulassen, damit sich neue Anwender anmelden können, direkt nachdem das Benutzerkonto „root“ für den Administrator angelegt wurde.

`terminals` (Vorgabe: `'()`)
Eine Liste von `greetd-terminal-configuration` für jedes Terminal, für das `greetd` gestartet werden soll.

`greeter-supplementary-groups` (Vorgabe: `'()`)
Die Liste der Gruppen, zu denen das Benutzerkonto `greeter` hinzugefügt werden soll. Zum Beispiel:

```
(greeter-supplementary-groups '("seat" "video"))
```

Beachten Sie, diese Aktion schlägt fehl, wenn es die Gruppe `seat` nicht gibt.

`greetd-terminal-configuration` [Datentyp]

Verbundsobjekt zur Konfiguration des greetd-Daemons auf einem der Terminals.

`greetd` (Vorgabe: `greetd`)
Das zu verwendende greetd-Paket.

`config-file-name`
Welchen Namen die Konfigurationsdatei des greetd-Daemons bekommen soll. Im Allgemeinen wird er automatisch aus dem Wert von `terminal-vt` abgeleitet.

`log-file-name`
Welchen Namen die Protokolldatei des greetd-Daemons bekommen soll. Im Allgemeinen wird er automatisch aus dem Wert von `terminal-vt` abgeleitet.

`terminal-vt` (Vorgabe: ‘`"7"`’)
Auf welchem virtuellen Terminal das hier läuft. Wir empfehlen, ein bestimmtes VT zu wählen und Konflikte zu vermeiden.

`terminal-switch` (Vorgabe: `#f`)
Ob dieses Terminal beim Start von `greetd` aktiv gemacht werden soll.

`default-session-user` (Vorgabe: ‘`"greeter"`’)
Mit welchem Benutzerkonto der Greeter ausgeführt werden soll.

`default-session-command` (Vorgabe: `(greetd-agreety-session)`)
Dafür können Sie entweder eine Instanz einer Konfiguration mit `greetd-agreety-session` angeben oder mit `gexp->script` ein dateiartiges Objekt als Greeter benutzen.

`greetd-agreety-session` [Datentyp]

Verbundstyp zur Konfiguration des greetd-Greeters agreety.

`agreety` (Vorgabe: `greetd`)
: Das Paket mit dem Befehl `/bin/agreety`.

`command` (Vorgabe: `(file-append bash "/bin/bash")`)
: Der bei erfolgreicher Anmeldung durch `/bin/agreety` auszuführende Befehl.

`command-args` (Vorgabe: `'("-l")`)
: Die Befehlszeilenargumente, die an den `command`-Befehl übergeben werden.

`extra-env` (Vorgabe: `'()`)
: Zusätzliche Umgebungsvariable, die bei der Anmeldung gesetzt werden sollen.

`xdg-env?` (Vorgabe: `#t`)
: Wenn es auf wahr steht, werden `XDG_RUNTIME_DIR` und `XDG_SESSION_TYPE` gesetzt, bevor `command` ausgeführt wird. Es ist zu bedenken, dass die `extra-env` sofort anschließend gesetzt werden und somit Vorrang haben.

`greetd-wlgreet-session` [Datentyp]

Allgemeiner Verbundstyp zur Konfiguration des greetd-Greeters wlgreet.

`wlgreet` (Vorgabe: `wlgreet`)
: Das Paket mit dem Befehl `/bin/wlgreet`.

`command` (Vorgabe: `(file-append sway "/bin/sway")`)
: Der bei erfolgreicher Anmeldung durch `/bin/wlgreet` auszuführende Befehl.

`command-args` (Vorgabe: `'()`)
: Die Befehlszeilenargumente, die an den `command`-Befehl übergeben werden.

`output-mode` (Vorgabe: `"all"`)
: Was für die Option `outputMode` in die TOML-Konfigurationsdatei eingetragen wird.

`scale` (Vorgabe: `1`)
: Was für die Option `scale` in die TOML-Konfigurationsdatei eingetragen wird.

`background` (Vorgabe: `'(0 0 0 0.9)`)
: Eine RGBA-Liste, die die Hintergrundfarbe der Anmeldeaufforderung angibt.

`headline` (Vorgabe: `'(1 1 1 1)`)
: Eine RGBA-Liste, die die Farbe der Titelzeile in der Benutzeroberfläche angibt.

`prompt` (Vorgabe: `'(1 1 1 1)`)
: Eine RGBA-Liste, die die Farbe von Aufforderungen in der Benutzeroberfläche angibt.

`prompt-error` (Vorgabe: `'(1 1 1 1)`)
: Eine RGBA-Liste, die die Farbe von Fehlern in der Benutzeroberfläche angibt.

`border` (Vorgabe: `'(1 1 1 1)`)
: Eine RGBA-Liste, die die Farbe von Umrandungen in der Benutzeroberfläche angibt.

`extra-env` (Vorgabe: `'()`)
: Zusätzliche Umgebungsvariable, die bei der Anmeldung gesetzt werden sollen.

`greetd-wlgreet-sway-session` [Datentyp]
Verbundstyp zur auf Sway bezogenen Konfiguration des greetd-Greeters wlgreet.

`wlgreet-session` (Vorgabe: `(greetd-wlgreet-session)`)
: Ein Verbundsobjekt vom Typ `greetd-wlgreet-session`, das die allgemeine wlgreet-Konfiguration enthält, zusätzlich zur auf Sway bezogenen `greetd-wlgreet-sway-session`.

`sway` (Vorgabe: `sway`)
: Das Paket mit dem Befehl `/bin/sway`.

`sway-configuration` (Vorgabe: #f)
: Ein dateiartiges Objekt, das eine Sway-Konfigurationsdatei enthält, die dem Pflichtteil der Konfiguration vorangestellt wird.

Hier ist ein Beispiel, wie Sie greetd unter Verwendung von wlgreet und Sway konfigurieren können:

```
(greetd-configuration
 ;; Das greeter-Benutzerkonto benötigt diese Berechtigungen, sonst stürzt
 ;; Sway beim Start ab.
 (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")))))))))
```

12.9.2 Geplante Auftragsausführung

Das Modul `(gnu services mcron)` enthält eine Schnittstelle zu GNU mcron, einem Daemon, der gemäß einem vorher festgelegten Zeitplan Aufträge (sogenannte „Jobs“) ausführt (siehe *GNU mcron*). GNU mcron ist ähnlich zum traditionellen `cron`-Daemon aus Unix; der größte Unterschied ist, dass mcron in Guile Scheme implementiert ist, wodurch einem viel Flexibilität bei der Spezifikation von Aufträgen und ihren Aktionen offen steht.

Das folgende Beispiel definiert ein Betriebssystem, das täglich die Befehle `updatedb` (siehe Abschnitt “Invoking updatedb” in *Finding Files*) und `guix gc` (siehe Abschnitt 6.5

[Aufruf von guix gc], Seite 61) ausführt sowie den Befehl `mkid` im Namen eines „unprivilegierten“ Nutzers ohne besondere Berechtigungen laufen lässt (siehe Abschnitt “mkid invocation” in *ID Database Utilities*). Zum Anlegen von Auftragsdefinitionen benutzt es G-Ausdrücke, die dann an mcron übergeben werden (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

```
(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 ()
           (execl (string-append #$findutils "/bin/updatedb")
                  "updatedb"
                  "--prunepaths=/tmp /var/tmp /gnu/store"))
         "updatedb"))

(define garbage-collector-job
  ;; Jeden Tag 5 Minuten nach Mitternacht Müll sammeln gehen.
  ;; Die Aktion des Auftrags ist ein Shell-Befehl.
  #~(job "5 0 * * *"            ;Vixie-cron-Syntax
         "guix gc -F 1G"))

(define idutils-job
  ;; Die Index-Datenbank des Benutzers "charlie" um 12:15 Uhr und
  ;; 19:15 Uhr aktualisieren. Dies wird aus seinem Persönlichen
  ;; Ordner heraus ausgeführt.
  #~(job '(next-minute-from (next-hour '(12 19)) '(15))
         (string-append #$idutils "/bin/mkid src")
         #:user "charlie"))

(operating-system
  ;; ...

  ;; In den %BASE-SERVICES kommt bereits eine Instanz des
  ;; 'mcron-service-type' vor. Wir erweitern sie um weitere
  ;; Aufträge mit einem 'simple-service'.
  (services (cons (simple-service 'my-cron-jobs
                                  mcron-service-type
                                  (list garbage-collector-job
                                        updatedb-job
                                        idutils-job))
                  %base-services)))
```

Tipp: Wenn Sie die Aktion einer Auftragsspezifikation als eine Prozedur angeben, sollten Sie ausdrücklich einen Namen für den Auftrag im dritten Argument

angeben, wie oben im Beispiel zum `updatedb-job` gezeigt. Andernfalls wird für den Auftrag nur „Lambda function“ in der Ausgabe von `herd schedule mcron` angezeigt, was viel zu wenig Aussagekraft hat!

Wenn Sie einen komplexeren Auftrag mit Scheme-Code auf oberster Ebene festlegen möchten, um zum Beispiel eine `use-modules`-Form einzuführen, können Sie Ihren Code in ein separates Programm verschieben, indem Sie die Prozedur `program-file` aus dem Modul `(guix gexp)` benutzen (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175). Das folgende Beispiel veranschaulicht dies.

```
(define %batterie-alarm-auftrag
  ;; Piepsen, wenn die Akkuladung in Prozent unter %MIN-NIVEAU fällt.
  #~(job
     '(next-minute (range 0 60 1))
     #$(program-file
        "batterie-alarm.scm"
        (with-imported-modules (source-module-closure
                                '((guix build utils)))
          #~(begin
              (use-modules (guix build utils)
                           (ice-9 popen)
                           (ice-9 regex)
                           (ice-9 textual-ports)
                           (srfi srfi-2))
              (define %min-niveau 20)
              (setenv "LC_ALL" "C")     ;Ausgabe auf Englisch
              (and-let* ((input-pipe (open-pipe*
                                      OPEN_READ
                                      #$(file-append acpi "/bin/acpi")))
                         (ausgabe (get-string-all input-pipe))
                         (m (string-match "Discharging, ([0-9]+)%" ausgabe))
                         (niveau (string->number (match:substring m 1)))
                         ((< niveau %min-niveau)))
                (format #t "Warnung: Batterieladung nur noch (~a%)~%" niveau)
                (invoke #$(file-append beep "/bin/beep") "-r5")))))))
```

Siehe Abschnitt “Guile Syntax” in *GNU mcron* für weitere Informationen zu mcron-Auftragsspezifikationen. Nun folgt die Referenz des mcron-Dienstes.

Wenn das System läuft, können Sie mit der Aktion `schedule` des Dienstes visualisieren lassen, welche mcron-Aufträge als Nächstes ausgeführt werden:

```
# herd schedule mcron
```

Das vorangehende Beispiel listet die nächsten fünf Aufgaben auf, die ausgeführt werden, aber Sie können auch angeben, wie viele Aufgaben angezeigt werden sollen:

```
# herd schedule mcron 10
```

`mcron-service-type` [Scheme-Variable]

Dies ist der Diensttyp des `mcron`-Dienstes. Als Wert verwendet er ein `mcron-configuration`-Objekt.

Dieser Diensttyp kann als Ziel einer Diensterweiterung verwendet werden, die ihn mit zusätzlichen Auftragsspezifikationen versorgt (siehe Abschnitt 12.18.1 [Dienstkompositionen], Seite 635). Mit anderen Worten ist es möglich, Dienste zu definieren, die weitere mcron-Aufträge ausführen lassen.

`mcron-configuration` [Datentyp]

Verfügbare `mcron-configuration`-Felder sind:

`mcron` (Vorgabe: `mcron`) (Typ: dateiartig)
: Welches mcron-Paket benutzt werden soll.

`jobs` (Vorgabe: `()`) (Typ: Liste-von-G-Ausdrücken)
: Dies muss eine Liste von G-Ausdrücken sein (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), die jeweils einer mcron-Auftragsspezifikation (der Spezifikation eines „Jobs“) entsprechen (siehe Abschnitt “Syntax” in *GNU mcron*).

`log?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Lässt Protokolle auf die Standardausgabe schreiben.

`log-format` (Vorgabe: `"~1@*~a ~a: ~a~%"`) (Typ: Zeichenkette)
: Eine Formatzeichenkette gemäß `(ice-9 format)` für die Protokollnachrichten. Mit dem Vorgabewert werden Nachrichten in der Form "'*Prozesskennung Name*: *Nachricht*"' geschrieben (siehe Abschnitt “Invoking mcron” in *GNU mcron*). Außerdem schreibt GNU Shepherd vor jeder Nachricht einen Zeitstempel.

12.9.3 Log-Rotation

Protokolldateien wie die in `/var/log` neigen dazu, bis ins Unendliche zu wachsen, deshalb ist es eine gute Idee, sie von Zeit zu Zeit zu *rotieren* – d.h. ihren Inhalt in separaten Dateien zu archivieren, welche optional auch komprimiert werden. Das Modul `(gnu services admin)` stellt eine Schnittstelle zu GNU Rot[t]log bereit, einem Werkzeug, um Protokolldateien („Log“-Dateien) zu rotieren (siehe *GNU Rot[t]log Manual*).

Dieser Dienst ist Teil der `%base-services` und daher standardmäßig mit seinen Vorgabeeinstellungen für übliche Log-Dateien aktiv. Das Beispiel unten zeigt, wie Sie ihn um eine weitere *rotation* erweitern können, wenn dies nötig wird (normalerweise kümmern sich darum schon die Dienste, die die Log-Dateien erzeugen):

```
(use-modules (guix) (gnu))
(use-service-modules admin)

(define my-log-files
  ;; Log-Dateien, die ich rotieren lassen will.
  '("/var/log/irgendein.log" "/var/log/noch-ein.log"))

(operating-system
  ;; ...
  (services (cons (simple-service 'meinen-kram-rotieren
                                  rottlog-service-type
                                  (list (log-rotation
```

```
                                        (frequency 'daily)
                                        (files my-log-files))))
                    %base-services)))
```

`rottlog-service-type` [Scheme-Variable]

Dies ist der Typ des Rottlog-Dienstes, dessen Wert ein `rottlog-configuration`-Objekt ist.

Andere Dienste können diesen Dienst um neue `log-rotation`-Objekte erweitern (siehe unten), wodurch die Auswahl an zu rotierenden Dateien ausgeweitet wird.

Dieser Diensttyp kann mcron-Aufträge definieren (siehe Abschnitt 12.9.2 [Geplante Auftragsausführung], Seite 301), die den rottlog-Dienst ausführen.

`rottlog-configuration` [Datentyp]

Datentyp, der die Konfiguration von rottlog repräsentiert.

`rottlog` (Vorgabe: `rottlog`)
: Das Rottlog-Paket, das verwendet werden soll.

`rc-file` (Vorgabe: `(file-append rottlog "/etc/rc")`)
: Die zu benutzende Rottlog-Konfigurationsdatei (siehe Abschnitt "Mandatory RC Variables" in *GNU Rot[t]log Manual*).

`rotations` (Vorgabe: `%default-rotations`)
: Eine Liste von `log-rotation`-Objekten, wie wir sie weiter unten definieren.

`jobs` Dies ist eine Liste von G-Ausdrücken. Jeder G-Ausdruck darin entspricht einer mcron-Auftragsspezifikation (siehe Abschnitt 12.9.2 [Geplante Auftragsausführung], Seite 301).

`log-rotation` [Datentyp]

Datentyp, der die Rotation einer Gruppe von Protokolldateien repräsentiert.

Um ein Beispiel aus dem Rottlog-Handbuch (siehe Abschnitt "Period Related File Examples" in *GNU Rot[t]log Manual*) aufzugreifen: Eine Log-Rotation kann auf folgende Art definiert werden:

```
(log-rotation
  (frequency 'daily) ;täglich
  (files '("/var/log/apache/*"))
  (options '("storedir apache-archives"
             "rotate 6"
             "notifempty"
             "nocompress")))
```

Die Liste der Felder ist folgendermaßen aufgebaut:

`frequency` (Vorgabe: `'weekly`)
: Die Häufigkeit der Log-Rotation, dargestellt als englischsprachiges Symbol.

`files` Die Liste der Dateien oder Glob-Muster für Dateien, die rotiert werden sollen.

`options` (Vorgabe: `%default-log-rotation-options`)
: Die Liste der Rottlog-Optionen für diese Rotation (siehe Abschnitt "Configuration parameters" in *Handbuch von GNU Rot[t]log*).

`post-rotate` (Vorgabe: `#f`)
: Entweder `#f` oder ein G-Ausdruck, der nach Abschluss der Rotation einmal ausgeführt wird.

`%default-rotations` [Scheme-Variable]
: Gibt wöchentliche Rotationen der `%rotated-files` und von `/var/log/guix-daemon.log` an.

`%rotated-files` [Scheme-Variable]
: Die Liste der von Syslog verwalteten Dateien, die rotiert werden sollen. Vorgegeben ist `'("/var/log/messages" "/var/log/secure" "/var/log/debug" "/var/log/maillog")`.

Manche Protokolldateien müssen einfach nur regelmäßig gelöscht werden, wenn sie alt geworden sind, ohne Sonderfälle und ohne Archivierung. Das trifft auf Erstellungsprotokolle zu, die `guix-daemon` unter `/var/log/guix/drvs` vorhält (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18). Der Dienst `log-cleanup` kann diesen Anwendungsfall übernehmen. Zum Beispiel ist Folgendes Teil der `%base-services` (siehe Abschnitt 12.9.1 [Basisdienste], Seite 281):

```
;; Regelmäßig alte Erstellungsprotokolle löschen.
(service log-cleanup-service-type
         (log-cleanup-configuration
          (directory "/var/log/guix/drvs")))
```

Dadurch sammeln sich Erstellungsprotokolle *nicht* bis in alle Ewigkeit an.

`log-cleanup-service-type` [Scheme-Variable]
: Der Diensttyp des Dienstes, um alte Protokolldateien zu löschen. Sein Wert muss ein `log-cleanup-configuration`-Verbundsobjekt sein, wie im Folgenden beschrieben.

`log-cleanup-configuration` [Datentyp]
: Der Datentyp repräsentiert die Konfiguration, um Protokolldateien zu löschen.

`directory`
: Der Name des Verzeichnisses mit den Protokolldateien.

`expiry` (Vorgabe: `(* 6 30 24 3600)`)
: Nach wie vielen Sekunden eine Datei für die Löschung vorgesehen ist (sechs Monate nach Vorgabe).

`schedule` (Vorgabe: `"30 12 01,08,15,22 * *"`)
: Eine Zeichenkette oder ein G-Ausdruck mit dem Zeitplan für mcron-Aufträge (siehe Abschnitt 12.9.2 [Geplante Auftragsausführung], Seite 301).

Anonip-Dienst

Anonip ist ein Datenschutz-Filter, der IP-Adressen aus den Protokollen von Web-Servern entfernt. Mit diesem Dienst wird eine FIFO erzeugt und jede Zeile, die in diese hinein geschrieben wird, wird mit anonip gefiltert und das gefilterte Protokoll in eine Zieldatei übertragen.

Folgendes Beispiel zeigt, wie die FIFO `/var/run/anonip/https.access.log` eingerichtet wird und die gefilterte Protokolldatei `/var/log/anonip/https.access.log` geschrieben wird.

```
(service anonip-service-type
         (anonip-configuration
           (input  "/var/run/anonip/https.access.log")
           (output "/var/log/anonip/https.access.log")))
```

Richten Sie Ihren Web-Server so ein, dass er seine Protokolle in die FIFO unter `/var/run/anonip/https.access.log` schreibt, und Sie finden die anonymisierte Protokolldatei in `/var/web-logs/https.access.log` vor.

`anonip-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration von anonip. Zu ihm gehören folgende Parameter:

`anonip` (Vorgabe: `anonip`)
: Das anonip-Paket, das benutzt werden soll.

`input`
: Der Dateiname der zu verarbeitenden Eingabeprotokolldatei. Durch den Dienst wird eine FIFO erzeugt, die diesen Namen trägt. Der Web-Server sollte seine Protokolle in diese FIFO hinein schreiben.

`output`
: Der Dateiname der verarbeiteten Protokolldatei.

Die folgenden optionalen Einstellungen können Sie angeben:

`skip-private?`
: Wenn das auf `#true` gesetzt ist, werden Adressen in privaten Adressbereichen *nicht* verborgen.

`column`
: Eine bei 1 beginnende Spaltennummer. Es werden IP-Adressen in der angegebenen Spalte verarbeitet (die Vorgabe ist Spalte 1).

`replacement`
: Durch welche Zeichenkette ersetzt wird, wenn die IP-Adresse *nicht* erkannt wird; zum Beispiel `"0.0.0.0"`.

`ipv4mask`
: Anzahl der Bits, die in IPv4-Adressen verborgen werden.

`ipv6mask`
: Anzahl der Bits, die in IPv6-Adressen verborgen werden.

`increment`
: Die IP-Adresse wird um diese Zahl erhöht. Vorgegeben ist null.

`delimiter`
: Welche Zeichenkette der Trenner im Protokoll ist.

`regex`
: Ein regulärer Ausdruck zur Erkennung von IP-Adressen. Er kann statt `column` benutzt werden.

12.9.4 Netzwerkeinrichtung

Durch das Modul `(gnu services networking)` werden Dienste zum Konfigurieren der Netzwerkschnittstellen und zum Einrichten der Netzwerkverbindung Ihrer Maschine bereitgestellt. Die Dienste decken unterschiedliche Arten ab, wie Sie Ihre Maschine einrichten können: Sie können eine statische Netzwerkkonfiguration einrichten, einen Client für das Dynamic Host Configuration Protocol (DHCP) benutzen oder Daemons wie NetworkManager oder Connman einsetzen, mit denen der gesamte Vorgang automatisch abläuft, automatisch auf Änderungen an der Verbindung reagiert wird und eine abstrahierte Benutzerschnittstelle bereitgestellt wird.

Auf einem Laptop sind NetworkManager und Connman bei weitem die komfortabelsten Optionen, darum enthalten die vorgegebenen Dienste für Desktop-Arbeitsumgebungen NetworkManager (siehe Abschnitt 12.9.9 [Desktop-Dienste], Seite 366). Für einen Server, eine virtuelle Maschine oder einen Container sind eine statische Netzwerkkonfiguration oder ein schlichter DHCP-Client meist angemessener.

Dieser Abschnitt beschreibt die verschiedenen Dienste zur Netzwerkeinrichtung, die Ihnen zur Verfügung stehen, angefangen bei der statischen Netzwerkkonfiguration.

`static-networking-service-type` [Scheme-Variable]

Dies ist der Diensttyp für statisch konfigurierte Netzwerkschnittstellen. Sein Wert muss eine Liste von `static-networking`-Verbundsobjekten sein. Jedes deklariert eine Menge von Adressen, Routen und Links, wie im Folgenden gezeigt.

Hier sehen Sie die einfachst mögliche Konfiguration, die nur über eine einzelne Netzwerkkarte („Network Interface Controller“, NIC) eine Verbindung nur für IPv4 herstellt.

```
;; Statische Netzwerkkonfiguration mit einer Netzwerkkarte, nur IPv4.
(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")))))
```

Obiges Code-Schnipsel kann ins `services`-Feld Ihrer Betriebssystemkonfiguration eingetragen werden (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250), um Ihre Maschine mit 10.0.2.15 als ihre IP-Adresse zu versorgen mit einer 24-Bit-Netzmaske für das lokale Netzwerk – also dass jede 10.0.2.*x*-Adresse im lokalen Netzwerk (LAN) ist. Kommunikation mit Adressen außerhalb des lokalen Netzwerks wird über 10.0.2.2 geleitet. Rechnernamen werden über Anfragen ans Domain Name System (DNS) an 10.0.2.3 aufgelöst.

`static-networking` [Datentyp]

Dieser Datentyp repräsentiert eine statische Netzwerkkonfiguration.

Folgendes Beispiel zeigt, wie Sie die Konfiguration einer Maschine deklarieren, die nur über eine einzelne Netzwerkkarte („Network Interface Controller“, NIC), die als `eno1` verfügbar ist, über eine IPv4-Adresse und eine IPv6-Adresse verbunden ist:

```
;; Netzwerkkonfiguration mit einer Netzwerkkarte, 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")))
```

Wenn Sie mit dem Befehl `ip` aus dem `iproute2`-Paket (`https://wiki.linuxfoundation.org/networking/iproute2`) von Linux-basierten Systemen vertraut sind, sei erwähnt, dass obige Deklaration gleichbedeutend damit ist, wenn Sie dies eingeben:

```
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
```

Führen Sie für mehr Informationen `man 8 ip` aus. Alteingesessene GNU/Linux-Nutzer werden sicherlich wissen, wie sie das mit `ifconfig` und `route` machen, aber wir ersparen es Ihnen.

Für den Datentyp stehen folgende Felder zur Verfügung:

`addresses`
`links` (Vorgabe: `'()`)
`routes` (Vorgabe: `'()`)
: Die Liste der `network-address`-, `network-link`- und `network-route`-Verbundsobjekte für dieses Netzwerk (siehe unten).

`name-servers` (Vorgabe: `'()`)
: Die Liste der IP-Adressen (als Zeichenketten) der DNS-Server. Diese IP-Adressen werden in `/etc/resolv.conf` geschrieben.

`provision` (Vorgabe: `'(networking)`)
: Wenn dies ein wahrer Wert ist, bezeichnet dies die Liste von Symbolen für den Shepherd-Dienst, der dieser Netzwerkkonfiguration entspricht.

`requirement` (Vorgabe: `'()`)
: Die Liste der Shepherd-Dienste, von denen dieser abhängt.

`network-address` [Datentyp]

Dieser Datentyp repräsentiert die IP-Adresse einer Netzwerkschnittstelle.

`device` Der Name der Netzwerkschnittstelle, die für diese Adresse benutzt wird – z.B. `"eno1"`.

`value` Die eigentliche IP-Adresse und Netzwerkmaske in CIDR-Notation (`https://de.wikipedia.org/wiki/Classless_Inter-Domain_Routing`) als Zeichenkette.

Zum Beispiel bezeichnet `"10.0.2.15/24"` die IPv4-Adresse 10.0.2.15 auf einem Subnetzwerk, dessen erste 24 Bit gleich sind – alle 10.0.2.*x*-Adressen befinden sich im selben lokalen Netzwerk

`ipv6?` Ob mit `value` eine IPv6-Adresse angegeben wird. Vorgegeben ist, dies automatisch festzustellen.

`network-route` [Datentyp]

Dieser Datentyp steht für eine Netzwerkroute.

`destination`
Das Ziel der Route (als Zeichenkette) entweder mit einer IP-Adresse und Netzwerkmaske oder `"default"` zum Einstellen der Vorgaberoute.

`source` (Vorgabe: `#f`)
Die Quelle der Route.

`device` (Vorgabe: `#f`)
Welches Gerät für diese Route benutzt wird – z.B. `"eno2"`.

`ipv6?` (Vorgabe: automatisch)
Ob es sich um eine IPv6-Route handelt. Das vorgegebene Verhalten ist, dies automatisch anhand des Eintrags in `destination` oder `gateway` zu bestimmen.

`gateway` (Vorgabe: `#f`)
Die IP-Adresse des Netzwerkzugangs (als Zeichenkette), über die der Netzwerkverkehr geleitet wird.

`network-link` [Datentyp]

Der Datentyp für einen Netzwerk-Link (siehe Abschnitt “Link” in *Guile-Netlink-Handbuch*).

`name` Der Name des Links – z.B. `"v0p0"`.

`type` Eine Zeichenkette, die für den Typ des Links steht – z.B. `'veth`.

`arguments`
Eine Liste der Argumente für diesen Link-Typ.

`%loopback-static-networking` [Scheme-Variable]

Dies ist das `static-networking`-Verbundsobjekt, das für das „Loopback-Gerät“ `lo` steht, mit IP-Adressen 127.0.0.1 und ::1, was den Shepherd-Dienst `loopback` zur Verfügung stellt.

`%qemu-static-networking` [Scheme-Variable]
Dies ist das `static-networking`-Verbundsobjekt, das für eine Netzwerkeinrichtung mit QEMUs als Nutzer ausgeführtem Netzwerkstapel („User-Mode Network Stack“) auf dem Gerät `eth0` steht (siehe Abschnitt “Using the user mode network stack” in *QEMU Documentation*).

`dhcp-client-service-type` [Scheme-Variable]
Dies ist der Diensttyp für den Dienst, der *dhcp* ausführt, einen Client für das „Dynamic Host Configuration Protocol“ (DHCP).

`dhcp-client-configuration` [Datentyp]
Der Datentyp, der die Konfiguration des DHCP-Client-Dienstes repräsentiert.

`package` (Vorgabe: `isc-dhcp`)
Das DHCP-Client-Paket, was benutzt werden soll.

`interfaces` (Vorgabe: `'all`)
Geben Sie entweder `'all` an oder die Liste der Namen der Schnittstellen, auf denen der DHCP-Client lauschen soll – z.B. `'("eno1")`.

Wenn es auf `'all` gesetzt ist, wird der DHCP-Client auf allen verfügbaren Netzwerkschnittstellen außer „loopback“, die aktiviert werden können, lauschen. Andernfalls lauscht der DHCP-Client nur auf den angegebenen Schnittstellen.

`network-manager-service-type` [Scheme-Variable]
Dies ist der Diensttyp für den NetworkManager-Dienst (`https://wiki.gnome.org/Projects/NetworkManager`). Der Wert dieses Diensttyps ist ein `network-manager-configuration`-Verbundsobjekt.

Dieser Dienst gehört zu den `%desktop-services` (siehe Abschnitt 12.9.9 [Desktop-Dienste], Seite 366).

`network-manager-configuration` [Datentyp]
Datentyp, der die Konfiguration von NetworkManager repräsentiert.

`network-manager` (Vorgabe: `network-manager`)
Das zu verwendende NetworkManager-Paket.

`dns` (Vorgabe: `"default"`)
Der Verarbeitungsmodus für DNS-Anfragen. Er hat Einfluss darauf, wie NetworkManager mit der Konfigurationsdatei `resolv.conf` verfährt.

‘`default`’ NetworkManager aktualisiert `resolv.conf`, damit sie die Nameserver enthält, die von zurzeit aktiven Verbindungen benutzt werden.

‘`dnsmasq`’ NetworkManager führt `dnsmasq` als lokal zwischenspeichernden Nameserver aus und aktualisiert `resolv.conf` so, dass es auf den lokalen Nameserver verweist. Falls Sie mit einem VPN verbunden sind, wird dafür eine getrennte DNS-Auflösung verwendet („Conditional Forwarding“).

Mit dieser Einstellung können Sie Ihre Netzwerkverbindung teilen. Wenn Sie sie zum Beispiel mit einem anderen Laptop über ein Ethernet-Kabel teilen möchten, können Sie `nm-connection-editor` öffnen und die Methode der Ethernet-Verbindung für IPv4 und IPv6 auf „Gemeinsam mit anderen Rechnern“ stellen und daraufhin die Verbindung neu herstellen (oder Ihren Rechner neu starten).

Sie können so auch eine Verbindung vom Wirts- zum Gastsystem in virtuellen Maschinen mit QEMU (siehe Abschnitt 3.8 [Guix in einer VM installieren], Seite 37) herstellen, d.h. eine „Host-to-Guest Connection“). Mit einer solchen Wirt-nach-Gast-Verbindung können Sie z.B. von einem Webbrowser auf Ihrem Wirtssystem auf einen Web-Server zugreifen, der auf der VM läuft (siehe Abschnitt 12.9.19 [Web-Dienste], Seite 466). Sie können sich damit auch über SSH mit der virtuellen Maschine verbinden (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314). Um eine Wirt-nach-Gast-Verbindung einzurichten, führen Sie einmal diesen Befehl aus:

```
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
```

Danach geben Sie bei jedem Start Ihrer virtuellen QEMU-Maschine (siehe Abschnitt 12.17 [Guix in einer VM starten], Seite 633) die Befehlszeilenoption `-nic tap,ifname=tap0,script=no,downscript=no` an `qemu-system-...` mit.

‘`none`’ NetworkManager verändert `resolv.conf` nicht.

`vpn-plugins` (Vorgabe: `'()`)
: Dies ist die Liste der verfügbaren Plugins für virtuelle private Netzwerke (VPN). Zum Beispiel kann das Paket `network-manager-openvpn` angegeben werden, womit NetworkManager virtuelle private Netzwerke mit OpenVPN verwalten kann.

`connman-service-type` [Scheme-Variable]

Mit diesem Diensttyp wird Connman (`https://01.org/connman`) ausgeführt, ein Programm zum Verwalten von Netzwerkverbindungen.

Sein Wert muss ein `connman-configuration`-Verbundsobjekt wie im folgenden Beispiel sein:

```
(service connman-service-type
         (connman-configuration
           (disable-vpn? #t)))
```

Weiter unten werden Details der `connman-configuration` erklärt.

`connman-configuration` [Datentyp]
Datentyp, der die Konfiguration von Connman repräsentiert.

`connman` (Vorgabe: *connman*)
Das zu verwendende Connman-Paket.

`disable-vpn?` (Vorgabe: `#f`)
Falls dies auf wahr gesetzt ist, wird Connmans VPN-Plugin deaktiviert.

`wpa-supplicant-service-type` [Scheme-Variable]
Dies ist der Diensttyp, um WPA Supplicant (`https://w1.fi/wpa_supplicant/`) auszuführen. Dabei handelt es sich um einen Authentisierungsdaemon, der notwendig ist, um sich gegenüber verschlüsselten WLAN- oder Ethernet-Netzwerken zu authentisieren.

`wpa-supplicant-configuration` [Datentyp]
Repräsentiert die Konfiguration des WPA-Supplikanten.

Sie hat folgende Parameter:

`wpa-supplicant` (Vorgabe: `wpa-supplicant`)
Das WPA-Supplicant-Paket, was benutzt werden soll.

`requirement` (Vorgabe: `'(user-processes loopback syslogd)`)
Die Liste der Dienste, die vor dem WPA-Supplikanten bereits gestartet sein sollen.

`dbus?` (Vorgabe: `#t`)
Ob auf Anfragen auf D-Bus gelauscht werden soll.

`pid-file` (Vorgabe: `"/var/run/wpa_supplicant.pid"`)
Wo die PID-Datei abgelegt wird.

`interface` (Vorgabe: `#f`)
Wenn dieses Feld gesetzt ist, muss es den Namen einer Netzwerkschnittstelle angeben, die von WPA Supplicant verwaltet werden soll.

`config-file` (Vorgabe: `#f`)
Optionale Konfigurationsdatei.

`extra-options` (Vorgabe: `'()`)
Liste zusätzlicher Befehlszeilenoptionen, die an den Daemon übergeben werden.

Manche Netzwerkgeräte wie Modems brauchen eine besondere Behandlung, worauf die folgenden Dienste abzielen.

`modem-manager-service-type` [Scheme-Variable]
Dies ist der Diensttyp für den ModemManager-Dienst (`https://wiki.gnome.org/Projects/ModemManager`). Der Wert dieses Diensttyps ist ein `modem-manager-configuration`-Verbundsobjekt.

Dieser Dienst gehört zu den `%desktop-services` (siehe Abschnitt 12.9.9 [Desktop-Dienste], Seite 366).

`modem-manager-configuration` [Datentyp]
Repräsentiert die Konfiguration vom ModemManager.

`modem-manager` (Vorgabe: `modem-manager`)
Das ModemManager-Paket, was benutzt werden soll.

`usb-modeswitch-service-type` [Scheme-Variable]
Dies ist der Diensttyp für den USB_ModeSwitch-Dienst (`https://www.draisberghof.de/usb_modeswitch/`). Der Wert dieses Diensttyps ist ein `usb-modeswitch-configuration`-Verbundsobjekt.

Wenn sie eingesteckt werden, geben sich manche USB-Modems (und andere USB-Geräte) zunächst als Nur-Lese-Speichermedien und nicht als Modem aus. Sie müssen erst einem Moduswechsel („Modeswitching“) unterzogen werden, bevor sie benutzt werden können. Der USB_ModeSwitch-Diensttyp installiert udev-Regeln, um bei diesen Geräten automatisch ein Modeswitching durchzuführen, wenn sie eingesteckt werden.

Dieser Dienst gehört zu den `%desktop-services` (siehe Abschnitt 12.9.9 [Desktop-Dienste], Seite 366).

`usb-modeswitch-configuration` [Datentyp]
Der Datentyp, der die Konfiguration von USB_ModeSwitch repräsentiert.

`usb-modeswitch` (Vorgabe: `usb-modeswitch`)
Das USB_ModeSwitch-Paket, das die Programmdateien für das Modeswitching enthält.

`usb-modeswitch-data` (Vorgabe: `usb-modeswitch-data`)
Das Paket, in dem die Gerätedaten und die udev-Regeldatei stehen, die USB_ModeSwitch benutzt.

`config-file` (Vorgabe: `#~(string-append #$usb-modeswitch:dispatcher "/etc/usb_modeswitch.conf")`)
Welche Konfigurationsdatei das USB_ModeSwitch-Aufrufprogramm („Dispatcher“) benutzt. Nach Vorgabe wird die mit USB_ModeSwitch ausgelieferte Konfigurationsdatei benutzt, die neben anderen Voreinstellungen die Protokollierung nach `/var/log` abschaltet. Wenn `#f` festgelegt wird, wird keine Konfigurationsdatei benutzt.

12.9.5 Netzwerkdienste

Das im vorherigen Abschnitt besprochene Modul `(gnu services networking)` stellt auch Dienste für fortgeschrittene Netzwerkeinrichtungen zur Verfügung, etwa um einen DHCP-Dienst für andere anzubieten, Pakete mit iptables oder nftables zu filtern, einen WLAN-Zugangspunkt mit `hostapd` verfügbar zu machen, den „Superdaemon“ `inetd` auszuführen und noch mehr. In diesem Abschnitt werden sie beschrieben.

`dhcpd-service-type` [Scheme-Prozedur]
Dieser Diensttyp definiert einen Dienst, der einen DHCP-Daemon ausführt. Um einen Dienst zu diesem Typ anzugeben, müssen Sie eine `<dhcpd-configuration>` bereitstellen. Zum Beispiel so:

```
(service dhcpd-service-type
```

```
(dhcpd-configuration
 (config-file (local-file "my-dhcpd.conf"))
 (interfaces '("enp0s25"))))
```

`dhcpd-configuration` [Datentyp]

`package` (Vorgabe: `isc-dhcp`)
: Das Paket, das den DHCP-Daemon zur Verfügung stellt. Von diesem Paket wird erwartet, dass es den Daemon unter dem Pfad `sbin/dhcpd` relativ zum Verzeichnis der Paketausgabe bereitstellt. Das vorgegebene Paket ist der DHCP-Server vom ISC (`https://www.isc.org/dhcp/`).

`config-file` (Vorgabe: `#f`)
: Die Konfigurationsdatei, die benutzt werden soll. Sie *muss* angegeben werden und wird an `dhcpd` mittels seiner Befehlszeilenoption `-cf` übergeben. Ein beliebiges „dateiartiges“ Objekt kann dafür angegeben werden (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175). Siehe `man dhcpd.conf` für Details, welcher Syntax die Konfigurationsdatei genügen muss.

`version` (Vorgabe: `"4"`)
: Die DHCP-Version, die benutzt werden soll. Der ISC-DHCP-Server unterstützt die Werte „4“, „6“ und „4o6“. Das Feld entspricht den Befehlszeilenoptionen `-4`, `-6` und `-4o6` von `dhcpd`. Siehe `man dhcpd` für Details.

`run-directory` (Vorgabe: `"/run/dhcpd"`)
: Das zu benutzende Laufzeit-Verzeichnis („run“-Verzeichnis). Wenn der Dienst aktiviert wird, wird dieses Verzeichnis erzeugt, wenn es noch nicht existiert.

`pid-file` (Vorgabe: `"/run/dhcpd/dhcpd.pid"`)
: Die zu benutzende PID-Datei. Dieses Feld entspricht der Befehlszeilenoption `-pf` von `dhcpd`. Siehe `man dhcpd` für Details.

`interfaces` (Vorgabe: `'()`)
: Die Namen der Netzwerkschnittstelle, auf der dhcpd auf Broadcast-Nachrichten lauscht. Wenn diese Liste nicht leer ist, werden ihre Elemente (diese müssen Zeichenketten sein) an den `dhcpd`-Aufruf beim Starten des Daemons angehängt. Es ist unter Umständen *nicht* nötig, hier Schnittstellen ausdrücklich anzugeben; siehe `man dhcpd` für Details.

`hostapd-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den hostapd-Daemon (`https://w1.fi/hostapd/`), mit dem ein WLAN-Zugangspunkt (ein „Access Point“ gemäß IEEE 802.11) und Authentifizierungsserver eingerichtet werden kann. Sein zugewiesener Wert muss eine `hostapd-configuration` sein wie im folgenden Beispiel:

```
;; Mit wlan1 den Zugangspunkt für "Mein Netzwerk" betreiben.
(service hostapd-service-type
         (hostapd-configuration
          (interface "wlan1")
          (ssid "Mein Netzwerk")
          (channel 12)))
```

`hostapd-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des hostapd-Dienstes. Er hat folgende Felder:

`package` (Vorgabe: `hostapd`)
: Das zu benutzende hostapd-Paket.

`interface` (Vorgabe: `"wlan0"`)
: Die Netzwerkschnittstelle, auf der der WLAN-Zugangspunkt betrieben wird.

`ssid`
: Die SSID (*Service Set Identifier*), eine das Netzwerk identifizierende Zeichenkette.

`broadcast-ssid?` (Vorgabe: `#t`)
: Ob diese SSID allgemein sichtbar sein soll.

`channel` (Vorlage: `1`)
: Der zu verwendende WLAN-Kanal.

`driver` (Vorgabe: `"nl80211"`)
: Über welchen Schnittstellentyp der Treiber angesprochen wird. `"nl80211"` wird von allen Linux-mac80211-Treibern benutzt. Schreiben Sie `"none"`, wenn hostapd für einen eigenständigen RADIUS-Server erstellt wird, der keine Draht- oder Drahtlosverbindung steuert.

`extra-settings` (Vorgabe: `""`)
: Weitere Einstellungen, die wie sie sind an die Konfigurationsdatei von hostapd angehängt werden. Siehe `https://w1.fi/cgit/hostap/plain/hostapd/hostapd.conf` für eine Referenz der Konfigurationsdatei.

`simulated-wifi-service-type` [Scheme-Variable]

Dies ist der Diensttyp für einen Dienst, um ein WLAN-Netzwerk zu simulieren. Das kann auf virtuellen Maschinen zu Testzwecken eingesetzt werden. Der Dienst lädt das `mac80211_hwsim`-Modul (`https://www.kernel.org/doc/html/latest/networking/mac80211_hwsim/mac80211_hwsim.html`) in den Linux-Kernel und startet hostapd, um ein Pseudo-WLAN-Netzwerk vorzutäuschen, das nach Vorgabe als `wlan0` sichtbar ist.

Der Wert des Dienstes ist ein `hostapd-configuration`-Verbundsobjekt.

`iptables-service-type` [Scheme-Variable]

Mit diesem Diensttyp wird eine iptables-Konfiguration eingerichtet. iptables ist ein Rahmen für Netzwerkpaketfilter, der vom Linux-Kernel unterstützt wird. Der Dienst unterstützt die Konfiguration von iptables für sowohl IPv4 als auch IPv6. Eine einfache Beispielkonfiguration, die alle eingehenden Verbindungen verweigert, die nicht an den SSH-Port 22 gehen, können Sie hier sehen:

```
(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` [Datentyp]

Repräsentiert die iptables-Konfiguration.

`iptables` (Vorgabe: `iptables`)
: Das zu benutzende iptables-Paket, das `iptables-restore` und `ip6tables-restore` zur Verfügung stellt.

`ipv4-rules` (Vorgabe: `%iptables-accept-all-rules`)
: Die zu benutzenden iptables-Regeln. Diese werden an `iptables-restore` übergeben. Als Regeln kann jedes „dateiartige“ Objekt angegeben werden (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

`ipv6-rules` (Vorgabe: `%iptables-accept-all-rules`)
: Die zu benutzenden ip6tables-Regeln. Diese werden an `ip6tables-restore` übergeben. Als Regeln kann jedes „dateiartige“ Objekt angegeben werden (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

`nftables-service-type` [Scheme-Variable]

Dieser Dienst richtet eine Konfiguration von nftables ein. nftables ist als Projekt ein Teil von Netfilter mit dem Ziel, den bestehenden Aufbau aus iptables, ip6tables, arptables und ebtables zu ersetzen. Es stellt einen neuen Rahmen für Netzwerkpaketfilter bereit sowie ein neues Werkzeug `nft` auf Anwendungsebene und eine Kompatibilitätsschicht für iptables. Dieser Dienst wird zusammen mit `%default-nftables-ruleset` ausgeliefert, einem vorgegebenen Satz von Regeln, der alle eingehenden Verbindungen außer auf dem SSH-Port 22 ablehnt. Um ihn zu benutzen, schreiben Sie einfach:

```
(service nftables-service-type)
```

`nftables-configuration` [Datentyp]

Datentyp, der die nftables-Konfiguration repräsentiert.

`package` (Vorgabe: `nftables`)
: Das nftables-Paket, das `nft` zur Verfügung stellt.

`ruleset` (Vorgabe: `%default-nftables-ruleset`)
: Die zu benutzenden nftables-Regeln. Als Regeln kann jedes „dateiartige“ Objekt angegeben werden (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

`ntp-service-type` [Scheme-Variable]
: Dies ist der Typ des Dienstes, der den `ntpd`-Daemon für das Network Time Protocol (`https://www.ntp.org`), kurz NTP, ausführt. Mit diesem Daemon wird die Systemuhr mit der Uhr auf den angegebenen NTP-Servern synchronisiert.

  Der Wert dieses Dienstes ist ein `ntpd-configuration`-Objekt, wie im Folgenden beschrieben.

`ntp-configuration` [Datentyp]
: Der Datentyp für die Dienstkonfiguration des NTP-Dienstes.

  `servers` (Vorgabe: `%ntp-servers`)
  : Dies ist die Liste der Server (`<ntp-server>`-Verbundsobjekte), mit denen `ntpd` synchronisiert wird. Siehe die Definition des `ntp-server`-Datentyps weiter unten.

  `allow-large-adjustment?` (Vorgabe: `#t`)
  : Hiermit wird festgelegt, ob `ntpd` die Uhr beim ersten Umstellen um mehr als 1.000 Sekunden ändern darf.

  `ntp` (Vorgabe: `ntp`)
  : Das NTP-Paket, was benutzt werden soll.

`%ntp-servers` [Scheme-Variable]
: Liste der Rechnernamen („Host“-Namen), die als vorgegebene NTP-Server benutzt werden. Dabei handelt es sich um die Server des NTP Pool Project (`https://www.ntppool.org/en/`).

`ntp-server` [Datentyp]
: Der Datentyp, der die Konfiguration eines NTP-Servers repräsentiert.

  `type` (Vorgabe: `'server`)
  : Die Art des NTP-Servers als Symbol, entweder `'pool`, `'server`, `'peer`, `'broadcast` oder `'manycastclient`.

  `address` Die Adresse des Servers als Zeichenkette.

  `options` NTPD-Optionen, die für diesen bestimmten Server gelten sollen, angegeben als Liste von Optionsnamen und/oder Tupeln aus je Optionsname und -wert. Im folgenden Beispiel wird ein Server definiert, der die Optionen `iburst` und `prefer` sowie `version` 3 und eine `maxpoll`-Zeit von 16 Sekunden benutzen soll.

  ```
  (ntp-server
   (type 'server)
   (address "ein.ntp.server.org")
   (options `(iburst (version 3) (maxpoll 16) prefer))))
  ```

`openntpd-service-type` [Scheme-Prozedur]

Hiermit wird `ntpd`, der Network-Time-Protocol-Daemon (NTP-Daemon), ausgeführt, in seiner OpenNTPD-Implementierung (`http://www.openntpd.org`). Der Daemon sorgt dafür, dass die Systemuhr mit den Uhren der eingestellten Server synchron bleibt.

```
(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` [Scheme-Variable]

Diese Variable bezeichnet eine Liste von Serveradressen, die in `%ntp-servers` definiert sind.

`openntpd-configuration` [Datentyp]

`openntpd` (Vorgabe: `(file-append openntpd "/sbin/ntpd")`)
: Das openntpd-Programm, das benutzt werden soll.

`listen-on` (Vorgabe: `'("127.0.0.1" "::1")`)
: Eine Liste von lokalen IP-Adressen oder Rechnernamen („Host“-Namen), auf denen der ntpd-Daemon lauschen soll.

`query-from` (Vorgabe: `'()`)
: Eine Liste von lokalen IP-Adressen, die der ntpd-Daemon für ausgehende Anfragen benutzen soll.

`sensor` (Vorgabe: `'()`)
: Hiermit geben Sie eine Liste von Zeitdifferenz-Sensorgeräten an, die ntpd benutzen soll. `ntpd` wird auf jeden Sensor lauschen, der auch tatsächlich existiert, und solche, die nicht existieren, ignorieren. Siehe die Dokumentation beim Anbieter (`https://man.openbsd.org/ntpd.conf`) für weitere Informationen.

`server` (Vorgabe: `'()`)
: Hiermit geben Sie eine Liste von IP-Adressen oder Rechnernamen von NTP-Servern an, mit denen synchronisiert werden soll.

`servers` (Vorgabe: `%openntp-servers`)
: Hiermit geben Sie eine Liste von IP-Adressen oder Rechnernamen von NTP-Pools an, mit denen synchronisiert werden soll.

`constraint-from` (Vorgabe: `'()`)
: `ntpd` kann so eingestellt werden, dass es das Datum aus der „Date“-Kopfzeile bei mit TLS übermittelten Anfragen an HTTPS-Server, denen vertraut wird, ausliest. Diese Zeitinformation wird nicht für Genauigkeit benutzt, sondern um mit authentifizierten Informationen die Aus-

wirkungen eines Man-in-the-Middle-Angriffs auf unauthentifizierte NTP-Kommunikation einzuschränken. Geben Sie hierzu eine Liste von URLs, IP-Adressen oder Rechnernamen („Host“-Namen) von HTTPS-Servern an, um eine solche Beschränkung („Constraint“) einzurichten.

`constraints-from` (Vorgabe: `'()`)
: Wie bei `constraint-from` geben Sie auch hier eine Liste von URLs, IP-Adressen oder Rechnernamen von HTTPS-Servern an, um eine Beschränkung einzurichten. Falls der Rechnername zu mehreren IP-Adressen aufgelöst wird, berechnet `ntpd` den Median von allen als Beschränkung.

`inetd-service-type` [Scheme-Variable]

Dieser Dienst führt den `inetd`-Daemon aus (siehe Abschnitt “inetd invocation” in *GNU Inetutils*). `inetd` lauscht auf Verbindungen mit Internet-Sockets und startet bei Bedarf das entsprechende Server-Programm, sobald eine Verbindung mit einem dieser Sockets hergestellt wird.

Der Wert dieses Dienstes ist ein `inetd-configuration`-Objekt. Im folgenden Beispiel wird der `inetd`-Daemon konfiguriert, um den eingebauten `echo`-Dienst sowie einen SMTP-Dienst anzubieten, wobei letzterer SMTP-Kommunikation über SSH an einen Server `smtp-server` über einen vom `rechnernamen` bezeichneten Zugang („Gateway“) weiterleitet:

```
(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" "/pfad/zum/ssh_schlüssel"
                "-W" "smtp-server:25" "benutzer@rechnername")))))))
```

Siehe unten für mehr Details über `inetd-configuration`.

`inetd-configuration` [Datentyp]

Datentyp, der die Konfiguration von `inetd` repräsentiert.

`program` (Vorgabe: `(file-append inetutils "/libexec/inetd")`)
: Das `inetd`-Programm, das benutzt werden soll.

`entries` (Vorgabe: `'()`)
: Eine Liste von `inetd`-Diensteinträgen. Jeder Eintrag sollte von einem `inetd-entry`-Konstruktor erzeugt werden.

`inetd-entry` [Datentyp]
Datentyp, der einen Eintrag in der `inetd`-Konfiguration repräsentiert. Jeder Eintrag entspricht einem Socket, auf dem `inetd` auf Anfragen lauscht.

`node` (Vorgabe: `#f`)
: Optional sollte hier als Zeichenkette eine kommagetrennte Liste lokaler Adressen angegeben werden, die `inetd` benutzen soll, wenn er stellvertretend für den angegebenen Dienst lauscht. Siehe Abschnitt "Configuration file" in *GNU Inetutils* für eine vollständige Beschreibung aller Optionen.

`name`
: Eine Zeichenkette. Dieser Name muss einem Eintrag in `/etc/services` entsprechen.

`socket-type`
: Entweder `'stream`, `'dgram`, `'raw`, `'rdm` oder `'seqpacket`.

`protocol`
: Eine Zeichenkette, die einem Eintrag in `/etc/protocols` entsprechen muss.

`wait?` (Vorgabe: `#t`)
: Ob `inetd` warten soll, bis der Server beendet ist, bevor es wieder auf neue Anfragen an den Dienst lauscht.

`user`
: Eine Zeichenkette mit dem Benutzernamen (und optional dem Gruppennamen) des Benutzers, als der dieser Server ausgeführt werden soll. Der Gruppenname kann als Suffix angegeben werden, getrennt durch einen Doppelpunkt oder Punkt, d.h. `"benutzer"`, `"benutzer:gruppe"` oder `"benutzer.gruppe"`.

`program` (Vorgabe: `"internal"`)
: Das Serverprogramm, das die Anfragen bedienen soll, oder `"internal"`, wenn `inetd` einen eingebauten Dienst verwenden soll.

`arguments` (Vorgabe: `'()`)
: Eine Liste von Zeichenketten oder dateiartigen Objekten, die dem Serverprogramm als Argumente übergeben werden, angefangen mit dem nullten Argument, d.h. dem Namen selbigen Serverprogramms. Bei in `inetd` eingebauten Diensten muss dieser Eintrag auf `'()` oder `'("internal")` gesetzt sein.

Siehe Abschnitt "Configuration file" in *GNU Inetutils* für eine mehr ins Detail gehende Erörterung jedes Konfigurationsfeldes.

`opendht-service-type` [Scheme-Variable]
Dieser Diensttyp dient dazu, einen OpenDHT-Knoten (`https://opendht.net`), `dhtnode`, zu betreiben. Mit dem Daemon kann ein eigener Proxy-Dienst für die

verteilte Hashtabelle (Distributed Hash Table, DHT) angeboten werden, den man in Jami und anderen Anwendungen angeben kann, damit sie sich darüber verbinden.

> **Wichtig:** Wenn Sie den OpenDHT-Proxy-Server nutzen, sollten die für ihn „sichtbaren“ IP-Adressen der Clients auch anderen Netzwerkteilnehmern gegenüber erreichbar sein. In der Praxis ist es am besten, den Proxy-Server auf einem Rechner mit öffentlich zugänglicher IP-Adresse außerhalb Ihres privaten Netzwerks zu betreiben. Wenn Sie den Proxy-Server zum Beispiel im privaten lokalen Netzwerk mit IPv4 zugänglich machen würden und dann mittels Portweiterleitung für die Außenwelt zugänglich machten, funktioniert das vielleicht bei externen Netzwerkteilnehmern, aber für die Geräte im lokalen Netzwerk des Proxys würde anderen OpenDHT-Knoten nur deren private Adresse mitgeteilt, womit sie keine Verbindung aufbauen können.

Der Wert dieses Dienstes ist ein `opendht-configuration`-Objekt, wie im Folgenden beschrieben.

`opendht-configuration` [Datentyp]

Verfügbare `opendht-configuration`-Felder sind:

`opendht` (Vorgabe: `opendht`) (Typ: dateiartig)
: Zu benutzendes `opendht`-Paket.

`peer-discovery?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob sich der Knoten am Multicast-Mechanismus zum Finden lokaler Netzwerkteilnehmer beteiligen soll.

`enable-logging?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob Protokollnachrichten über syslog geschrieben werden sollen. Weil es so ausführlich ist, ist es nach Vorgabe deaktiviert.

`debug?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob Protokollnachrichten der Fehlersuch-Ausführlichkeitsstufe aktiviert werden sollen. Diese Option wirkt sich *nicht* aus, wenn Protokollierung ganz abgeschaltet ist.

`bootstrap-host` (Vorgabe: `"bootstrap.jami.net:4222"`) (Typ: Vielleicht-Zeichenkette)
: Der Rechnername des Knotens, über den eine erste Verbindung ins OpenDHT-Netzwerk aufgebaut wird. Es kann eine bestimmte Portnummer angegeben werden, indem als Suffix `:PORT` angehängt wird. Vorgegeben ist, den Bootstrap-Knoten von Jami zu benutzen, aber man kann jeden Rechnernamen eines Knotens angeben. Es ist auch möglich, Bootstrapping zu deaktivieren, indem man dies ausdrücklich auf den Wert `%unset-value` festlegt.

`port` (Vorgabe: `4222`) (Typ: Vielleicht-Zahl)
: An welchen UDP-Port sich der OpenDHT-Knoten binden soll. Wird er *nicht* angegeben, wird ein verfügbarer Port automatisch ausgewählt.

`proxy-server-port` (Typ: Vielleicht-Zahl)
: Einen Proxy-Server auf dem angegebenen Port lauschen lassen.

`proxy-server-port-tls` (Typ: Vielleicht-Zahl)
: Einen Proxy-Server auf dem angegebenen Port auf TLS-Verbindungen lauschen lassen.

`tor-service-type` [Scheme-Variable]
: Dies ist der Diensttyp für den Dienst, der den Tor-Daemon (`https://torproject.org`) für anonyme Netzwerkrouten ausführt. Der Dienst benutzt für seine Konfiguration ein `<tor-configuration>`-Verbundsobjekt. Vorgegeben ist, dass der Tor-Daemon als „unprivilegierter“ Nutzer `tor` ausgeführt wird, einem Mitglied der `tor`-Benutzergruppe ohne besondere Berechtigungen.

`tor-configuration` [Datentyp]
: `tor` (Vorgabe: `tor`)
  : Das Paket, das den Tor-Daemon zur Verfügung stellt. Von diesem Paket wird erwartet, dass es den Daemon unter dem Pfad `bin/tor` relativ zum Ausgabeverzeichnis verfügbar macht. Das vorgegebene Paket ist die Implementierung des Tor-Projekts (`https://www.torproject.org`).

  `config-file` (Vorgabe: `(plain-file "empty" "")`)
  : Die Konfigurationsdatei, die benutzt werden soll. Sie wird an eine vorgegebene Konfigurationsdatei angehängt und die sich daraus ergebende Konfigurationsdatei wird dann an `tor` über dessen Befehlszeilenoption `-f` übergeben. Hier kann jedes „dateiartige“ Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) angegeben werden. Siehe `man tor` für Details zur Syntax der Konfigurationsdatei.

  `hidden-services` (Vorgabe: `'()`)
  : Die Liste der zu benutzenden „versteckten Dienste“ als `<hidden-service>`-Verbundsobjekte. Für jeden versteckten Dienst, den Sie in dieser Liste eintragen, werden automatisch entsprechende Einstellungen zur vorgefertigten Konfigurationsdatei hinzugefügt. Sie können `<hidden-service>`-Verbundsobjekte bequem mit der unten beschriebenen Prozedur `tor-hidden-service` erzeugen lassen.

  `socks-socket-type` (Vorgabe: `'tcp`)
  : Welche Art von Socket Tor für seinen SOCKS-Socket in der Voreinstellung benutzen soll. Dafür muss entweder `'tcp` oder `'unix` angegeben werden. Für `'tcp` wird Tor nach Voreinstellung auf dem TCP-Port 9050 auf der loopback-Schnittstelle (d.h. localhost) lauschen. Für `'unix` wird Tor auf dem UNIX-Socket `/var/run/tor/socks-sock` lauschen, auf den Mitglieder der `tor`-Benutzergruppe Schreibberechtigung erhalten.

    Wenn Sie detailliertere Anpassungen am SOCKS-Socket vornehmen wollen, belassen Sie `socks-socket-type` bei seinem vorgegebenen Wert `'tcp` und benutzen Sie `config-file`, um diese Voreinstellung mit Ihrer eigenen `SocksPort`-Option zu überspielen.

  `control-socket?` (Vorgabe: `#f`)
  : Ob ein „Steuerungs-Socket“ bereitgestellt werden soll, über den Tor angesteuert werden kann, um zum Beispiel Onion-Dienste zur Laufzeit zu

instanziieren. Wird hier `#t` angegeben, nimmt Tor Steuerungsbefehle auf dem Unix-Socket `/var/run/tor/control-sock` entgegen, auf den Mitglieder der `tor`-Benutzergruppe Schreibzugriff bekommen.

`tor-hidden-service` *Name Zuordnung* [Scheme-Prozedur]

Hiermit wird ein neuer *versteckter Dienst* von Tor mit diesem *Name*n definiert, der die *Zuordnung* herstellt. Die *Zuordnung* ist eine Liste von Port-/Rechner-Tupeln wie hier:

```
'((22 "127.0.0.1:22")
  (80 "127.0.0.1:8080"))
```

In diesem Beispiel wird Port 22 des versteckten Dienstes an den ihm zugeordneten lokalen Port 22 weitergeleitet und Port 80 wird an den lokalen Port 8080 weitergeleitet.

Dadurch wird ein Verzeichnis `/var/lib/tor/hidden-services/`*Name* erstellt, worin sich in der Datei `hostname` der `.onion`-Rechnername („Host“-Name) des versteckten Dienstes befindet.

Siehe die Dokumentation des Tor-Projekts (`https://www.torproject.org/docs/tor-hidden-service.html.en`) für weitere Informationen.

Das Modul `(gnu services rsync)` bietet die folgenden Dienste an:

Sie könnten einen rsync-Daemon einsetzen wollen, um Dateien verfügbar zu machen, damit jeder (oder nur Sie) bestehende Dateien herunterladen oder neue Dateien hochladen kann.

`rsync-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den rsync-Daemon (`https://rsync.samba.org`), er benutzt ein `rsync-configuration`-Verbundsobjekt wie in diesem Beispiel:

```
;; Zwei Verzeichnisse über rsync exportieren.  Wie vorgegeben
;; lauscht rsync auf allen Netzwerkschnittstellen.
(service rsync-service-type
         (rsync-configuration
           (modules (list (rsync-module
                           (name "musik")
                           (file-name "/srv/zik")
                           (read-only? #f))
                          (rsync-module
                           (name "filme")
                           (file-name "/home/charlie/filme"))))))
```

Siehe unten für Details zur `rsync-configuration`.

`rsync-configuration` [Datentyp]

Datentyp, der die Konfiguration für den `rsync-service` repräsentiert.

`package` (Vorgabe: *rsync*)
: Zu benutzendes `rsync`-Paket.

`address` (Vorgabe: `#f`)
: Auf welcher IP-Adresse `rsync` auf eingehende Verbindungen lauscht. Wird nichts angegeben, wird auf allen verfügbaren Adressen gelauscht.

`port-number` (Vorgabe: `873`)
: Der TCP-Port, auf dem `rsync` auf eingehende Verbindungen lauscht. Wenn die Portnummer kleiner als `1024` ist, muss `rsync` als Administratornutzer `root` und auch mit dieser Benutzergruppe gestartet werden.

`pid-file` (Vorgabe: `"/var/run/rsyncd/rsyncd.pid"`)
: Der Name der Datei, in die `rsync` seine PID schreibt.

`lock-file` (Vorgabe: `"/var/run/rsyncd/rsyncd.lock"`)
: Der Name der Datei, die `rsync` als seine Sperrdatei verwendet.

`log-file` (Vorgabe: `"/var/log/rsyncd.log"`)
: Der Name der Datei, in die `rsync` seine Protokolle schreibt.

`user` (Vorgabe: `"root"`)
: Das Benutzerkonto, dem der `rsync`-Prozess gehören soll.

`group` (Vorgabe: `"root"`)
: Die Benutzergruppe des `rsync`-Prozesses.

`uid` (Vorgabe: `"rsyncd"`)
: Der Benutzername oder der Benutzeridentifikator (d.h. die „User-ID“), mit dem Dateiübertragungen zum und vom Modul stattfinden sollen, wenn der Daemon als Administratornutzer `root` läuft.

`gid` (Vorgabe: `"rsyncd"`)
: Benutzergruppenname oder Gruppenidentifikator („Group-ID“), mit dem auf das Modul zugegriffen wird.

`modules` (Vorgabe: `%default-modules`)
: Liste von „Modulen“ – d.h. Verzeichnissen, die mit rsync exportiert werden. Jedes Element muss ein `rsync-module`-Verbund sein, wie nun beschrieben wird.

`rsync-module` [Datentyp]

Dies ist der Datentyp für rsync-„Module“. Ein Modul ist ein Verzeichnis, das über das rsync-Protokoll exportiert wird. Die verfügbaren Felder sind wie folgt:

`name`
: Der Modulname. Dieser ist der Name, der in URLs benutzt wird. Zum Beispiel, wenn das Modul `musik` heißt, wird die entsprechende URL `rsync://host.example.org/musik` sein.

`file-name`
: Der Name des Verzeichnisses, das exportiert wird.

`comment` (Vorgabe: `""`)
: Kommentar, der mit dem Modul verbunden ist. Clientbenutzerschnittstellen dürfen das anzeigen, wenn sie die Liste der verfügbaren Module bekommen.

`read-only?` (Vorgabe: `#t`)
: Ob der Client Dateien hochladen können soll. Wenn dies falsch ist, wird das Hochladen autorisiert werden, wenn die Berechtigungen dort, wo der Daemon läuft, es erlauben.

`chroot?` (Vorgabe: `#t`)
: Wenn es auf wahr steht, wechselt der rsync-Daemon das Wurzelverzeichnis in das Verzeichnis des Moduls, bevor er Dateiübertragungen mit dem Client unternimmt. Das ist besser für die Sicherheit, aber es geht nur, wenn rsync als Administratornutzer root läuft.

`timeout` (Vorgabe: `300`)
: Wie viele Sekunden ein Prozess untätig bleiben darf, bis eine Verbindung zum Client getrennt wird.

Das Modul `(gnu services syncthing)` bietet die folgenden Dienste an:

Sie könnten einen syncthing-Daemon benutzen wollen, wenn Sie Dateien auf zwei oder mehr Rechnern haben und diese in Echtzeit synchronisieren wollen, geschützt vor neugierigen Blicken.

`syncthing-service-type` [Scheme-Variable]
: Dies ist der Diensttyp für den syncthing-Daemon (`https://syncthing.net/`), er benutzt ein `syncthing-configuration`-Verbundsobjekt wie in diesem Beispiel:

```
(service syncthing-service-type
         (syncthing-configuration (user "alice")))
```

: Siehe unten für Details zur `syncthing-configuration`.

`syncthing-configuration` [Datentyp]
: Datentyp, der die Konfiguration für den `syncthing-service-type` repräsentiert.

`syncthing` (Vorgabe: *syncthing*)
: Zu benutzendes `syncthing`-Paket.

`arguments` (Vorgabe: *'()*)
: Liste der Befehlszeilenoptionen, die an das `syncthing`-Programm übergeben werden.

`logflags` (Vorgabe: *0*)
: Die Summe aus den Protokollierungsoptionen, siehe die Dokumentation von Syncthing zu logflags (`https://docs.syncthing.net/users/syncthing.html#cmdoption-logflags`).

`user` (Vorgabe: *#f*)
: Das Benutzerkonto, mit dem der Syncthing-Dienst ausgeführt wird. Es wird vorausgesetzt, dass der angegebene Benutzer existiert.

`group` (Vorgabe: "*users*")
: Die Gruppe, mit der der Syncthing-Dienst ausgeführt wird. Es wird vorausgesetzt, dass die angegebene Gruppe existiert.

`home` (Vorgabe: *#f*)
: Das gemeinsame Verzeichnis für sowohl Konfiguration als auch Daten. In der Vorgabeeinstellung würde das in `$HOME` gespeicherte Verzeichnis das Konfigurationsverzeichnis des mit `user` festgelegten Syncthing-Nutzers.

Des Weiteren bietet das Modul `(gnu services ssh)` die folgenden Dienste an.

`lsh-service` [*#:host-key "/etc/lsh/host-key"*] [*#:daemonic? #t*] [*#:interfaces '()*] [*#:port-number 22*] [*#:allow-empty-passwords? #f*] [*#:root-login? #f*] [*#:syslog-output? #t*] [#:x11-forwarding? #t] [#:tcp/ip-forwarding? #t] [#:password-authentication? #t] [#:public-key-authentication? #t] [#:initialize? #t] [Scheme-Prozedur]

Das `lshd`-Programm auf dem *lsh*-Paket so ausführen, dass es auf dem Port mit Portnummer *port-number* lauscht. *host-key* muss eine Datei angeben, die den Rechnerschlüssel enthält, die nur für den Administratornutzer lesbar sein darf.

Wenn *daemonic?* wahr ist, entkoppelt sich `lshd` vom Terminal, auf dem er läuft, und schickt seine Protokolle an syslogd, außer *syslog-output?* ist auf falsch gesetzt. Selbstverständlich hängt der lsh-service dann auch von der Existenz eines syslogd-Dienstes ab. Wenn *pid-file?* wahr ist, schreibt `lshd` seine PID in die Datei namens *pid-file*.

Wenn *initialize?* wahr ist, wird der Startwert zur Verschlüsselung ebenso wie der Rechnerschlüssel bei der Dienstaktivierung erstellt, falls sie noch nicht existieren. Das kann lange dauern und Anwenderinteraktion kann dabei erforderlich sein.

Wenn *initialize?* falsch ist, bleibt es dem Nutzer überlassen, den Zufallsgenerator zu initialisieren (siehe Abschnitt “lsh-make-seed” in *LSH Manual*) und ein Schlüsselpaar zu erzeugen, dessen privater Schlüssel in der mit *host-key* angegebenen Datei steht (siehe Abschnitt “lshd basics” in *LSH Manual*).

Wenn *interfaces* leer ist, lauscht lshd an allen Netzwerkschnittstellen auf Verbindungen, andernfalls muss *interfaces* eine Liste von Rechnernamen („Host“-Namen) oder Adressen bezeichnen.

allow-empty-passwords? gibt an, ob Anmeldungen mit leeren Passwörtern akzeptiert werden sollen, und *root-login?* gibt an, ob Anmeldungen als Administratornutzer „root“ akzeptiert werden sollen.

Die anderen Felder sollten selbsterklärend sein.

`openssh-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den OpenSSH-Secure-Shell-Daemon (`http://www.openssh.org`), `sshd`. Sein Wert muss ein `openssh-configuration`-Verbundsobjekt wie in diesem Beispiel sein:

```
(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"))))))
```

Siehe unten für Details zur `openssh-configuration`.

Dieser Dienst kann mit weiteren autorisierten Schlüsseln erweitert werden, wie in diesem Beispiel:

```
(service-extension openssh-service-type
```

```
                          (const `(("charlie"
                                    ,(local-file "charlie.pub")))))
```

`openssh-configuration` [Datentyp]

Dies ist der Verbundstyp für die Konfiguration von OpenSSHs `sshd`.

`openssh` (Vorgabe: *openssh*)
: Das zu benutzende OpenSSH-Paket.

`pid-file` (Vorgabe: `"/var/run/sshd.pid"`)
: Der Name der Datei, in die `sshd` seine PID schreibt.

`port-number` (Vorgabe: `22`)
: Der TCP-Port, auf dem `sshd` auf eingehende Verbindungen lauscht.

`max-connections` (Vorgabe: `200`)
: Harte Grenze, wie viele Client-Verbindungen gleichzeitig möglich sind, durchgesetzt durch den im inetd-Stil startenden Shepherd-Dienst (siehe Abschnitt "Service De- and Constructors" in *The GNU Shepherd Manual*).

`permit-root-login` (Vorgabe: `#f`)
: Dieses Feld bestimmt, ob und wann Anmeldungen als Administratornutzer „root“ erlaubt sind. Wenn es `#f` ist, sind Anmeldungen als Administrator gesperrt, bei `#t` sind sie erlaubt. Wird hier das Symbol `'prohibit-password` angegeben, dann sind Anmeldungen als Administrator erlaubt, aber nur, wenn keine passwortbasierte Authentifizierung verwendet wird.

`allow-empty-passwords?` (Vorgabe: `#f`)
: Wenn dies auf wahr gesetzt ist, können sich Nutzer, deren Passwort leer ist, anmelden. Ist es falsch, können sie es nicht.

`password-authentication?` (Vorgabe: `#t`)
: Wenn dies wahr ist, können sich Benutzer mit ihrem Passwort anmelden. Wenn es falsch ist, müssen sie andere Authentisierungsmethoden benutzen.

`public-key-authentication?` (Vorgabe: `#t`)
: Wenn dies wahr ist, können Benutzer zur Anmeldung mit ihrem öffentlichen Schlüssel authentifiziert werden. Wenn es falsch ist, müssen sie andere Authentisierungsmethoden benutzen.

  Autorisierte öffentliche Schlüssel werden in `~/.ssh/authorized_keys` gespeichert. Dies wird nur für Protokollversion 2 benutzt.

`x11-forwarding?` (Vorgabe: `#f`)
: Wenn dies auf wahr gesetzt ist, ist das Weiterleiten von Verbindungen an grafische X11-Clients erlaubt – mit anderen Worten funktionieren dann die `ssh`-Befehlszeilenoptionen `-X` und `-Y`.

`allow-agent-forwarding?` (Vorgabe: `#t`)
: Ob Weiterleitung an den SSH-Agenten zugelassen werden soll.

`allow-tcp-forwarding?` (Vorgabe: `#t`)
: Ob Weiterleitung von TCP-Kommunikation zugelassen werden soll.

`gateway-ports?` (Vorgabe: `#f`)
: Ob Ports als Zugang für eingehende Verbindungen („Gateway-Ports“) weitergeleitet werden dürfen.

`challenge-response-authentication?` (Vorgabe: `#f`)
: Gibt an, ob „Challenge-Response“-Authentifizierung zugelassen wird (z.B. über PAM).

`use-pam?` (Vorgabe: `#t`)
: Aktiviert die Pluggable-Authentication-Module-Schnittstelle. Wenn es auf `#t` gesetzt ist, wird dadurch PAM-Authentisierung über `challenge-response-authentication?` und `password-authentication?` aktiviert, zusätzlich zur Verarbeitung von PAM-Konten und Sitzungsmodulen für alle Authentisierungsarten.

  Weil PAM-Challenge-Response-Authentisierung oft für dieselben Zwecke wie Passwortauthentisierung eingesetzt wird, sollten Sie entweder `challenge-response-authentication?` oder `password-authentication?` deaktivieren.

`print-last-log?` (Vorgabe: `#t`)
: Hiermit wird angegeben, ob `sshd` Datum und Uhrzeit der letzten Anmeldung anzeigen soll, wenn sich ein Benutzer interaktiv anmeldet.

`subsystems` (Vorgabe: `'(("sftp" "internal-sftp"))`)
: Hiermit werden externe Subsysteme konfiguriert (z.B. ein Dateiübertragungsdaemon).

  Diese werden als Liste von zweielementigen Listen angegeben, von denen jede den Namen des Subsystems und einen Befehl (mit optionalen Befehlszeilenargumenten) benennt, der bei einer Anfrage an das Subsystem ausgeführt werden soll.

  Der Befehl `internal-sftp` implementiert einen SFTP-Server im selben Prozess. Alternativ kann man den `sftp-server`-Befehl angeben:

```
(service openssh-service-type
         (openssh-configuration
          (subsystems
           `(("sftp" ,(file-append openssh "/libexec/sftp-server"))))))
```

`accepted-environment` (Vorgabe: `'()`)
: Eine Liste von Zeichenketten, die die Umgebungsvariablen benennen, die exportiert werden dürfen.

  Jede Zeichenkette wird zu einer eigenen Zeile in der Konfigurationsdatei. Siehe die Option `AcceptEnv` in `man sshd_config`.

  Mit diesem Beispiel können SSH-Clients die Umgebungsvariable `COLORTERM` exportieren. Sie wird von Terminal-Emulatoren gesetzt, die Farben unterstützen. Sie können Sie in der Ressourcendatei Ihrer Shell benutzen, um Farben in der Eingabeaufforderung und in Befehlen zu aktivieren, wenn diese Variable gesetzt ist.

```
(service openssh-service-type
```

```
                    (openssh-configuration
                      (accepted-environment '("COLORTERM"))))
```

`authorized-keys` (Vorgabe: `'()`)
: Dies ist die Liste der autorisierten Schlüssel. Jedes Element der Liste ist ein Benutzername gefolgt von einem oder mehr dateiartigen Objekten, die öffentliche SSH-Schlüssel repräsentieren. Zum Beispiel werden mit

```
(openssh-configuration
  (authorized-keys
    `(("rekado" ,(local-file "rekado.pub"))
      ("chris" ,(local-file "chris.pub"))
      ("root" ,(local-file "rekado.pub") ,(local-file "chris.pub")))))
```

die angegebenen öffentlichen Schlüssel für die Benutzerkonten `rekado`, `chris` und `root` registriert.

Weitere autorisierte Schlüssel können als `service-extension` hinzugefügt werden.

Beachten Sie, dass das hier neben `~/.ssh/authorized_keys` *ohne* sich zu stören benutzt werden kann.

`generate-host-keys?` (Vorgabe: `#t`)
: Ob Schlüsselpaare für den Rechner mit `ssh-keygen -A` unter `/etc/ssh` erzeugt werden sollen, wenn es noch keine gibt.

Das Erzeugen von Schlüsselpaaren dauert nur ein paar Sekunden, wenn genug Entropie vorrätig ist, und findet nur einmal statt. Wenn Sie es z.B. auf einer virtuellen Maschine *nicht* brauchen, etwa weil Sie die Rechnerschlüssel schon von anderswo bekommen und die zusätzliche Zeit beim Systemstart ein Problem ist, schalten Sie es vielleicht lieber aus.

`log-level` (Vorgabe: `'info`)
: Dieses Symbol gibt die Stufe der Protokollierung an: `quiet` (schweigsam), `fatal`, `error`, `info`, `verbose` (ausführlich), `debug` (Fehlersuche) etc. Siehe die Handbuchseite für `sshd_config` für die vollständige Liste der Stufenbezeichnungen.

`extra-content` (Vorgabe: `""`)
: Dieses Feld kann benutzt werden, um beliebigen Text an die Konfigurationsdatei anzuhängen. Es ist besonders bei ausgeklügelten Konfigurationen nützlich, die anders nicht ausgedrückt werden können. Zum Beispiel würden mit dieser Konfiguration Anmeldungen als Administratornutzer „root“ grundsätzlich untersagt, lediglich für eine bestimmte IP-Adresse wären sie zugelassen:

```
(openssh-configuration
  (extra-content "\
Match Address 192.168.0.1
  PermitRootLogin yes"))
```

`dropbear-service` [*Konfiguration*] [Scheme-Prozedur]
Den Dropbear-SSH-Daemon (`https://matt.ucc.asn.au/dropbear/dropbear.html`) mit der angegebenen *Konfiguration* ausführen, einem `<dropbear-configuration>`-Objekt.

Wenn Sie zum Beispiel einen Dropbear-Dienst angeben möchten, der auf Port 1234 lauscht, dann fügen Sie diesen Aufruf ins `services`-Feld des Betriebssystems ein:

```
(dropbear-service (dropbear-configuration
                   (port-number 1234)))
```

`dropbear-configuration` [Datentyp]
Dieser Datentyp repräsentiert die Konfiguration eines Dropbear-SSH-Daemons.

`dropbear` (Vorgabe: *dropbear*)
Das zu benutzende Dropbear-Paket.

`port-number` (Vorgabe: 22)
Die Portnummer des TCP-Ports, auf dem der Daemon auf eingehende Verbindungen wartet.

`syslog-output?` (Vorgabe: `#t`)
Ob eine Ausgabe für Syslog aktiviert sein soll.

`pid-file` (Vorgabe: `"/var/run/dropbear.pid"`)
Der Dateiname der PID-Datei des Daemons.

`root-login?` (Vorgabe: `#f`)
Ob Anmeldungen als Administratornutzer `root` möglich sein sollen.

`allow-empty-passwords?` (Vorgabe: `#f`)
Ob leere Passwörter zugelassen sein sollen.

`password-authentication?` (Vorgabe: `#t`)
Ob passwortbasierte Authentisierung zugelassen sein soll.

`autossh-service-type` [Scheme-Variable]
Dies ist der Diensttyp für das AutoSSH-Programm (`https://www.harding.motd.ca/autossh`), das eine Kopie von `ssh` ausführt und diese überwacht. Bei Bedarf wird sie neugestartet, für den Fall, dass sie abstürzt oder keine Kommunikation mehr verarbeitet. AutoSSH kann von Hand aus der Befehlszeile heraus aufgerufen werden, indem man Argumente an die Binärdatei `autossh` aus dem Paket `autossh` übergibt, aber es kann auch als ein Guix-Dienst ausgeführt werden. Letzteres wird hier beschrieben.

AutoSSH kann benutzt werden, um an den lokalen Rechner gerichtete Kommunikation an eine entfernte Maschine über einen SSH-Tunnel weiterzuleiten. Dabei gilt die `~/.ssh/config`-Datei des AutoSSH ausführenden Benutzers.

Um zum Beispiel einen Dienst anzugeben, der autossh mit dem Benutzerkonto `pino` ausführt und alle lokalen Verbindungen auf Port `8081` an `entfernt:8081` über einen SSH-Tunnel durchzureichen, fügen Sie folgenden Aufruf in das `services`-Feld des Betriebssystems ein:

```
(service autossh-service-type
         (autossh-configuration
          (user "pino")
          (ssh-options (list "-T" "-N" "-L" "8081:localhost:8081" "entfernt.net")
```

`autossh-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des AutoSSH-Dienstes.

`user` (Vorgabe: `"autossh"`)
: Das Benutzerkonto, mit dem der AutoSSH-Dienst ausgeführt wird. Es wird vorausgesetzt, dass der angegebene Benutzer existiert.

`poll` (Vorgabe: `600`)
: Gibt an, wie oft die Verbindung geprüft wird („Poll Time“), in Sekunden.

`first-poll` (Vorgabe: `#f`)
: Gibt an, wie viele Sekunden AutoSSH vor der ersten Verbindungsprüfung abwartet. Nach dieser ersten Prüfung werden weitere Prüfungen mit der in `poll` angegebenen Regelmäßigkeit durchgeführt. Wenn dies auf `#f` gesetzt ist, erfährt die erste Verbindungsprüfung keine Sonderbehandlung, sondern benutzt die gleichen Zeitabstände, die auch mit `poll` festgelegt wurden.

`gate-time` (Vorgabe: `30`)
: Gibt an, wie viele Sekunden lang eine SSH-Verbindung aktiv sein muss, bis sie als erfolgreich angesehen wird.

`log-level` (Vorgabe: `1`)
: Die Protokollierungsstufe. Sie entspricht den bei Syslog verwendeten Stufen, d.h. `0` verschweigt die meisten Ausgaben, während `7` die gesprächigste Stufe ist.

`max-start` (Vorgabe: `#f`)
: Wie oft SSH (neu) gestartet werden darf, bevor AutoSSH aufgibt und sich beendet. Steht dies auf `#f`, gibt es keine Begrenzung und AutoSSH kann endlos neu starten.

`message` (Vorgabe: `""`)
: Welche Nachricht beim Prüfen von Verbindungen an die zurückkommende Echo-Nachricht angehängt werden soll.

`port` (Vorgabe: `"0"`)
: Welche Ports zum Überprüfen der Verbindung benutzt werden. Steht dies auf `"0"`, werden keine Überprüfungen durchgeführt. Steht es auf `"`*`n`*`"` für eine positive ganze Zahl *n*, werden die Ports *n* und *n*+1 zum Überwachen der Verbindung eingesetzt, wobei auf Port *n* Daten dafür gesendet werden und Port `n+1` deren Echo empfangen soll. Steht es auf `"`*`n`*`:`*`m`*`"` für positive ganze Zahlen *n* und *m*, werden Ports *n* und *m* zur Überprüfung eingesetzt, wozu Port *n* zum Senden der Prüfdaten eingesetzt wird und Port *m* das Echo empfängt.

`ssh-options` (Vorgabe: `'()`)
: Die Liste der Befehlszeilenargumente, die an `ssh` weitergegeben werden sollen, wenn es ausgeführt wird. Die Befehlszeilenoptionen `-f` und `-M` sind AutoSSH vorbehalten; sie anzugeben, führt zu undefiniertem Verhalten.

`webssh-service-type` [Scheme-Variable]

Dies ist der Diensttyp für das Programm WebSSH (`https://webssh.huashengdun.org/`), das einen webbasierten SSH-Client ausführt. WebSSH kann von Hand aus der Befehlszeile heraus aufgerufen werden, indem man Argumente an die Binärdatei `wssh` aus dem Paket `webssh` übergibt, aber es kann auch als ein Guix-Dienst ausgeführt werden. Letzteres wird hier beschrieben.

Um zum Beispiel einen Dienst anzugeben, der WebSSH an der Loopback-Schnittstelle auf Port `8888` ausführt, wobei die Richtlinie ist, dass Verbindungen abgelehnt werden außer zu einer Liste von erlaubten Rechnern, und eine auf HTTPS-Verbindungen lauschende NGINX als inversen Proxy dafür fungieren zu lassen, fügen Sie folgende Aufrufe in das `services`-Feld des Betriebssystems ein:

```
(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` [Datentyp]

Repräsentiert die Konfiguration für den Dienst `webssh-service`.

`package` (Vorgabe: `webssh`)
: Zu benutzendes `webssh`-Paket.

`user-name` (Vorgabe: `"webssh"`)
: Der Benutzername oder der Benutzeridentifikator (d.h. die „User-ID“), mit dem Dateiübertragungen zum und vom Modul stattfinden sollen.

`group-name` (Vorgabe: `"webssh"`)
: Benutzergruppenname oder Gruppenidentifikator („Group-ID“), mit dem auf das Modul zugegriffen wird.

`address` (Vorgabe: `#f`)
: Die IP-Adresse, auf der `webssh` auf eingehende Verbindungen lauscht.

`port` (Vorgabe: `8888`)
: Der TCP-Port, auf dem `webssh` auf eingehende Verbindungen lauscht.

`policy` (Vorgabe: `#f`)
: Die Verbindungsrichtlinie `reject` setzt voraus, dass erlaubte Rechner in *known-hosts* eingetragen werden.

`known-hosts` (Vorgabe: `'()`)
: Eine Liste der Rechner, die für eine SSH-Verbindung aus `webssh` zugelassen sind.

`log-file` (Vorgabe: `"/var/log/webssh.log"`)
: Der Name der Datei, in die `webssh` seine Protokolle schreibt.

`log-level` (Vorgabe: `#f`)
: Das Protokollierungsniveau.

`%facebook-host-aliases` [Scheme-Variable]

Diese Variable enthält eine Zeichenkette, die Sie für `/etc/hosts` benutzen können (siehe Abschnitt "Host Names" in *Referenzhandbuch der GNU-C-Bibliothek*). Jede Zeile enthält einen Eintrag, der einen bekannten Servernamen des Facebook-Online-Dienstes – z.B. `www.facebook.com` – an den lokalen Rechner umleitet – also an `127.0.0.1` oder dessen IPv6-Gegenstück `::1`.

Normalerweise wird diese Variable im Feld `hosts-file` einer `operating-system`-Betriebssystemdeklaration benutzt (siehe Abschnitt 12.2 [„operating-system"-Referenz], Seite 258):

```
(use-modules (gnu) (guix))

(operating-system
  (host-name "mymachine")
  ;; ...
  (hosts-file
    ;; Eine /etc/hosts-Datei, mit der "localhost" und
    ;; "mymachine" als Alias-Namen eingerichtet werden
    ;; und für die Facebook-Servernamen stattdessen
    ;; Alias-Namen benutzt werden.
    (plain-file "hosts"
                (string-append (local-host-aliases host-name)
                               %facebook-host-aliases))))
```

Dieser Mechanismus kann verhindern, dass lokal laufende Programme, wie z.B. Web-Browser, auf Facebook zugreifen.

Das Modul `(gnu services avahi)` stellt die folgende Definition zur Verfügung.

`avahi-service-type` [Scheme-Variable]

Dieser Dienst führt den `avahi-daemon` aus, einen systemweiten mDNS-/DNS-SD-Anbieter, mit dem im lokalen Netzwerk befindliche Geräte erkannt werden können („Service Discovery") und Rechnernamen selbstständig aufgelöst werden können („Zero-Configuration") (siehe `https://avahi.org/`). Sein Wert muss ein `avahi-configuration`-Verbundsobjekt sein – siehe unten.

Dieser Dienst erweitert den Name Service Cache Daemon (nscd), damit er `.local`-Rechnernamen mit nss-mdns (`https://0pointer.de/lennart/projects/nss-mdns/`) auflösen kann. Siehe Abschnitt 12.12 [Name Service Switch], Seite 607, für Informationen zur Auflösung von Rechnernamen.

Des Weiteren wird das *avahi*-Paket zum Systemprofil hinzugefügt, damit Befehle wie `avahi-browse` einfach benutzt werden können.

`avahi-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration von Avahi.

`host-name` (Vorgabe: `#f`)
: Wenn dies auf etwas anderes als `#f` gesetzt ist, wird es anderen als Rechnername für diese Maschine mitgeteilt, andernfalls wird der tatsächliche Rechnername anderen mitgeteilt.

`publish?` (Vorgabe: `#t`)
: Wenn es auf wahr gesetzt ist, dürfen Rechnernamen und Avahi-Dienste über das Netzwerk mitgeteilt werden (als Broadcast).

`publish-workstation?` (Vorgabe: `#t`)
: Wenn es auf wahr gesetzt ist, teilt `avahi-daemon` den Rechnernamen dieser Maschine und die IP-Adresse über mDNS auf dem lokalen Netzwerk öffentlich mit. Um die auf Ihrem lokalen Netzwerk mitgeteilten Rechnernamen zu sehen, können Sie das hier ausführen:

```
avahi-browse _workstation._tcp
```

`wide-area?` (Vorgabe: `#f`)
: Wenn dies auf wahr gesetzt ist, ist DNS-SD über „Unicast DNS“ aktiviert.

`ipv4?` (Vorgabe: `#t`)
`ipv6?` (Vorgabe: `#t`)
: Mit diesen Feldern wird festgelegt, ob IPv4-/IPv6-Sockets verwendet werden.

`domains-to-browse` (Vorgabe: `'()`)
: Dies ist eine Liste von Domänen, die durchsucht werden.

`openvswitch-service-type` [Scheme-Variable]

Dies ist der Diensttyp des Open-vSwitch-Dienstes (`https://www.openvswitch.org`), der als Wert ein `openvswitch-configuration`-Objekt hat.

`openvswitch-configuration` [Datentyp]

Der Datentyp, der die Konfiguration von Open vSwitch repräsentiert, einem auf mehreren Schichten arbeitenden („multilayer“) virtuellen Switch, der für massenhafte Netzwerkautomatisierung durch programmatische Erweiterungen eingesetzt werden kann.

`package` (Vorgabe: *openvswitch*)
: Das Paketobjekt vom Open vSwitch.

`pagekite-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den PageKite-Dienst (`https://memcached.org/`), einem Angebot zur getunnelten Netzwerkumleitung, womit bloß auf localhost lauschende Server öffentlich erreichbar gemacht werden können. Mit PageKite werden die Server für andere erreichbar, selbst wenn Ihre Maschine nur über eine restriktive Firewall oder eine Netzwerkadressübersetzung („Network Address Translation“, NAT) ohne Portweiterleitung mit dem Internet verbunden ist. Der Wert dieses Dienstes ist ein `pagekite-configuration`-Verbundsobjekt.

Hier ist ein Beispiel, wodurch die lokal laufenden HTTP- und SSH-Daemons zugänglich gemacht werden:

```
(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` [Datentyp]

Der Datentyp, der die Konfiguration von PageKite repräsentiert.

`package` (Vorgabe: *pagekite*)
: Paketobjekt von PageKite.

`kitename` (Vorgabe: `#f`)
: PageKite-Name, um sich gegenüber dem Vordergrundserver zu authentisieren.

`kitesecret` (Vorgabe: `#f`)
: Das gemeinsame Geheimnis, das eine Authentisierung gegenüber dem Vordergrundserver ermöglicht. Wahrscheinlich sollten Sie es besser als Teil von `extra-file` angeben.

`frontend` (Vorgabe: `#f`)
: Eine Verbindung zum angegebenen PageKite-Vordergrundserver herstellen statt zu dem Dienst von pagekite.net.

`kites` (Vorgabe: `'("http:@kitename:localhost:80:@kitesecret")`)
: Die Liste der zu benutzenden „Kites“ für Dienste. Nach Vorgabe wird der HTTP-Server auf Port 80 nach außen hin zugänglich gemacht. Dienste sind im Format `protokoll:kitename:rechnername:port:geheimnis` anzugeben.

`extra-file` (Vorgabe: `#f`)
: Eine Konfigurationsdatei, die zusätzlich eingelesen werden soll. Es wird erwartet, dass Sie diese manuell erstellen. Benutzen Sie sie, wenn Sie zusätzliche Optionen angeben möchten und um gemeinsame Geheimnisse abseits von Guix' Zuständigkeitsbereich zu verwalten.

`yggdrasil-service-type` [Scheme-Variable]

Der Diensttyp, um eine Verbindung mit dem Yggdrasil-Netzwerk (`https://yggdrasil-network.github.io/`) herzustellen, einer frühen Implementierungsstufe eines völlig Ende-zu-Ende-verschlüsselten IPv6-Netzwerks.

Die Wegfindung im Yggdrasil-Netzwerk verläuft unabhängig vom Namen der Netzwerkknoten („Name-independent Routing“) und verwendet kryptografisch erzeugte Adressen. Durch die statische Adressierung können Sie dieselbe Adresse so lange weiterbenutzen, wie Sie möchten, selbst wenn Sie sich an einem anderen Ort befinden als vorher, und Sie können auch, wann immer Sie möchten, eine neue Adresse erzeugen (durch Erzeugung neuer Schlüssel). Siehe `https://yggdrasil-network.github.io/2018/07/28/addressing.html`

Übergeben Sie einen Wert vom Typ `yggdrasil-configuration`, um ihn eine Verbindung zu öffentlichen und/oder lokalen Netzwerkteilnehmern („Peers“) herstellen zu lassen.

Hier ist ein Beispiel für die Nutzung mit öffentlichen Peers und einer statischen Adresse. Die statischen Schlüssel zum Signieren und Verschlüsseln werden in `/etc/yggdrasil-private.conf` definiert (dies ist die Vorgabe für `config-file`).

```
;; Teil des operating-system in der Betriebssystemdeklaration
(service yggdrasil-service-type
              (yggdrasil-configuration
               (autoconf? #f) ;nur öffentliche Peers benutzen
               (json-config
               ;; nehmen Sie eine von
               ;; https://github.com/yggdrasil-network/public-peers
                '((peers . #("tcp://1.2.3.4:1337"))))
               ;; /etc/yggdrasil-private.conf ist der Vorgabewert von config-file
               ))

# Beispielinhalt für /etc/yggdrasil-private.conf
{
  # Ihr öffentlicher Schlüssel. Ihre Peers können Sie um
  # den Schlüssel bitten, um ihn in deren Konfiguration der
  # AllowedPublicKeys einzutragen.
  PublicKey: 64277...

  # Ihr privater Signierschlüssel. Geben Sie ihn NICHT weiter!
  PrivateKey: 5c750...
}
```

`yggdrasil-configuration` [Datentyp]

Repräsentiert die Konfiguration von Yggdrasil.

`package` (Vorgabe: `yggdrasil`)
: Paketobjekt von Yggdrasil.

`json-config` (Vorgabe: `'()`)
: Der Inhalt von `/etc/yggdrasil.conf`. Er wird mit `/etc/yggdrasil-private.conf` zusammengelegt. Beachten Sie, dass diese Einstellungen im Guix-Store gespeichert werden, auf den alle Nutzer Zugriff haben. **Speichern Sie dort nicht Ihre privaten Schlüssel**. Siehe die Ausgabe von `yggdrasil -genconf` für eine kurze Übersicht über gültige Schlüssel und ihre Vorgabewerte.

`autoconf?` (Vorgabe: `#f`)
: Ob der automatische Modus benutzt werden soll. Ihn zu aktivieren, bedeutet, dass Yggdrasil eine dynamische IP-Adresse benutzt und sich mit IPv6-Nachbarn verbindet.

`log-level` (Vorgabe: `'info`)
: Wie detailliert die Protokolle sein sollen. Schreiben Sie `'debug` für einen höheren Detailgrad.

`log-to` (Vorgabe: `'stdout`)
: Wohin Protokolle geschickt werden. Die Vorgabe ist, dass der Dienst ein Protokoll seiner Standardausgabe in `/var/log/yggdrasil.log` schreibt. Die Alternative ist `'syslog`, wodurch die Ausgabe an den laufenden syslog-Dienst geschickt wird.

`config-file` (Vorgabe: `"/etc/yggdrasil-private.conf"`)
: Wo die HJSON-Datei liegt, aus der sensible Daten geladen werden. Hier sollten private Schlüssel gespeichert werden, wodurch nicht nach jedem Neustart eine zufällige Adresse benutzt wird. Verwenden Sie `#f` zum Deaktivieren. In dieser Datei festgelegte Optionen haben Vorrang vor `json-config`. Tragen Sie anfangs die Ausgabe von `yggdrasil -genconf` ein. Um eine statische Adresse zu benutzen, löschen Sie alles außer den folgenden Optionen:

  - `EncryptionPublicKey`
  - `EncryptionPrivateKey`
  - `SigningPublicKey`
  - `SigningPrivateKey`

`ipfs-service-type` [Scheme-Variable]
: Der Diensttyp, um sich mit dem IPFS-Netzwerk (`https://ipfs.io`) zu verbinden, einem weltweiten, versionierten, von einem Netzwerkteilnehmer zum anderen („peer-to-peer“) verteilten Dateisystem. Geben Sie ein `ipfs-configuration`-Verbundsobjekt an, um die für den Netzwerkzugang (Gateway) und die Anwendungsschnittstelle (API) verwendeten Ports zu ändern.

  Hier ist eine Beispielkonfiguration, um nicht standardmäßige Ports einzusetzen:

```
(service ipfs-service-type
         (ipfs-configuration
          (gateway "/ip4/127.0.0.1/tcp/8880")
          (api "/ip4/127.0.0.1/tcp/8881")))
```

`ipfs-configuration` [Datentyp]
: Repräsentiert die Konfiguration des IPFS.

  `package` (Vorgabe: `go-ipfs`)
  : Paketobjekt von IPFS.

  `gateway` (Vorgabe: `"/ip4/127.0.0.1/tcp/8082"`)
  : Die Adresse des Netzwerkzugangs im Format einer Multiadresse („multiaddress“).

`api` (Vorgabe: `"/ip4/127.0.0.1/tcp/5001"`)
: Die Adresse des API-Endpunkts im Format einer Multiadresse („multiaddress“).

`keepalived-service-type` [Scheme-Variable]
: Dies ist der Diensttyp für die Software Keepalived (`https://www.keepalived.org/`) für virtuelle Router, `keepalived`. Sein Wert muss ein `keepalived-configuration`-Verbundsobjekt sein, wie in diesem Beispiel für die Maschine erster Wahl (Master):

```
(service keepalived-service-type
         (keepalived-configuration
           (config-file (local-file "keepalived-master.conf"))))
```

Dabei enthält `keepalived-master.conf`:

```
vrrp_instance meine-gruppe {
  state MASTER
  interface enp9s0
  virtual_router_id 100
  priority 100
  unicast_peer { 10.0.0.2 }
  virtual_ipaddress {
    10.0.0.4/24
  }
}
```

Für die Ersatzmaschine (Backup):

```
(service keepalived-service-type
         (keepalived-configuration
          (config-file (local-file "keepalived-backup.conf"))))
```

Dort enthält `keepalived-backup.conf`:

```
vrrp_instance meine-gruppe {
  state BACKUP
  interface enp9s0
  virtual_router_id 100
  priority 99
  unicast_peer { 10.0.0.3 }
  virtual_ipaddress {
    10.0.0.4/24
  }
}
```

12.9.6 Unbeaufsichtigte Aktualisierungen

Guix stellt einen Dienst zum Durchführen *unbeaufsichtigter Aktualisierungen* zur Verfügung: Das System rekonfiguriert sich selbst in regelmäßigen Abständen mit der neuesten Version von Guix. Guix System verfügt über mehrere Eigenschaften, durch die unbeaufsichtigtes Aktualisieren zu einer sicheren Angelegenheit wird:

- Aktualisierungen sind transaktionell: Entweder ist die Aktualisierung erfolgreich oder sie schlägt fehl, aber Sie können sich nicht in einem Systemzustand „dazwischen“ wiederfinden.

- Ein Protokoll über die Aktualisierungen liegt vor – Sie können es mit `guix system list-generations` einsehen – und Rücksetzungen auf vorherige Generationen sind möglich, für den Fall, dass das System nach der Aktualisierung nicht mehr funktioniert oder sich nicht wie gewollt verhält.
- Der Code für Kanäle wird authentifiziert, wodurch Sie sich sicher sein können, dass Sie nur unverfälschten Code ausführen (siehe Kapitel 7 [Kanäle], Seite 77).
- `guix system reconfigure` verweigert ein Herabstufen auf alte Versionen; das macht *Herabstufungsangriffe* („Downgrade Attacks“) unmöglich.

Um unbeaufsichtigte Aktualisierungen einzurichten, fügen Sie eine Instanz des Diensttyps `unattended-upgrade-service-type` wie im folgenden Beispiel zur Liste Ihrer Betriebssystemdienste hinzu:

```
(service unattended-upgrade-service-type)
```

Durch die Vorgabeeinstellungen hat dies wöchentliche Aktualisierungen zur Folge, jeden Sonntag um Mitternacht. Sie müssen keine Betriebssystemkonfigurationsdatei angeben: `/run/current-system/configuration.scm` wird benutzt, wodurch immer Ihre letzte Konfiguration weiter verwendet wird – siehe den [provenance-service-type], Seite 643, für mehr Informationen über diese Datei.

Mehrere Dinge können konfiguriert werden, insbesondere die Zeitabstände zwischen Aktualisierungen und die Dienste (also die Daemons), die nach Abschluss neu gestartet werden sollen. Wenn die Aktualisierung erfolgreich ist, dann stellt das System sicher, dass Systemgenerationen mit einem Alter größer als ein festgelegter Schwellwert gelöscht werden, wie bei `guix system delete-generations`. Siehe die folgende Referenz der Konfigurationsmöglichkeiten für Näheres.

Um sicherzugehen, dass unbeaufsichtigte Aktualisierungen auch tatsächlich durchgeführt werden, können Sie `guix system describe` ausführen. Um Fehlschläge zu untersuchen, schauen Sie in die Protokolldatei für unbeaufsichtigte Aktualisierungen (siehe unten).

`unattended-upgrade-service-type` [Scheme-Variable]

Dies ist der Diensttyp für unbeaufsichtigte Aktualisierungen. Damit wird ein mcron-Auftrag eingerichtet (siehe Abschnitt 12.9.2 [Geplante Auftragsausführung], Seite 301), der `guix system reconfigure` mit der neuesten Version der angegebenen Kanäle ausführt.

Sein Wert muss ein `unattended-upgrade-configuration`-Verbundsobjekt sein (siehe unten).

`unattended-upgrade-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des Dienstes für unbeaufsichtigte Aktualisierungen. Folgende Felder stehen zur Verfügung:

`schedule` (Vorgabe: `"30 01 * * 0"`)

Dies ist der Aktualisierungsplan, ausgedrückt als ein G-Ausdruck mit einem mcron-Auftragsplan (siehe Abschnitt “Guile Syntax” in *GNU mcron*).

`channels` (Vorgabe: `#~%default-channels`)

Dieser G-Ausdruck gibt an, welche Kanäle für die Aktualisierung benutzt werden sollen (siehe Kapitel 7 [Kanäle], Seite 77). Nach Vorgabe wird die Spitze des offiziellen `guix`-Kanals benutzt.

`operating-system-file` (Vorgabe: `"/run/current-system/configuration.scm"`)
: Dieses Feld gibt an, welche Betriebssystemkonfigurationsdatei verwendet werden soll. Nach Vorgabe wird die Konfigurationsdatei der aktuellen Konfiguration wiederverwendet.

  Es gibt jedoch Fälle, wo es nicht ausreicht, auf `/run/current-system/configuration.scm` zu verweisen, zum Beispiel weil sich diese mit `local-file` oder Ähnlichem auf weitere Dateien wie öffentliche SSH-Schlüssel, andere Konfigurationsdateien usw. bezieht. Dann empfehlen wir, etwas wie hier zu schreiben:

```
(unattended-upgrade-configuration
  (operating-system-file
    (file-append (local-file "." "config-dir" #:recursive? #t)
                 "/config.scm")))
```

  Das bewirkt, dass das ganze aktuelle Verzeichnis in den Store importiert wird und Bezug auf die `config.scm` darin genommen wird. So funktioniert die Nutzung von `local-file` in `config.scm` wie erwartet. Siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175, für Informationen zu `local-file` und `file-append`.

`services-to-restart` (Vorgabe: `'(mcron)`)
: Dieses Feld gibt die Shepherd-Dienste an, die nach Abschluss der Aktualisierung neu gestartet werden sollen.

  Die Dienste werden direkt nach Abschluss neu gestartet, so wie es `herd restart` tun würde. Dadurch wird sichergestellt, dass die neueste Version läuft – bedenken Sie, dass `guix system reconfigure` nach Vorgabe nur diejenigen Dienste neu startet, die zurzeit nicht laufen, was einem konservativen Verhalten entspricht: Störungen werden minimiert, aber veraltete Dienste laufen weiter.

  Führen Sie `herd status` aus, um herauszufinden, welche Dienste Sie neu starten lassen können. Siehe Abschnitt 12.9 [Dienste], Seite 280, für allgemeine Informationen über Dienste. Oft würde man Dienste wie unter anderem `ntpd` und `ssh-daemon` neu starten lassen.

  Nach Vorgabe wird nur der `mcron`-Dienst neu gestartet. Dadurch wird beim nächsten Mal sicherlich die neueste Version des Auftrags für unbeaufsichtigte Aktualisierungen benutzt.

`system-expiration` (Vorgabe: `(* 3 30 24 3600)`)
: Nach wie vielen Sekunden Systemgenerationen auslaufen. Systemgenerationen, die dieses Alter überschritten haben, werden über `guix system delete-generations` gelöscht, nachdem eine Aktualisierung abgeschlossen wurde.

  > **Anmerkung:** Der Dienst für unbeaufsichtigte Aktualisierungen lässt jedoch den Müllsammler nicht laufen. Sie möchten wahrscheinlich Ihren eigenen mcron-Auftrag einrichten, der regelmäßig `guix gc` ausführt.

`maximum-duration` (Vorgabe: `3600`)
: Wie viele Sekunden eine Aktualisierung höchstens dauern darf. Nach dieser Zeit wird die Aktualisierung abgebrochen.

  Diese Begrenzung dient vor allem dazu, zu verhindern, dass „die ganze Welt“ neu erstellt oder neu heruntergeladen wird.

`log-file` (Vorgabe: `"/var/log/unattended-upgrade.log"`)
: Die Datei, in die über unbeaufsichtigte Aktualisierungen Protokoll geführt wird.

12.9.7 X Window

Unterstützung für das grafische Anzeigesystem X Window – insbesondere Xorg – wird vom Modul `(gnu services xorg)` zur Verfügung gestellt. Beachten Sie, dass es *keine* `xorg-service`-Prozedur gibt, sondern der X-Server durch eine Software zur *Anmeldeverwaltung* gestartet wird (ein „Login Manager“). Vorgegeben ist, dass zur Anzeigenverwaltung der GNOME Display Manager (GDM) benutzt wird.

GDM ermöglicht es seinen Nutzern natürlich auch, sich bei anderen Fensterverwaltungssystemen und Arbeitsumgebungen als GNOME anzumelden. Wer GNOME benutzt, kann Funktionalitäten wie eine automatische Bildschirmsperre nur verwenden, wenn die Anmeldung über GDM läuft.

Um X11 zu benutzen, müssen Sie ein Programme zur *Fensterverwaltung* („Window-Manager“) oder mehrere davon installieren – zum Beispiel die Pakete `windowmaker` oder `openbox` –, vorzugsweise indem Sie sie in das `packages`-Feld Ihrer Betriebssystemdefinition eintragen (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258).

GDM hat auch Unterstützung für Wayland: Es kann selbst als Wayland-Client gestartet werden und Wayland-Sitzungen starten. Ersteres ist auch die Voraussetzung für Letzteres. Um den Wayland-Modus zu aktivieren, setzen Sie `wayland?` auf `#t` in der `gdm-configuration`.

`gdm-service-type` [Scheme-Variable]
: Dies ist der Diensttyp für den GNOME Desktop Manager (`https://wiki.gnome.org/Projects/GDM/`), GDM, ein Programm zur Verwaltung grafischer Anzeigeserver, das grafische Benutzeranmeldungen durchführt. Sein Wert muss eine `gdm-configuration` sein (siehe unten).

  GDM liest die in den `.desktop`-Dateien in `/run/current-system/profile/share/xsessions` (bei X11-Sitzungen) und nach `/run/current-system/profile/share/wayland-sessions` (bei Wayland-Sitzungen) befindlichen *Sitzungstypen* ein und stellt diese seinen Nutzern zur Auswahl auf dem Anmeldebildschirm. Pakete wie `gnome`, `xfce`, `i3` und `sway` stellen `.desktop`-Dateien bereit; wenn diese Pakete zu den systemweit verfügbaren Paketen hinzugefügt werden, werden diese automatisch auf dem Anmeldebildschirm angezeigt.

  Des Weiteren werden `~/.xsession`-Dateien berücksichtigt. Wenn es vorhanden ist, muss `~/.xsession` eine ausführbare Datei sein, die ein Programm zur Fensterverwaltung und/oder andere X-Clients startet.

`gdm-configuration` [Datentyp]

`auto-login?` (Vorgabe: `#f`)
`default-user` (Vorgabe: `#f`)
Wenn `auto-login?` falsch ist, zeigt GDM einen Anmeldebildschirm an.

Wenn `auto-login?` wahr ist, meldet GDM automatisch den in `default-user` angegebenen voreingestellten Benutzer an.

`auto-suspend?` (Vorgabe: `#t`)
Wenn es wahr ist, wird GDM automatisch das System in den Bereitschaftsmodus schalten, bis wieder jemand persönlich anwesend ist und es aus dem Arbeitsspeicher (RAM) wieder aufwecken kann. Wenn geplant ist, die Maschine für entfernte Sitzungen oder SSH zu benutzen, sollte es auf falsch gesetzt werden, damit GDM nicht entfernte Sitzungen unterbricht oder die Verfügbarkeit der Maschine stört.

`debug?` (Vorgabe: `#f`)
Wenn es wahr ist, schreibt GDM Informationen zur Fehlersuche in sein Protokoll.

`gnome-shell-assets` (Vorgabe: ...)
Liste der GNOME-Shell-„Assets“, die GDM benötigt, d.h. Symbolthema, Schriftarten etc.

`xorg-configuration` (Vorgabe: `(xorg-configuration)`)
Xorg-Server für grafische Oberflächen konfigurieren.

`x-session` (Vorgabe: `(xinitrc)`)
Das Skript, das vor dem Starten einer X-Sitzung ausgeführt werden soll.

`xdmcp?` (Vorgabe: `#f`)
Wenn dies auf wahr gesetzt ist, wird das X Display Manager Control Protocol (XDMCP) aktiviert. Sie sollten es nur in vertrauenswürdigen Umgebungen aktivieren, denn das Protokoll ist *nicht* sicher. Wenn es aktiviert ist, lauscht GDM auf XDMCP-Anfragen auf UDP-Port 177.

`dbus-daemon` (Vorgabe: `dbus-daemon-wrapper`)
Der Dateiname der ausführbaren Datei des `dbus-daemon`-Programms.

`gdm` (Vorgabe: `gdm`)
Das GDM-Paket, was benutzt werden soll.

`wayland?` (Vorgabe: `#f`)
Wenn es wahr ist, wird Wayland in GDM aktiviert. Das ist nötig, um Wayland-Sitzungen zu benutzen.

`wayland-session` (Vorgabe: `gdm-wayland-session-wrapper`)
Das Wrapper-Skript für Wayland-Sitzungen, was benutzt werden soll. Es ist notwendig, um die Umgebung zu starten.

`slim-service-type` [Scheme-Variable]
Dies ist der Diensttyp für die schlanke grafische Anmeldungsverwaltung SLiM für X11.

Wie GDM liest SLiM die in `.desktop`-Dateien beschriebenen Sitzungstypen aus und ermöglicht es Nutzern, eine Sitzung darunter im Anmeldebildschirm durch Drücken von *F1* auszuwählen. Auch `~/.xsession`-Dateien können benutzt werden.

Anders als GDM wird durch SLiM die Benutzersitzung nicht auf einem anderen virtuellen Terminal gestartet, nachdem man sich anmeldet. Die Folge davon ist, dass man nur eine einzige grafische Sitzung starten kann. Wenn Sie mehrere, gleichzeitig laufende grafische Sitzungen starten können möchten, müssen Sie mehrere SLiM-Dienste zu ihren Systemdiensten hinzufügen. Das folgende Beispiel zeigt, wie Sie den vorgegebenen GDM-Dienst durch zwei SLiM-Dienste auf tty7 und tty8 ersetzen.

```
(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` [Datentyp]

Datentyp, der die Konfiguration des `slim-service-type` repräsentiert.

`allow-empty-passwords?` (Vorgabe: `#t`)
: Ob Anmeldungen mit leeren Passwörtern möglich sein sollen.

`gnupg?` (Vorgabe: `#f`)
: Wenn dies aktiviert ist, wird durch `pam-gnupg` automatisch versucht, die GPG-Schlüssel des Nutzers über `gpg-agent` mit dem Anmeldepasswort zu entsperren. Für jeden Schlüssel, der entsperrt werden soll, müssen Sie den Keygrip, der den Schlüssel bezeichnet, in `~/.pam-gnupg` schreiben; er kann mit `gpg -K --with-keygrip` erfragt werden. Damit das funktioniert, müssen Sie Presetting von Passphrasen aktivieren, indem Sie `allow-preset-passphrase` in `~/.gnupg/gpg-agent.conf` hinzufügen.

`auto-login?` (Vorgabe: `#f`)
`default-user` (Vorgabe: `""`)
: Wenn `auto-login?` falsch ist, zeigt SLiM einen Anmeldebildschirm an.

  Wenn `auto-login?` wahr ist, meldet SLiM automatisch den in `default-user` angegebenen voreingestellten Benutzer an.

`theme` (Vorgabe: `%default-slim-theme`)
`theme-name` (Vorgabe: `%default-slim-theme-name`)
: Das grafische Thema, was benutzt werden soll, mit seinem Namen.

`auto-login-session` (Vorgabe: `#f`)
: Wenn es wahr ist, muss es den Namen der ausführbaren Datei angeben, die als voreingestellte Sitzung gestartet werden soll – z.B. `(file-append windowmaker "/bin/windowmaker")`.

  Wenn es falsch ist, wird eine von einer der `.desktop`-Dateien in `/run/current-system/profile` und `~/.guix-profile` beschriebenen Sitzungen benutzt.

  > **Anmerkung:** Sie müssen mindestens ein Fensterverwaltungsprogramm in das Systemprofil oder Ihr Benutzerprofil installieren, ansonsten können Sie sich, sofern `auto-login-session` falsch ist, nicht anmelden.

`xorg-configuration` (Vorgabe: `(xorg-configuration)`)
: Xorg-Server für grafische Oberflächen konfigurieren.

`display` (Vorgabe: `":0"`)
: Die Anzeige, auf welcher der Xorg-Server für grafische Oberflächen gestartet werden soll.

`vt` (Vorgabe: `"vt7"`)
: Das virtuelle Terminal, auf dem der Xorg-Server für grafische Oberflächen gestartet werden soll.

`xauth` (Vorgabe: `xauth`)
: Das XAuth-Paket, das benutzt werden soll.

`shepherd` (Vorgabe: `shepherd`)
: Das Shepherd-Paket, das benutzt wird, wenn `halt` und `reboot` aufgerufen werden.

`sessreg` (Vorgabe: `sessreg`)
: Das sessreg-Paket, das zum Registrieren der Sitzung benutzt werden soll.

`slim` (Vorgabe: `slim`)
: Das zu benutzende SLiM-Paket.

`%default-theme` [Scheme-Variable]
`%default-theme-name` [Scheme-Variable]
: Das vorgegebene Thema für das Aussehen von SLiM mit seinem Namen.

`sddm-service-type` [Scheme-Variable]
: Dies ist der Typ des Dienstes, mit dem die SDDM-Anzeigenverwaltung (`https://github.com/sddm/sddm`) gestartet wird. Sein Wert muss ein `sddm-configuration`-Verbundsobjekt sein (siehe unten).

  Hier ist ein Beispiel für seine Verwendung:

```
(service sddm-service-type
         (sddm-configuration
           (auto-login-user "alice")
           (auto-login-session "xfce.desktop")))
```

`sddm-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration der SDDM-Anmeldeverwaltung. Die verfügbaren Felder sind:

`sddm` (Vorgabe: `sddm`)
: Das SDDM-Paket, was benutzt werden soll.

`display-server` (Vorgabe: `"x11"`)
: Einen Anzeigeserver auswählen, der für den Anmeldebildschirm verwendet werden soll. Zulässige Werte sind ‘`"x11"`’ oder ‘`"wayland"`’.

`numlock` (Vorgabe: "on")
: Gültige Werte sind ‘`"on"`’, ‘`"off"`’ oder ‘`"none"`’.

`halt-command` (Vorgabe: `#~(string-append #$shepherd "/sbin/halt")`)
: Der Befehl, der zum Anhalten des Systems ausgeführt wird.

`reboot-command` (Vorgabe: `#~(string-append #$shepherd "/sbin/reboot")`)
: Der Befehl, der zum Neustarten des Systems ausgeführt wird.

`theme` (Vorgabe: "maldives")
: Welches Thema für das Aussehen benutzt werden soll. Mit SDDM mitgelieferte Themen sind ‘`"elarun"`’, ‘`"maldives"`’ und ‘`"maya"`’.

`themes-directory` (Vorgabe: "/run/current-system/profile/share/sddm/themes")
: Verzeichnis, wo Themen gefunden werden können.

`faces-directory` (Vorgabe: "/run/current-system/profile/share/sddm/faces")
: Verzeichnis, wo Avatarbilder gefunden werden können.<

`default-path` (Vorgabe: "/run/current-system/profile/bin")
: Welcher PATH voreingestellt sein soll.

`minimum-uid` (Vorgabe: 1000)
: Der kleinste Benutzeridentifikator (UID), mit dem Benutzer in SDDM angezeigt werden und sich anmelden können.

`maximum-uid` (Vorgabe: 2000)
: Der größte Benutzeridentifikator (UID), mit dem Benutzer in SDDM angezeigt werden.<

`remember-last-user?` (Vorgabe: #t)
: Den zuletzt ausgewählten Benutzer voreinstellen.

`remember-last-session?` (Vorgabe: #t)
: Die zuletzt ausgewählte Sitzung voreinstellen.

`hide-users` (Vorgabe: "")
: Benutzernamen, die in SDDM *nicht* sichtbar sein sollen.

`hide-shells` (Vorgabe: `#~(string-append #$shadow "/sbin/nologin")`)
: Benutzerkonten, für die als Shell eine davon eingestellt ist, wird SDDM *nicht* anzeigen.

`session-command` (Vorgabe: `#~(string-append #$sddm "/share/sddm/scripts/wayland-session")`)
: Das Skript, das vor dem Starten einer Wayland-Sitzung ausgeführt werden soll.

`sessions-directory` (Vorgabe: `"/run/current-system/profile/share/wayland-sessions"`)
: Verzeichnis, das nach .desktop-Dateien zum Starten von Wayland-Sitzungen durchsucht wird.

`xorg-configuration` (Vorgabe: `(xorg-configuration)`)
: Xorg-Server für grafische Oberflächen konfigurieren.

`xauth-path` (Vorgabe: `#~(string-append #$xauth "/bin/xauth")`)
: Pfad von xauth.

`xephyr-path` (Vorgabe: `#~(string-append #$xorg-server "/bin/Xephyr")`)
: Pfad von Xephyr.

`xdisplay-start` (Vorgabe: `#~(string-append #$sddm "/share/sddm/scripts/Xsetup")`)
: Skript, das nach dem Starten vom Xorg-Server ausgeführt wird.

`xdisplay-stop` (Vorgabe: `#~(string-append #$sddm "/share/sddm/scripts/Xstop")`)
: Skript, das vor dem Stoppen vom Xorg-Server ausgeführt wird.

`xsession-command` (Vorgabe: `xinitrc`)
: Das Skript, das vor dem Starten einer X-Sitzung ausgeführt werden soll.

`xsessions-directory` (Vorgabe: `"/run/current-system/profile/share/xsessions"`)
: Verzeichnis, das nach .desktop-Dateien zum Starten von X-Sitzungen durchsucht wird.<

`minimum-vt` (Vorgabe: 7)
: Das kleinste virtuelle Terminal, das benutzt werden darf.

`auto-login-user` (Vorgabe: `""`)
: Das Benutzerkonto, mit dem man automatisch angemeldet wird. Wenn es leer ist, gibt es keine automatische Anmeldung.

`auto-login-session` (Vorgabe: `""`)
: Gibt den Namen der `.desktop`-Datei an, die bei automatischer Anmeldung für die Sitzung verwendet wird, oder eine leere Zeichenkette.

`relogin?` (Vorgabe: #f)
: Ob nach dem Abmelden neu angemeldet werden soll.

`lightdm-service-type` [Scheme-Variable]

Dies ist der Typ des Dienstes, mit dem die LightDM-Anzeigenverwaltung (`https://github.com/canonical/lightdm`) ausgeführt wird. Sein Wert muss ein `lightdm-configuration`-Verbundsobjekt sein. Dieses wird weiter unten beschrieben. LightDM zeichnet sich dadurch aus, dass er mit TigerVNC integriert ist und man so leichten

Fernzugriff über das XDMCP-Protokoll einrichten kann, so dass man von außen mit einem entfernten Rechner eine Sitzung anmelden kann.

Für die Grundeinstellung geben Sie einfach an:

```
(service lightdm-service-type)
```

Ein zielgerichteteres Beispiel wäre etwa, die VNC-Fähigkeit und weitere Funktionalitäten zusammen mit ausführlicher Protokollierung zu aktivieren. Das ginge so:

```
(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` [Datentyp]

Verfügbare `lightdm-configuration`-Felder sind:

`lightdm` (Vorgabe: `lightdm`) (Typ: dateiartig)
: Das zu benutzende lightdm-Paket.

`allow-empty-passwords?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob man sich auch bei Benutzerkonten, die ein leeres Passwort haben, anmelden kann.

`debug?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ausführliche Ausgaben schreiben.

`xorg-configuration` (Typ: xorg-configuration)
: Eine Konfiguration des Xorg-Servers, aus der das Start-Skript für den Xorg-Server erzeugt wird. Sie können für jeden Seat mit dem Feld `xserver-command` des `<lightdm-seat-configuration>`-Verbunds genauere Einstellungen vornehmen, wenn Sie möchten.

`greeters` (Typ: Liste-von-„greeter-configuration“)
: Die LightDM-Greeter-Konfigurationen, welche die zu benutzenden Greeter festlegen.

`seats` (Typ: Liste-von-„seat-configuration“)
: Die Konfigurationen der „Seats“. Ein Seat in LightDM entspricht ungefähr einem Benutzer.

`xdmcp?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob ein XDMCP-Server auf UDP-Port 177 lauschen soll.

`xdmcp-listen-address` (Typ: Vielleicht-Zeichenkette)
: Der Rechnername oder die IP-Adresse, wofür der XDMCP-Server auf eingehende Verbindungen lauscht. Wird nichts angegeben, wird auf allen verfügbaren Rechnernamen bzw. IP-Adressen gelauscht.

`vnc-server?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob ein VNC-Server gestartet werden soll.

`vnc-server-command` (Typ: dateiartig)
: Welcher Xvnc-Befehl für den VNC-Server benutzt werden soll. So können zusätzliche, anderweitig unerreichbare Optionen dem Befehl mitgegeben werden. Zum Beispiel können Sie dessen Sicherheitsmaßnahmen abschalten:

```
(vnc-server-command (file-append tigervnc-server "/bin/Xvnc"
                                 " -SecurityTypes None" ))
```

Oder Sie können eine Passwortdatei für den klassischen (unsicheren) Authentifizierungsmechanismus VncAuth festlegen:

```
(vnc-server-command (file-append tigervnc-server "/bin/Xvnc"
                                 " -PasswordFile /var/lib/lightdm/.vnc/
```

Die Passwortdatei sollte man manuell erzeugen mit dem Befehl `vncpasswd`. Beachten Sie, dass LightDM neue Sitzungen für VNC-Nutzer anlegt, deshalb müssen sich diese genauso authentisieren wie lokale Benutzer.

`vnc-server-listen-address` (Typ: Vielleicht-Zeichenkette)
: Auf welchem Rechnernamen oder welcher IP-Adresse der VNC-Server auf eingehende Verbindungen lauscht. Wird nichts angegeben, wird auf allen Rechnernamen und Adressen gelauscht.

`vnc-server-port` (Vorgabe: `5900`) (Typ: Zahl)
: Die TCP-Portnummer, auf der der VNC-Server lauschen soll.

`extra-config` (Vorgabe: `()`) (Typ: Liste-von-Zeichenketten)
: Zusätzliche Konfigurationswerte, die an LightDMs Konfigurationsdatei angehängt werden.

`lightdm-gtk-greeter-configuration` [Datentyp]

Verfügbare `lightdm-gtk-greeter-configuration`-Felder sind:

`lightdm-gtk-greeter` (Vorgabe: `lightdm-gtk-greeter`) (Typ: dateiartig)
: Das „lightdm-gtk-greeter“-Paket, was benutzt werden soll.

`assets` (Vorgabe: `(adwaita-icon-theme gnome-themes-extrahicolor-icon-theme)`)
: (Typ: Liste-von-Dateiartigen) Die Liste von Paketen, die zum Greeter dazugehören, z.B. Pakete mit Symbolthemen, die deren Aussehen bestimmen.

`theme-name` (Vorgabe: `"Adwaita"`) (Typ: Zeichenkette)
: Der Name des zu benutzenden Themas.

`icon-theme-name` (Vorgabe: `\"Adwaita\"`) (Typ: Zeichenkette)
Der Name des zu benutzenden Symbolthemas.

`cursor-theme-name` (Vorgabe: `\"Adwaita\"`) (Typ: Zeichenkette)
Der Name des zu benutzenden Zeigerthemas.

`cursor-theme-size` (Vorgabe: `16`) (Typ: Zahl)
Die Größe, die für das Zeigerthema benutzt wird.

`allow-debugging?` (Typ: Vielleicht-Boolescher-Ausdruck)
Setzen Sie es auf #t, wenn Sie die Fehlersuch-Ausführlichkeitsstufe aktivieren möchten.

`background` (Typ: dateiartig)
Welches Hintergrundbild benutzt werden soll.

`at-spi-enabled?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
Funktionen zur Barrierefreiheit aktivieren. Dazu wird AT-SPI bereitgestellt, eine Schnittstelle für Assistenzprogramme (Assistive Technology Service Provider Interface).

`a11y-states` (Vorgabe: `(contrast font keyboard reader)`) (Typ:
Liste-von-Barrierefreiheiten) Welche Funktionen zur Barrierefreiheit aktiviert sein sollen. Erwartet wird eine Liste von Symbolen.

`reader` (Typ: Vielleicht-dateiartig)
Der Befehl, um einen Bildschirmleser zu starten.

`extra-config` (Vorgabe: `()`) (Typ: Liste-von-Zeichenketten)
Zusätzliche Konfigurationswerte, die an die Konfigurationsdatei des LightDM GTK Greeter angehängt werden.

`lightdm-seat-configuration` [Datentyp]
Verfügbare `lightdm-seat-configuration`-Felder sind:

`name` (Typ: Seat-Name)
Der Name des Seats. Wenn ein Sternchen (*) im Namen des Seats angegeben wird, wirkt sich die Seat-Konfiguration auf alle passenden Seat-Namen aus.

`user-session` (Typ: Vielleicht-Zeichenkette)
Welche Sitzung voreingestellt wird. Der Name der Sitzung muss eine in Kleinbuchstaben geschriebene Zeichenkette sein wie `"gnome"`, `"ratpoison"` und so weiter.

`type` (Vorgabe: `local`) (Typ: Seat-Typ)
Um welchen Typ von Seat es sich handelt, entweder lokal (`local`) oder als Fernverbindung (`xremote`).

`autologin-user` (Typ: Vielleicht-Zeichenkette)
Der Name des Benutzers, als der man automatisch angemeldet wird.

`greeter-session` (Vorgabe: `lightdm-gtk-greeter`) (Typ: Greeter-Sitzung) Welche Greeter-Sitzung geöffnet werden soll, angegeben als Symbol. Derzeit wird nur `lightdm-gtk-greeter` unterstützt.

`xserver-command` (Typ: Vielleicht-dateiartig)
: Welcher Xorg-Server-Befehl ausgeführt wird.

`session-wrapper` (Typ: dateiartig)
: Der xinitrc-Wrapper für die Sitzung.

`extra-config` (Vorgabe: `()`) (Typ: Liste-von-Zeichenketten)
: Zusätzliche Konfigurationswerte, die an den Abschnitt zur Konfiguration des Seats angehängt werden.

`xorg-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des grafischen Anzeigeservers Xorg. Beachten Sie, dass es keinen Xorg-Dienst gibt, sondern der X-Server von einer „Anzeigenverwaltung“ wie GDM, SDDM, LightDM oder SLiM gestartet wird. Deswegen wird aus der Konfiguration dieser Anzeigenverwaltungen ein `xorg-configuration`-Verbundsobjekt konstruiert.

`modules` (Vorgabe: `%default-xorg-modules`)
: Dies ist eine Liste von *Modulpaketen*, die vom Xorg-Server geladen werden – z.B. `xf86-video-vesa`, `xf86-input-keyboard` und so weiter.

`fonts` (Vorgabe: `%default-xorg-fonts`)
: Dies ist eine Liste von Verzeichnissen mit Schriftarten, die zum Schriftartensuchpfad („Font Path“) des Servers hinzugefügt werden.

`drivers` (Vorgabe: `'()`)
: Dies muss entweder die leere Liste sein – in diesem Fall wird durch Xorg automatisch ein Grafiktreiber ausgewählt – oder eine Liste von Treibernamen, die in dieser Reihenfolge durchprobiert werden – z.B. `("modesetting" "vesa")`.

`resolutions` (Vorgabe: `'()`)
: Wenn `resolutions` die leere Liste ist, wird automatisch durch Xorg eine passende Bildschirmauflösung gewählt. Andernfalls muss hier eine Liste von Bildschirmauflösungen angegeben werden – z.B. `((1024 768) (640 480))`.

`keyboard-layout` (Vorgabe: `#f`)
: Wenn es auf `#f` gesetzt ist, benutzt Xorg die voreingestellte Tastaturbelegung, also normalerweise US English („QWERTY“) für eine PC-Tastatur mit 105 Tasten.

  Andernfalls muss hier ein `keyboard-layout`-Objekt stehen, das angibt, welche Tastaturbelegung aktiv sein soll, während Xorg läuft. Siehe Abschnitt 12.7 [Tastaturbelegung], Seite 276, für mehr Informationen, wie die Tastaturbelegung angegeben werden kann.

`extra-config` (Vorgabe: `'()`)
: Dies ist eine Liste von Zeichenketten oder Objekten, die an die Konfigurationsdatei angehängt werden. Mit ihnen wird zusätzlicher Text wortwörtlich zur Konfigurationsdatei hinzugefügt.

`server` (Vorgabe: `xorg-server`)
: Dieses Paket stellt den Xorg-Server zur Verfügung.

`server-arguments` (Vorgabe: `%default-xorg-server-arguments`)
: Dies ist die Liste der Befehlszeilenargumente, die an den X-Server übergeben werden. Die Vorgabe ist `-nolisten tcp`.

`set-xorg-configuration` *Konfiguration* [*login-manager-service-type*] [Scheme-Prozedur]
: *Benennt, welche* `Konfiguration` *die* Anmeldeverwaltung (vom Typ *login-manager-service-type*) benutzen soll, als ein `<xorg-configuration>`-Verbundsobjekt.

  Da die Xorg-Konfiguration in die Konfiguration der Anmeldeverwaltung eingebettet ist – z.B. in einer `gdm-configuration` –, bietet diese Prozedur eine Kurzschreibweise zum Ändern der Xorg-Konfiguration.

`xorg-start-command` [*Konfiguration*] [Scheme-Prozedur]
: Hier wird ein `startx`-Skript geliefert, in welchem die Module, Schriftarten usw. verfügbar sind, die in der *Konfiguration* angegeben wurden. Das Ergebnis soll anstelle von `startx` benutzt werden.

  Normalerweise wird der X-Server von der Anmeldeverwaltung gestartet.

`screen-locker-service` *Paket* [*Programm*] [Scheme-Prozedur]
: Das *Paket* zur Menge der setuid-Programme hinzufügen, worin sich ein Programm zum Sperren des Bildschirms oder ein Bildschirmschoner befinden muss, der mit dem Befehl *Programm* gestartet wird, und einen PAM-Eintrag dafür hinzufügen. Zum Beispiel macht

  ```
  (screen-locker-service xlockmore "xlock")
  ```

  das gute alte XlockMore benutzbar.

12.9.8 Druckdienste

Das Modul `(gnu services cups)` stellt eine Guix-Dienstdefinition für den CUPS-Druckdienst zur Verfügung. Wenn Sie Druckerunterstützung zu einem Guix-System hinzufügen möchten, dann fügen Sie einen `cups-service`-Dienst in die Betriebssystemdefinition ein.

`cups-service-type` [Scheme-Variable]
: Der Diensttyp für den CUPS-Druckserver. Als Wert muss eine gültige CUPS-Konfiguration angegeben werden (siehe unten). Um die Voreinstellungen zu verwenden, schreiben Sie einfach nur:

  ```
  (service cups-service-type)
  ```

Mit der CUPS-Konfiguration stellen Sie die grundlegenden Merkmale Ihrer CUPS-Installation ein: Auf welcher Schnittstelle sie lauscht, wie mit einem fehlgeschlagenen Druckauftrag umzugehen ist, wie viel in Protokolle geschrieben werden soll und so weiter. Um einen Drucker hinzuzufügen, müssen Sie jedoch die URL `http://localhost:631` besuchen oder ein Werkzeug wie die Druckereinstellungsdienste von GNOME benutzen. Die Vorgabe ist, dass beim Konfigurieren eines CUPS-Dienstes ein selbstsigniertes Zertifikat erzeugt wird, um sichere Verbindungen mit dem Druckserver zu ermöglichen.

Nehmen wir an, Sie wollen die Weboberfläche von CUPS aktivieren und Unterstützung für Epson-Drucker über das `epson-inkjet-printer-escpr`-Paket und für HP-Drucker über

das `hplip-minimal`-Paket aktivieren. Sie können das auf diese Weise gleich machen (dazu müssen Sie vorher angeben, dass das Modul `(gnu packages cups)` benutzt werden soll):

```
(service cups-service-type
         (cups-configuration
           (web-interface? #t)
           (extensions
             (list cups-filters epson-inkjet-printer-escpr hplip-minimal))))
```

Anmerkung: Wenn Sie die Qt5-basierte grafische Benutzeroberfläche benutzen möchten, die dem hplip-Paket beiliegt, sollten Sie das `hplip`-Paket installieren, entweder in die Konfigurationsdatei Ihres Betriebssystems oder für Ihr Benutzerkonto.

Im Folgenden sehen Sie die verfügbaren Konfigurationsparameter. Vor jeder Parameterdefinition wird ihr Typ angegeben, zum Beispiel steht '`Zeichenketten-Liste foo`' für einen Parameter `foo`, der als Liste von Zeichenketten angegeben werden muss. Sie können die Konfiguration aber auch in einer einzigen Zeichenkette angeben, wenn Sie eine alte `cupsd.conf`-Datei von einem anderen System weiterbenutzen möchten; siehe das Abschnittsende für mehr Details.

Die verfügbaren `cups-configuration`-Felder sind:

`„package“ cups` [`cups-configuration`-Parameter]
Das CUPS-Paket.

`„package“-Liste extensions` (*Vorgabe:* (`list brlaser cups-filters epson-inkjet-printer-escpr foomatic-filters hplip-minimal splix`)) [`cups-configuration`-Parameter]
Treiber und andere Erweiterungen für das CUPS-Paket.

`„files-configuration“ files-configuration` [`cups-configuration`-Parameter]
Konfiguration, wo Protokolle abgelegt werden sollen, welche Verzeichnisse für Druckspoolerwarteschlangen benutzt werden sollen und ähnliche Berechtigungen erfordernde Konfigurationsparameter.

Verfügbare Felder einer `files-configuration` sind:

`Protokollpfad access-log` [`files-configuration`-Parameter]
Hiermit wird der Dateiname des Zugriffsprotokolls („Access Log“) festgelegt. Wenn ein leerer Name angegeben wird, wird kein Protokoll erzeugt. Der Wert `stderr` lässt Protokolleinträge in die Standardfehlerdatei schreiben, wenn das Druckplanungsprogramm im Vordergrund läuft, oder an den Systemprotokolldaemon (also Syslog), wenn es im Hintergrund läuft. Der Wert `syslog` bewirkt, dass Protokolleinträge an den Systemprotokolldaemon geschickt werden. Der Servername darf in Dateinamen als die Zeichenkette `%s` angegeben werden, so kann etwa `/var/log/cups/%s-access_log` angegeben werden.

Die Vorgabe ist '`"/var/log/cups/access_log"`'.

`Dateiname cache-dir` [`files-configuration`-Parameter]
Wo CUPS zwischengespeicherte Daten ablegen soll.

Die Vorgabe ist '`"/var/cache/cups"`'.

`Zeichenkette config-file-perm` [`files-configuration`-Parameter]
Gibt die Berechtigungen für alle Konfigurationsdateien an, die das Planungsprogramm schreibt.

Beachten Sie, dass auf die Berechtigungen der Datei printers.conf eine Maske gelegt wird, wodurch Zugriffe nur durch das planende Benutzerkonto erlaubt werden (in der Regel der Administratornutzer „root“). Der Grund dafür ist, dass Druckergeräte-URIs manchmal sensible Authentisierungsdaten enthalten, die nicht allgemein auf dem System bekannt sein sollten. Es gibt keine Möglichkeit, diese Sicherheitsmaßnahme abzuschalten.

Die Vorgabe ist ‘`"0640"`’.

`Protokollpfad error-log` [`files-configuration`-Parameter]
Hiermit wird der Dateiname des Fehlerprotokolls („Error Log“) festgelegt. Wenn ein leerer Name angegeben wird, wird kein Fehlerprotokoll erzeugt. Der Wert `stderr` lässt Protokolleinträge in die Standardfehlerdatei schreiben, wenn das Planungsprogramm im Vordergrund läuft, oder an den Systemprotokolldaemon (also Syslog), wenn es im Hintergrund läuft. Der Wert `syslog` bewirkt, dass Protokolleinträge an den Systemprotokolldaemon geschickt werden. Der Servername darf in Dateinamen als die Zeichenkette `%s` angegeben werden, so kann etwa `/var/log/cups/%s-error_log` angegeben werden.

Die Vorgabe ist ‘`"/var/log/cups/error_log"`’.

`Zeichenkette fatal-errors` [`files-configuration`-Parameter]
Gibt an, bei welchen Fehlern das Druckplanungsprogramm terminieren soll. Die Zeichenketten für die Arten sind:

`none` Keine Fehler führen zur Beendigung.

`all` Jeder der im Folgenden aufgeführten Fehler terminiert den Druckplaner.

`browse` Fehler bei der Suche während der Initialisierung terminieren den Druckplaner, zum Beispiel wenn keine Verbindung zum DNS-SD-Daemon aufgebaut werden kann.

`config` Syntaxfehler in der Konfigurationsdatei terminieren den Druckplaner.

`listen` Fehler beim Lauschen oder Portfehler (entsprechend der Direktiven „Listen“ oder „Port“) terminieren den Druckplaner; ausgenommen sind Fehler bei IPv6 auf Loopback- oder `any`-Adressen.

`log` Fehler beim Erzeugen von Protokolldateien terminieren den Druckplaner.

`permissions`
Falsche Zugriffsberechtigungen auf zum Starten benötigten Dateien terminieren den Druckplaner, zum Beispiel wenn auf gemeinsame TLS-Zertifikats- und Schlüsseldateien von allen lesend zugegriffen werden kann.

Die Vorgabe ist ‘`"all -browse"`’.

`Boolescher-Ausdruck file-device?` [`files-configuration`-Parameter]
Gibt an, welches Pseudogerät für neue Druckerwarteschlangen benutzt werden kann. Die URI `file:///dev/null` wird immer zugelassen.

Vorgegeben ist '`#f`'.

`Zeichenkette group` [`files-configuration`-Parameter]
Gibt Namen oder Identifikator der Benutzergruppe an, die zum Ausführen von externen Programmen verwendet wird.

Die Vorgabe ist '`"lp"`'.

`Zeichenkette log-file-group` [`files-configuration`-Parameter]
Gibt Namen oder Identifikator der Benutzergruppe an, mit der Protokolldateien gespeichert werden.

Die Vorgabe ist '`"lpadmin"`'.

`Zeichenkette log-file-perm` [`files-configuration`-Parameter]
Gibt die Berechtigungen für alle Protokolldateien an, die das Planungsprogramm schreibt.

Die Vorgabe ist '`"0644"`'.

`log-location page-log` [`files-configuration`-Parameter]
Hiermit wird der Dateiname des Seitenprotokolls („Page Log“) festgelegt. Wenn ein leerer Name angegeben wird, wird kein Seitenprotokoll erzeugt. Der Wert `stderr` lässt Protokolleinträge in die Standardfehlerdatei schreiben, wenn das Planungsprogramm im Vordergrund läuft, oder an den Systemprotokolldaemon (also Syslog), wenn es im Hintergrund läuft. Der Wert `syslog` bewirkt, dass Protokolleinträge an den Systemprotokolldaemon geschickt werden. Der Servername darf in Dateinamen als die Zeichenkette `%s` angegeben werden, so kann etwa `/var/log/cups/%s-page_log` angegeben werden.

Die Vorgabe ist '`"/var/log/cups/page_log"`'.

`Zeichenkette remote-root` [`files-configuration`-Parameter]
Gibt den Benutzernamen an, der für unauthentifizierte Zugriffe durch Clients verwendet wird, die sich als der Administratornutzer „root“ anmelden. Vorgegeben ist `remroot`.

Die Vorgabe ist '`"remroot"`'.

`Dateiname request-root` [`files-configuration`-Parameter]
Gibt das Verzeichnis an, in dem Druckaufträge und andere Daten zu HTTP-Anfragen abgelegt werden.

Die Vorgabe ist '`"/var/spool/cups"`'.

`Isolierung sandboxing` [`files-configuration`-Parameter]
Gibt die Stufe der Sicherheitsisolierung („Sandboxing“) an, die auf Druckfilter, Hintergrundsysteme (Backends) und andere Kindprozesse des Planungsprogramms angewandt wird; entweder `relaxed` („locker“) oder `strict` („strikt“). Diese Direktive wird zurzeit nur auf macOS benutzt/unterstützt.

Die Vorgabe ist '`strict`'.

`Dateiname server-keychain` [`files-configuration`-Parameter]
Gibt an, wo TLS-Zertifikate und private Schlüssel gespeichert sind. CUPS wird in diesem Verzeichnis öffentliche und private Schlüssel suchen: `.crt`-Dateien für PEM-kodierte Zertifikate und zugehörige `.key`-Dateien für PEM-kodierte private Schlüssel.

Die Vorgabe ist '`"/etc/cups/ssl"`'.

`Dateiname server-root` [`files-configuration`-Parameter]
Gibt das Verzeichnis an, das die Serverkonfigurationsdateien enthält.

Die Vorgabe ist '`"/etc/cups"`'.

`Boolescher-Ausdruck sync-on-close?` [`files-configuration`-Parameter]
Gibt an, ob das Planungsprogramm fsync(2) aufrufen soll, nachdem es in Konfigurations- oder Zustandsdateien geschrieben hat.

Vorgegeben ist '`#f`'.

`Leerzeichengetrennte-Zeichenketten-Liste system-group` [`files-configuration`-Parameter]
Gibt die Benutzergruppe(n) an, die als die `@SYSTEM`-Gruppen für die Authentisierung benutzt werden können.

`Dateiname temp-dir` [`files-configuration`-Parameter]
Gibt das Verzeichnis an, in das temporäre Dateien gespeichert werden.

Die Vorgabe ist '`"/var/spool/cups/tmp"`'.

`Zeichenkette user` [`files-configuration`-Parameter]
Gibt den Benutzernamen oder -identifikator an, mit dessen Benutzerkonto externe Programme ausgeführt werden.

Die Vorgabe ist '`"lp"`'.

`Zeichenkette set-env` [`files-configuration`-Parameter]
Legt die angegebene Umgebungsvariable auf einen Wert (englisch „Value“) fest, die an Kindprozesse übergeben wird.

Die Vorgabe ist '`"variable value"`'.

`Zugriffsprotokollstufe access-log-level` [`cups-configuration`-Parameter]
Gibt an, mit welcher Detailstufe das Protokoll in der AccessLog-Datei geführt wird. Bei der Stufe `config` wird protokolliert, wenn Drucker und Klassen hinzugefügt, entfernt oder verändert werden, und wenn auf Konfigurationsdateien zugegriffen oder sie aktualisiert werden. Bei der Stufe `actions` wird protokolliert, wenn Druckaufträge eingereicht, gehalten, freigegeben, geändert oder abgebrochen werden sowie alles, was bei `config` Protokollierung auslöst. Bei der Stufe `all` wird jede Anfrage protokolliert.

Die Vorgabe ist '`actions`'.

`Boolescher-Ausdruck auto-purge-jobs?` [`cups-configuration`-Parameter]
Gibt an, ob Daten über den Auftragsverlauf automatisch gelöscht werden sollen, wenn Sie nicht mehr zur Berechnung von Druckkontingenten benötigt werden.

Vorgegeben ist '`#f`'.

`Kommagetrennte-Zeichenketten-Liste` [`cups-configuration`-Parameter]
`browse-dns-sd-sub-types`

Gibt eine Liste von DNS-SD-Subtypen an, die anderen für jeden geteilten Drucker mitgeteilt werden sollen. Zum Beispiel wird bei '`"_cups" "_print"`' den Netzwerk-Clients mitgeteilt, dass sowohl Teilen zwischen CUPS als auch IPP Everywhere unterstützt werden.

Die Vorgabe ist '`"_cups"`'.

`Protokolle-zur-lokalen-Suche` [`cups-configuration`-Parameter]
`browse-local-protocols`

Gibt an, welche Protokolle zum lokalen Teilen („Freigeben“) von Druckern benutzt werden sollen.

Die Vorgabe ist '`dnssd`'.

`Boolescher-Ausdruck browse-web-if?` [`cups-configuration`-Parameter]

Gibt an, ob die Weboberfläche von CUPS anderen mitgeteilt wird.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck browsing?` [`cups-configuration`-Parameter]

Gibt an, ob geteilte Drucker bei Druckersuchen mitgeteilt werden.

Vorgegeben ist '`#f`'.

`Zeichenkette classification` [`cups-configuration`-Parameter]

Gibt die Geheimhaltungsstufe des Servers an. Jeder gültige Deckblattname („Banner“-Name) kann benutzt werden, einschließlich '`"classified"`', '`"confidential"`', '`"secret"`', '`"topsecret"`' und '`"unclassified"`'. Wird kein Deckblatt angegeben, werden Funktionen für sicheres Drucken abgeschaltet.

Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck classify-override?` [`cups-configuration`-Parameter]

Gibt an, ob Nutzer bei einzelnen Druckaufträgen eine andere als die voreingestellte Geheimhaltungsstufe (für das Deckblatt) vorgeben können, indem sie die Option `job-sheets` einstellen.

Vorgegeben ist '`#f`'.

`Voreingestelle-Authentifizierungsart` [`cups-configuration`-Parameter]
`default-auth-type`

Gibt an, wie man nach Voreinstellung authentifiziert wird.

Die Vorgabe ist '`Basic`'.

`Voreingestellte-Verschlüsselung` [`cups-configuration`-Parameter]
`default-encryption`

Gibt an, ob für authentifizierte Anfragen Verschlüsselung benutzt wird.

Die Vorgabe ist '`Required`'.

`Zeichenkette default-language` [`cups-configuration`-Parameter]

Gibt an, welche Sprache für Text und Weboberfläche voreingestellt benutzt werden soll.

Die Vorgabe ist '`"en"`'.

`Zeichenkette default-paper-size` [`cups-configuration`-Parameter]
Gibt das voreingestellte Papierformat für neue Druckwarteschlangen an. Bei '`"Auto"`' wird eine der Locale entsprechende Voreinstellung gewählt, während bei '`"None"`' kein Papierformat voreingestellt ist. Verfügbare Formatbezeichnungen sind typischerweise '`"Letter"`' oder '`"A4"`'.

Die Vorgabe ist '`"Auto"`'.

`Zeichenkette default-policy` [`cups-configuration`-Parameter]
Gibt die voreingestellte Zugriffsrichtlinie an, die benutzt werden soll.

Die Vorgabe ist '`"default"`'.

`Boolescher-Ausdruck default-shared?` [`cups-configuration`-Parameter]
Gibt an, ob lokale Drucker nach Voreinstellung geteilt werden sollen.

Die Vorgabe ist '`#t`'.

`Nichtnegative-ganze-Zahl` [`cups-configuration`-Parameter]
`dirty-clean-interval`
Gibt an, mit welcher Verzögerung Konfigurations- und Zustandsdateien aktualisiert werden sollen. Der Wert 0 lässt die Aktualisierung so bald wie möglich stattfinden, in der Regel nach ein paar Millisekunden.

Die Vorgabe ist '`30`'.

`Fehlerrichtlinie error-policy` [`cups-configuration`-Parameter]
Gibt an, wie beim Auftreten eines Fehlers verfahren werden soll. Mögliche Werte sind `abort-job`, wodurch der fehlgeschlagene Druckauftrag verworfen wird, `retry-job`, wodurch der Druckauftrag später erneut versucht wird, `retry-current-job`, wodurch der fehlgeschlagene Druckauftrag sofort erneut versucht wird, und `stop-printer`, wodurch der Drucker angehalten wird.

Die Vorgabe ist '`stop-printer`'.

`Nichtnegative-ganze-Zahl filter-limit` [`cups-configuration`-Parameter]
Gibt die Maximalkosten von Filtern an, die nebenläufig ausgeführt werden, wodurch Probleme durch Platten-, Arbeitsspeicher- und Prozessorressourcennutzung minimiert werden können. Eine Beschränkung von 0 deaktiviert die Filterbeschränkung. Ein durchschnittlicher Druck mit einem Nicht-PostScript-Drucker erfordert eine Filterbeschränkung von mindestens ungefähr 200. Ein PostScript-Drucker erfordert eine halb so hohe Filterbeschränkung (100). Wird die Beschränkung unterhalb dieser Schwellwerte angesetzt, kann das Planungsprogramm effektiv nur noch einen einzelnen Druckauftrag gleichzeitig abarbeiten.

Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl filter-nice` [`cups-configuration`-Parameter]
Gibt die Planungspriorität von Filtern an, die zum Drucken eines Druckauftrags ausgeführt werden. Der nice-Wert kann zwischen 0, der höchsten Priorität, und 19, der niedrigsten Priorität, liegen.

Die Vorgabe ist '`0`'.

`Rechnernamensauflösungen` [`cups-configuration`-Parameter]
`host-name-lookups`

Gibt an, ob inverse Namensauflösungen („Reverse Lookups“) bei sich verbindenden Clients durchgeführt werden sollen. Die Einstellung `double` lässt `cupsd` verifizieren, dass der anhand der Adresse aufgelöste Rechnername zu einer der für den Rechnernamen zurückgelieferten Adressen passt. „Double“-Namensauflösungen verhindern auch, dass sich Clients mit unregistrierten Adressen mit Ihrem Server verbinden können. Setzen Sie diese Option nur dann auf `#t` oder `double`, wenn es unbedingt notwendig ist.

Vorgegeben ist '`#f`'.

`Nichtnegative-ganze-Zahl job-kill-delay` [`cups-configuration`-Parameter]

Gibt die Anzahl an Sekunden an, wie lange vor dem Abwürgen der mit einem abgebrochenen oder gehaltenen Druckauftrag assoziierten Filter- und Hintergrundprozesse (dem „Backend“) gewartet wird.

Die Vorgabe ist '`30`'.

`Nichtnegative-ganze-Zahl` [`cups-configuration`-Parameter]
`job-retry-interval`

Gibt das Zeitintervall zwischen erneuten Versuchen von Druckaufträgen in Sekunden an. Dies wird in der Regel für Fax-Warteschlangen benutzt, kann aber auch für normale Druckwarteschlangen benutzt werden, deren Fehlerrichtlinie `retry-job` oder `retry-current-job` ist.

Die Vorgabe ist '`30`'.

`Nichtnegative-ganze-Zahl job-retry-limit` [`cups-configuration`-Parameter]

Gibt die Anzahl an, wie oft ein Druckauftrag erneut versucht wird. Dies wird in der Regel für Fax-Warteschlangen benutzt, kann aber auch für normale Druckwarteschlangen benutzt werden, deren Fehlerrichtlinie `retry-job` oder `retry-current-job` ist.

Die Vorgabe ist '`5`'.

`Boolescher-Ausdruck keep-alive?` [`cups-configuration`-Parameter]

Gibt an, ob HTTP-„keep-alive“ für Verbindungen unterstützt wird.

Die Vorgabe ist '`#t`'.

`Nichtnegative-ganze-Zahl` [`cups-configuration`-Parameter]
`limit-request-body`

Gibt die Maximalgröße von zu druckenden Dateien, IPP-Anfragen und HTML-Formulardaten an. Eine Beschränkung von 0 deaktiviert die Beschränkung.

Die Vorgabe ist '`0`'.

`Mehrzeilige-Zeichenketten-Liste listen` [`cups-configuration`-Parameter]

Lauscht auf den angegebenen Schnittstellen auf Verbindungen. Gültige Werte haben die Form *Adresse*:*Port*, wobei die *Adresse* entweder eine von eckigen Klammern umschlossene IPv6-Adresse, eine IPv4-Adresse oder `*` ist; letztere steht für alle Adressen. Werte können auch Dateinamen lokaler UNIX-Sockets sein. Die Listen-Direktive ähnelt der Port-Direktive, macht es aber möglich, den Zugriff für bestimmte Schnittstellen oder Netzwerke einzuschränken.

`Nichtnegative-ganze-Zahl listen-back-log` [`cups-configuration`-Parameter]
Gibt die Anzahl ausstehender Verbindungen an, die möglich sein soll. Normalerweise betrifft dies nur sehr ausgelastete Server, die die MaxClients-Beschränkung erreicht haben. Es kann aber auch eine Rolle spielen, wenn versucht wird, gleichzeitig sehr viele Verbindungen herzustellen. Wenn die Beschränkung erreicht wird, sperrt das Betriebssystem den Aufbau weiterer Verbindungen, bis das Druckplanungsprogramm die ausstehenden akzeptieren konnte.

Die Vorgabe ist '`128`'.

`„location-access-controls“-Liste location-access-controls` [`cups-configuration`-Parameter]
Gibt eine Liste zusätzlicher Zugriffssteuerungen („Access Controls“) an.

Verfügbare `location-access-controls`-Felder sind:

`Dateiname path` [`location-access-controls`-Parameter]
Gibt den URI-Pfad an, für den die Zugriffssteuerung gilt.

`Zugriffssteuerungs-Liste access-controls` [`location-access-controls`-Parameter]
Zugriffssteuerungen für jeden Zugriff auf diesen Pfad, im selben Format wie die `access-controls` bei `operation-access-control`.

Die Vorgabe ist '`()`'.

`„method-access-controls“-Liste method-access-controls` [`location-access-controls`-Parameter]
Zugriffssteuerungen für methodenspezifische Zugriffe auf diesen Pfad.

Die Vorgabe ist '`()`'.

Verfügbare `method-access-controls`-Felder sind:

`Boolescher-Ausdruck reverse?` [`method-access-controls`-Parameter]
Falls dies `#t` ist, gelten die Zugriffssteuerungen für alle Methoden außer den aufgelisteten Methoden. Andernfalls gelten sie nur für die aufgelisteten Methoden.

Vorgegeben ist '`#f`'.

`Methoden-Liste methods` [`method-access-controls`-Parameter]
Methoden, für die diese Zugriffssteuerung gilt.

Die Vorgabe ist '`()`'.

`Zugriffssteuerungs-Liste access-controls` [`method-access-controls`-Parameter]
Zugriffssteuerungsdirektiven als eine Liste von Zeichenketten. Jede Zeichenkette steht für eine Direktive wie z.B. '`"Order allow,deny"`'.

Die Vorgabe ist '`()`'.

`Nichtnegative-ganze-Zahl log-debug-history` [`cups-configuration`-Parameter]
: Gibt die Anzahl der Nachrichten zur Fehlersuche an, die für Protokolle vorgehalten werden, wenn ein Fehler in einem Druckauftrag auftritt. Nachrichten zur Fehlersuche werden unabhängig von der LogLevel-Einstellung protokolliert.

  Die Vorgabe ist '`100`'.

`Protokollstufe log-level` [`cups-configuration`-Parameter]
: Gibt die Stufe der Protokollierung in die ErrorLog-Datei an. Der Wert `none` stoppt alle Protokollierung, während `debug2` alles protokollieren lässt.

  Die Vorgabe ist '`info`'.

`Protokollzeitformat log-time-format` [`cups-configuration`-Parameter]
: Gibt das Format von Datum und Uhrzeit in Protokolldateien an. Der Wert `standard` lässt ganze Sekunden ins Protokoll schreiben, während bei `usecs` Mikrosekunden protokolliert werden.

  Die Vorgabe ist '`standard`'.

`Nichtnegative-ganze-Zahl max-clients` [`cups-configuration`-Parameter]
: Gibt die Maximalzahl gleichzeitig bedienter Clients an, die vom Planer zugelassen werden.

  Die Vorgabe ist '`100`'.

`Nichtnegative-ganze-Zahl max-clients-per-host` [`cups-configuration`-Parameter]
: Gibt die Maximalzahl gleichzeitiger Clients von derselben Adresse an, die zugelassen werden.

  Die Vorgabe ist '`100`'.

`Nichtnegative-ganze-Zahl max-copies` [`cups-configuration`-Parameter]
: Gibt die Maximalzahl der Kopien an, die ein Nutzer vom selben Druckauftrag ausdrucken lassen kann.

  Die Vorgabe ist '`9999`'.

`Nichtnegative-ganze-Zahl max-hold-time` [`cups-configuration`-Parameter]
: Gibt die maximale Zeitdauer an, die ein Druckauftrag im Haltezustand `indefinite` bleiben darf, bevor er abgebrochen wird. Ein Wert von 0 deaktiviert ein Abbrechen gehaltener Druckaufträge.

  Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl max-jobs` [`cups-configuration`-Parameter]
: Gibt die Maximalzahl gleichzeitiger Druckaufträge an, die noch zugelassen wird. Wenn Sie es auf 0 setzen, wird die Anzahl Druckaufträge *nicht* beschränkt.

  Die Vorgabe ist '`500`'.

`Nichtnegative-ganze-Zahl max-jobs-per-printer` [`cups-configuration`-Parameter]
: Gibt die Maximalzahl gleichzeitiger Druckaufträge an, die pro Drucker zugelassen werden. Ein Wert von 0 lässt bis zu MaxJobs-viele Druckaufträge pro Drucker zu.

  Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl max-jobs-per-user` [`cups-configuration`-Parameter]

Gibt die Maximalzahl gleichzeitiger Druckaufträge an, die pro Benutzer zugelassen werden. Ein Wert von 0 lässt bis zu MaxJobs-viele Druckaufträge pro Benutzer zu.

Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl max-job-time` [`cups-configuration`-Parameter]

Gibt die maximale Anzahl an, wie oft das Drucken eines Druckauftrags versucht werden kann, bis er abgebrochen wird, in Sekunden. Wenn Sie es auf 0 setzen, wird das Abbrechen „feststeckender" Druckaufträge deaktiviert.

Die Vorgabe ist '`10800`'.

`Nichtnegative-ganze-Zahl max-log-size` [`cups-configuration`-Parameter]

Gibt die Maximalgröße der Protokolldateien an, bevor sie rotiert werden, in Bytes. Beim Wert 0 wird Protokollrotation deaktiviert.

Die Vorgabe ist '`1048576`'.

`Nichtnegative-ganze-Zahl multiple-operation-timeout` [`cups-configuration`-Parameter]

Gibt die maximale Zeitdauer an, wie lange es zwischen Dateien in einem Druckauftrag mit mehreren Dateien dauern darf, in Sekunden.

Die Vorgabe ist '`900`'.

`Zeichenkette page-log-format` [`cups-configuration`-Parameter]

Gibt das Format der Zeilen im PageLog-Seitenprotokoll an. Folgen, die mit Prozentzeichen ('`%`') beginnen, werden durch die jeweils zugehörigen Informationen ersetzt, während alle anderen Zeichen wortwörtlich übernommen werden. Die folgenden Prozentfolgen werden erkannt:

'`%%`' ein einzelnes Prozentzeichen einfügen

'`%{name}`' den Wert des angegebenen IPP-Attributs einfügen

'`%C`' die Anzahl der Kopien der aktuellen Seite einfügen

'`%P`' die aktuelle Seitenzahl einfügen

'`%T`' das aktuelle Datum und Uhrzeit im allgemeinen Protokollformat einfügen

'`%j`' den Druckauftragsidentifikator („Job-ID") einfügen

'`%p`' den Druckernamen einfügen

'`%u`' den Benutzernamen einfügen

Wird die leere Zeichenkette als Wert angegeben, wird Seitenprotokollierung abgeschaltet. Die Zeichenkette `%p %u %j %T %P %C %{job-billing} %{job-originating-host-name} %{job-name} %{media} %{sides}` erzeugt ein Seitenprotokoll mit den üblichen Elementen.

Die Vorgabe ist '`""`'.

`Umgebungsvariable environment-variables` [`cups-configuration`-Parameter]
Übergibt die angegebene(n) Umgebungsvariable(n) an Kindprozesse, als Liste von Zeichenketten.

Die Vorgabe ist '`()`'.

`„policy-configuration“-Liste policies` [`cups-configuration`-Parameter]
Gibt die benannten Zugriffssteuerungsrichtlinien an.

Verfügbare `policy-configuration`-Felder sind:

`Zeichenkette name` [`policy-configuration`-Parameter]
Der Name der Richtlinie.

`Zeichenkette job-private-access` [`policy-configuration`-Parameter]
Gibt eine Zugriffsliste der privaten Werte eines Druckauftrags an. `@ACL` wird auf die Werte von requesting-user-name-allowed oder requesting-user-name-denied des Druckers abgebildet. `@OWNER` wird auf den Besitzer des Druckauftrags abgebildet. `@SYSTEM` wird auf die Gruppen abgebildet, die im `system-group`-Feld der `files-configuration` aufgelistet sind, aus der die Datei `cups-files.conf(5)` erzeugt wird. Zu den anderen möglichen Elementen der Zugriffsliste gehören Namen bestimmer Benutzerkonten und `@`*`Benutzergruppe`*, was für Mitglieder einer bestimmten Benutzergruppe steht. Die Zugriffsliste kann auch einfach als `all` oder `default` festgelegt werden.

Die Vorgabe ist '`"@OWNER @SYSTEM"`'.

`Zeichenkette job-private-values` [`policy-configuration`-Parameter]
Gibt die Liste der Druckauftragswerte an, die geschützt werden sollten, oder `all`, `default` oder `none`.

Die Vorgabe ist '`"job-name job-originating-host-name job-originating-user-name phone"`'.

`Zeichenkette subscription-private-access` [`policy-configuration`-Parameter]
Gibt eine Zugriffsliste für die privaten Werte eines Abonnements an. `@ACL` wird auf die Werte von requesting-user-name-allowed oder requesting-user-name-denied des Druckers abgebildet. `@OWNER` wird auf den Besitzer des Druckauftrags abgebildet. `@SYSTEM` wird auf die Gruppen abgebildet, die im `system-group`-Feld der `files-configuration` aufgelistet sind, aus der die Datei `cups-files.conf(5)` erzeugt wird. Zu den anderen möglichen Elementen der Zugriffsliste gehören Namen bestimmer Benutzerkonten und `@`*`Benutzergruppe`*, was für Mitglieder einer bestimmten Benutzergruppe steht. Die Zugriffsliste kann auch einfach als `all` oder `default` festgelegt werden.

Die Vorgabe ist '`"@OWNER @SYSTEM"`'.

`Zeichenkette subscription-private-values` [`policy-configuration`-Parameter]
Gibt die Liste der Druckauftragswerte an, die geschützt werden sollten, oder `all`, `default` oder `none`.

Die Vorgabe ist '`"notify-events notify-pull-method notify-recipient-uri notify-subscriber-user-name notify-user-data"`'.

`„operation-access-controls“-Liste` [`policy-configuration`-Parameter]
`access-controls`
Zugriffssteuerung durch IPP-Betrieb.
Die Vorgabe ist ‘`()`’.

`Boolescher-Ausdruck-oder-Nichtnegative-ganze-Zahl` [`cups-configuration`-Parameter]
`preserve-job-files`
Gibt an, ob die Dateien eines Druckauftrags (Dokumente) erhalten bleiben, nachdem ein Druckauftrag ausgedruckt wurde. Wenn eine Zahl angegeben wird, bleiben die Dateien des Druckauftrags für die angegebene Zahl von Sekunden nach dem Drucken erhalten. Ein boolescher Wert bestimmt ansonsten, ob sie auf unbestimmte Zeit erhalten bleiben.
Die Vorgabe ist ‘`86400`’.

`Boolescher-Ausdruck-oder-Nichtnegative-ganze-Zahl` [`cups-configuration`-Parameter]
`preserve-job-history`
Gibt an, ob der Druckauftragsverlauf nach dem Drucken eines Druckauftrags erhalten bleibt. Wenn eine Zahl angegeben wird, bleibt der Druckauftragsverlauf für die angegebene Zahl von Sekunden nach dem Drucken erhalten. Bei `#t` bleibt der Druckauftragsverlauf so lange erhalten, bis die MaxJobs-Beschränkung erreicht wurde.
Die Vorgabe ist ‘`#t`’.

`Nichtnegative-ganze-Zahl reload-timeout` [`cups-configuration`-Parameter]
Gibt an, wie lange vor dem Neustart des Planungsprogramms auf den Abschluss eines Druckauftrages gewartet wird.
Die Vorgabe ist ‘`30`’.

`Zeichenkette rip-cache` [`cups-configuration`-Parameter]
Gibt die maximale Menge an genutztem Arbeitsspeicher für das Konvertieren von Dokumenten in eine Rastergrafik (eine „Bitmap“) für einen Drucker an.
Die Vorgabe ist ‘`"128m"`’.

`Zeichenkette server-admin` [`cups-configuration`-Parameter]
Gibt die E-Mail-Adresse des Serveradministrators an.
Die Vorgabe ist ‘`"root@localhost.localdomain"`’.

`Rechnernamens-Liste-oder-* server-alias` [`cups-configuration`-Parameter]
Die ServerAlias-Direktive wird zur Prüfung der HTTP-Host-Kopfzeile benutzt, wenn sich Clients mit dem Planungsprogramm über externe Schnittstellen verbinden. Wenn der besondere Name `*` benutzt wird, könnte Ihr System möglicherweise bekannten browserbasierten DNS-Rebinding-Angriffen ausgesetzt werden, selbst wenn auf die Angebote nur über eine Firewall zugegriffen wird. Wenn alternative Namen nicht automatisch erkannt werden, empfehlen wir, jeden alternativen Namen in der ServerAlias-Direktive aufzulisten, statt `*` zu benutzen.
Die Vorgabe ist ‘`*`’.

`Zeichenkette server-name` [`cups-configuration`-Parameter]
Gibt den vollständigen Rechnernamen („Fully-Qualified Host Name“) des Servers an.
Die Vorgabe ist ‘`"localhost"`’.

`Server-Tokens server-tokens` [`cups-configuration`-Parameter]
Gibt an, welche Informationen in der Server-Kopfzeile von HTTP-Antworten vorkommen. `None` deaktiviert die Server-Kopfzeile. `ProductOnly` liefert `CUPS`. `Major` liefert `CUPS 2`. `Minor` liefert `CUPS 2.0`. `Minimal` liefert `CUPS 2.0.0`. `OS` liefert `CUPS 2.0.0 (`*uname*`)`, wobei *uname* die Ausgabe des Befehls `uname` ist. `Full` liefert `CUPS 2.0.0 (`*uname*`) IPP/2.0`.

Die Vorgabe ist '`Minimal`'.

`Mehrzeilige-Zeichenketten-Liste ssl-listen` [`cups-configuration`-Parameter]
Lauscht auf den angegebenen Schnittstellen auf verschlüsselte Verbindungen. Gültige Werte haben die Form *Adresse*:*Port*, wobei die *Adresse* entweder eine von eckigen Klammern umschlossene IPv6-Adresse, eine IPv4-Adresse oder `*` ist; letztere steht für alle Adressen.

Die Vorgabe ist '`()`'.

`SSL-Optionen ssl-options` [`cups-configuration`-Parameter]
Legt Verschlüsselungsoptionen fest. Nach Vorgabe unterstützt CUPS nur Verschlüsselung mit TLS v1.0 oder höher mit bekannten, sicheren „Cipher Suites“. Es ist weniger sicher, Optionen mit `Allow` („erlauben“) zu verwenden, und es erhöht die Sicherheit, Optionen mit `Deny` („verweigern“) zu benutzen. Die Option `AllowRC4` aktiviert die 128-Bit-RC4-Cipher-Suites, die manche alten Clients brauchen. Die Option `AllowSSL3` aktiviert SSL v3.0, das manche alte Clients brauchen, die TLS v1.0 nicht implementieren. Die Option `DenyCBC` deaktiviert alle CBC-Cipher-Suites. Die Option `DenyTLS1.0` deaktiviert Unterstützung für TLS v1.0 – dadurch wird TLS v1.1 zur kleinsten noch unterstützten Protokollversion.

Die Vorgabe ist '`()`'.

`Boolescher-Ausdruck strict-conformance?` [`cups-configuration`-Parameter]
Gibt an, ob das Druckplanungsprogramm von Clients fordert, dass sie sich strikt an die IPP-Spezifikationen halten.

Vorgegeben ist '`#f`'.

`Nichtnegative-ganze-Zahl timeout` [`cups-configuration`-Parameter]
Gibt die Zeitbeschränkung für HTTP-Anfragen an, in Sekunden.

Die Vorgabe ist '`900`'.

`Boolescher-Ausdruck web-interface?` [`cups-configuration`-Parameter]
Gibt an, ob die Weboberfläche zur Verfügung gestellt wird.

Vorgegeben ist '`#f`'.

Mittlerweile denken Sie wahrscheinlich: „Ich bitte dich, Guix-Handbuch, ich mag dich, aber kannst du jetzt bitte mit den Konfigurationsoptionen aufhören?“ Damit hätten Sie recht. Allerdings sollte ein weitere Punkt noch erwähnt werden: Vielleicht haben Sie eine bestehende `cupsd.conf`, die Sie verwenden möchten. In diesem Fall können Sie eine `opaque-cups-configuration` als die Konfiguration eines `cups-service-type` übergeben.

Verfügbare `opaque-cups-configuration`-Felder sind:

„package“ cups [opaque-cups-configuration-Parameter]
Das CUPS-Paket.

Zeichenkette cupsd.conf [opaque-cups-configuration-Parameter]
Der Inhalt der Datei cupsd.conf als eine Zeichenkette.

Zeichenkette cups-files.conf [opaque-cups-configuration-Parameter]
Der Inhalt der Datei cups-files.conf als eine Zeichenkette.

Wenn Ihre cupsd.conf und cups-files.conf zum Beispiel in Zeichenketten mit dem entsprechenden Namen definiert sind, könnten Sie auf diese Weise einen CUPS-Dienst instanziieren:

```
(service cups-service-type
         (opaque-cups-configuration
           (cupsd.conf cupsd.conf)
           (cups-files.conf cups-files.conf)))
```

12.9.9 Desktop-Dienste

Das Modul (gnu services desktop) stellt Dienste zur Verfügung, die meistens bei „Desktop“-Einrichtungen für grafische Nutzung praktisch sind – also auf einer Maschine mit einem grafischem Anzeigeserver, vielleicht mit einer grafischen Benutzeroberfläche, usw. Im Modul werden auch Dienste definiert, die bestimmte Arbeitsumgebungen wie GNOME, Xfce oder MATE bereitstellen.

Um es einfacher zu machen, definiert das Modul auch eine Variable mit denjenigen Diensten, die man auf einer Maschine mit einer grafischen Umgebung und Netzwerkunterstützung erwarten würde:

%desktop-services [Scheme-Variable]
Dies ist eine Liste von Diensten, die %base-services ergänzt und weitere Dienste hinzufügt oder bestehende anpasst, um für eine normale „Desktop“-Nutzung geeignet zu sein.

Insbesondere wird eine grafische Anmeldeverwaltung hinzugefügt (siehe Abschnitt 12.9.7 [X Window], Seite 342), ebenso Programme zur Bildschirmsperre, ein Werkzeug zur Netzwerkverwaltung (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314) mit Unterstützung für Modems (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314), Energieverbrauchs- und Farbverwaltungsdienste, Anmeldungs- und Sitzungsverwaltung über elogind, der Berechtigungsdienst Polkit, der Ortungsdienst GeoClue, der AccountsService-Daemon, mit dem autorisierte Benutzer Systempasswörter ändern können, ein NTP-Client (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314) und der Avahi-Daemon. Außerdem wird der Name Service Switch konfiguriert, damit er nss-mdns benutzt (siehe Abschnitt 12.12 [Name Service Switch], Seite 607).

Die %desktop-services-Variable kann als das services-Feld einer operating-system-Deklaration genutzt werden (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258).

Daneben können die Prozeduren gnome-desktop-service-type, xfce-desktop-service, mate-desktop-service-type, lxqt-desktop-service-type und

`enlightenment-desktop-service-type` jeweils GNOME, Xfce, MATE und/oder Enlightenment zu einem System hinzufügen. „GNOME hinzufügen“ bedeutet, dass Dienste auf Systemebene wie z.B. Hilfsprogramme zur Anpassung der Hintergrundbeleuchtung und des Energieverbrauchs zum System hinzugefügt werden und `polkit` und `dbus` entsprechend erweitert werden, wodurch GNOME mit erhöhten Berechtigungen auf eine begrenzte Zahl von speziellen Systemschnittstellen zugreifen kann. Zusätzlich bedeutet das Hinzufügen eines durch `gnome-desktop-service-type` erzeugten Dienstes, dass das GNOME-Metapaket ins Systemprofil eingefügt wird. Genauso wird beim Einfügen des Xfce-Dienstes nicht nur das `xfce`-Metapaket zum Systemprofil hinzugefügt, sondern dem Thunar-Dateiverwaltungsprogramm wird auch die Berechtigung gegeben, ein Fenster mit Administratorrechten zu öffnen, wenn der Benutzer sich mit dem Administratorpasswort über die standardmäßige grafische Oberfläche von Polkit authentisiert. „MATE hinzufügen“ bedeutet, dass `polkit` und `dbus` entsprechend erweitert werden, wodurch MATE mit erhöhten Berechtigungen auf eine begrenzte Zahl von speziellen Systemschnittstellen zugreifen kann. Zusätzlich bedeutet das Hinzufügen eines durch `mate-desktop-service-type` erzeugten Dienstes, dass das MATE-Metapaket ins Systemprofil eingefügt wird. „Enlightenment hinzufügen“ bedeutet, dass `dbus` entsprechend erweitert wird und mehrere Binärdateien von Enlightenment als setuid eingerichtet werden, wodurch das Programm zum Sperren des Bildschirms und andere Funktionen von Enlightenment wie erwartet funktionieren.

Die Arbeitsumgebungen in Guix benutzen standardmäßig den Xorg-Anzeigeserver. Falls Sie das neuere Anzeigeserverprotokoll namens Wayland benutzen möchten, müssen Sie die Wayland-Unterstützung in GDM aktivieren (siehe [wayland-gdm], Seite 342). Alternativ können Sie den Dienst `sddm-service` anstelle von GDM für die grafische Anmeldeverwaltung einrichten. Dann sollten Sie in SDDM die Sitzung „GNOME (Wayland)“ auswählen. Alternativ können Sie auch versuchen, GNOME mit Wayland manuell aus einer Konsole (TTY) mit dem Befehl „XDG_SESSION_TYPE=wayland exec dbus-run-session gnome-session“ zu starten. Derzeit wird Wayland nur von GNOME unterstützt.

`gnome-desktop-service-type` [Scheme-Variable]

Dies ist der Typ des Dienstes, der die GNOME-Arbeitsumgebung (`https://www.gnome.org`) bereitstellt. Sein Wert ist ein `gnome-desktop-configuration`-Objekt (siehe unten).

Dieser Dienst fügt das `gnome`-Paket zum Systemprofil hinzu und erweitert Polkit um die von `gnome-settings-daemon` benötigten Aktionen.

`gnome-desktop-configuration` [Datentyp]

Verbundsobjekt für die Konfiguration der GNOME-Arbeitsumgebung.

`gnome` (Vorgabe: `gnome`)
: Welches GNOME-Paket benutzt werden soll.

`xfce-desktop-service-type` [Scheme-Variable]

Der Typ des Dienstes, um die Xfce-Arbeitsumgebung (`https://xfce.org/`) auszuführen. Sein Wert ist ein `xfce-desktop-configuration`-Objekt (siehe unten).

Dieser Dienst fügt das Paket `xfce` zum Systemprofil hinzu und erweitert Polkit, damit `thunar` befähigt wird, das Dateisystem aus einer Benutzersitzung heraus mit

Administratorrechten zu bearbeiten, nachdem sich der Benutzer mit dem Administratorpasswort authentisiert hat.

Bedenken Sie, dass `xfce4-panel` und seine Plugin-Pakete in dasselbe Profil installiert werden sollten, um sicherzugehen, dass sie kompatibel sind. Wenn Sie diesen Dienst benutzen, sollten Sie zusätzliche Plugins (`xfce4-whiskermenu-plugin`, `xfce4-weather-plugin` usw.) ins `packages`-Feld Ihres `operating-system` eintragen.

`xfce-desktop-configuration` [Datentyp]
Verbundstyp für Einstellungen zur Xfce-Arbeitsumgebung.

`xfce` (Vorgabe: `xfce`)
Das Xfce-Paket, was benutzt werden soll.

`mate-desktop-service-type` [Scheme-Variable]
Dies ist der Typ des Dienstes, um die MATE-Arbeitsumgebung (`https://mate-desktop.org/`) auszuführen. Sein Wert ist ein `mate-desktop-configuration`-Objekt (siehe unten).

Dieser Dienst fügt das Paket `mate` ins Systemprofil ein und erweitert Polkit um die Aktionen aus dem `mate-settings-daemon`.

`mate-desktop-configuration` [Datentyp]
Verbundstyp für die Einstellungen der MATE-Arbeitsumgebung.

`mate` (Vorgabe: `mate`)
Das MATE-Paket, was benutzt werden soll.

`lxqt-desktop-service-type` [Scheme-Variable]
Dies ist der Typ des Dienstes, um die LXQt-Arbeitsumgebung (`https://lxqt-project.org`) auszuführen. Sein Wert ist ein `lxqt-desktop-configuration`-Objekt (siehe unten).

Dieser Dienst fügt das Paket `lxqt` ins Systemprofil ein.

`lxqt-desktop-configuration` [Datentyp]
Verbundstyp für Einstellungen zur LXQt-Arbeitsumgebung.

`lxqt` (Vorgabe: `lxqt`)
Das LXQT-Paket, was benutzt werden soll.

`enlightenment-desktop-service-type` [Scheme-Variable]
Liefert einen Dienst, der das `enlightenment`-Paket zum Systemprofil hinzufügt und D-Bus mit den Aktionen aus `efl` erweitert.

`enlightenment-desktop-service-configuration` [Datentyp]
`enlightenment` (Vorgabe: `enlightenment`)
Das Enlightenment-Paket, was benutzt werden soll.

Weil die Desktopdienste GNOME, Xfce und MATE so viele Pakete ins System mitnehmen, gehören diese nicht zu den Vorgaben in der `%desktop-services`-Variablen. Um GNOME, Xfce oder MATE hinzuzufügen, benutzen Sie einfach `cons` zum Anhängen an die `%desktop-services` im `services`-Feld Ihrer `operating-system`-Deklaration:

```
(use-modules (gnu))
```

```
(use-service-modules desktop)
(operating-system
  ...
  ;; cons* adds items to the list given as its last argument.
  (services (cons* (service gnome-desktop-service-type)
                   (service xfce-desktop-service)
                   %desktop-services))
  ...)
```

Diese Arbeitsumgebungen stehen dann im grafischen Anmeldefenster zur Auswahl.

Die eigentlichen Dienstdefinitionen, die in `%desktop-services` stehen und durch `(gnu services dbus)` und `(gnu services desktop)` zur Verfügung gestellt werden, werden im Folgenden beschrieben.

`dbus-service` [*#:dbus* `dbus`] [*#:services* '()] [*#:verbose?*] [Scheme-Prozedur]
Liefert einen Dienst, der den „Systembus“ mit `dbus`

ausführt, mit Unterstützung für die als *services* übergebenen Dienste. Wenn *verbose?* auf wahr gesetzt ist, wird die Umgebungsvariable '`DBUS_VERBOSE`' für ausführliche Protokollierung auf '`1`' gesetzt. Damit das auch etwas bewirkt, muss für *dbus* ein D-Bus-Paket mit Unterstützung dafür angegeben werden, etwa `dbus-verbose`. Das ausführliche Protokoll finden Sie in `/var/log/dbus-daemon.log`.

D-Bus (`https://dbus.freedesktop.org/`) ist eine Einrichtung zur Interprozesskommunikation. Deren Systembus wird benutzt, damit Systemdienste miteinander kommunizieren können und damit sie bei systemweiten Ereignissen benachrichtigt werden können.

Als *services* muss eine Liste von Paketen übergeben werden, die ein Verzeichnis `etc/dbus-1/system.d` mit zusätzlichen D-Bus-Konfigurations- und Richtliniendateien enthalten. Damit zum Beispiel der Avahi-Daemon den Systembus benutzen kann, muss *services* gleich `(list avahi)` sein.

`elogind-service` [*#:config* `Konfiguration`] [Scheme-Prozedur]
Liefert einen Dienst, der den Anmelde- und Sitzungsdaemon `elogind` ausführt. Elogind (`https://github.com/elogind/elogind`) stellt eine D-Bus-Schnittstelle bereit, über die ausgelesen werden kann, welche Benutzer angemeldet sind und welche Sitzungen sie geöffnet haben, und außerdem das System in Bereitschaft versetzt werden kann, der Bereitschaftsmodus unterdrückt werden kann, das System neu gestartet werden kann und anderes.

Die meisten Energieereignisse auf Systemebene in einem Rechner werden von elogind behandelt, wie etwa ein Versetzen des Systems in Bereitschaft, wenn der Rechner zugeklappt wird, oder ein Herunterfahren beim Drücken des Stromschalters.

Das *config*-Schlüsselwort gibt die Konfiguration für elogind an und sollte das Ergebnis eines Aufrufs von `(elogind-configuration (`*Parameter Wert*`)...)` sein. Verfügbare Parameter und ihre Vorgabewerte sind:

```
kill-user-processes?
          #f

kill-only-users
          ()
```

```
kill-exclude-users
          ("root")

inhibit-delay-max-seconds
          5

handle-power-key
          poweroff

handle-suspend-key
          suspend

handle-hibernate-key
          hibernate

handle-lid-switch
          suspend

handle-lid-switch-docked
          ignore

handle-lid-switch-external-power
          *unspecified*

power-key-ignore-inhibited?
          #f

suspend-key-ignore-inhibited?
          #f

hibernate-key-ignore-inhibited?
          #f

lid-switch-ignore-inhibited?
          #t

holdoff-timeout-seconds
          30

idle-action
          ignore

idle-action-seconds
          (* 30 60)

runtime-directory-size-percent
          10

runtime-directory-size
          #f

remove-ipc?
          #t

suspend-state
          ("mem" "standby" "freeze")
```

```
suspend-mode
          ()

hibernate-state
          ("disk")

hibernate-mode
          ("platform" "shutdown")

hybrid-sleep-state
          ("disk")

hybrid-sleep-mode
          ("suspend" "platform" "shutdown")
```

`accountsservice-service` [*#:accountsservice* `accountsservice`] *Liefert einen Dienst, der* [Scheme-Prozedur]
AccountsService ausführt. Dabei handelt es sich um einen Systemdienst, mit dem verfügbare Benutzerkonten aufgelistet und deren Passwörter geändert werden können, und Ähnliches. AccountsService arbeitet mit PolicyKit zusammen, um es Benutzern ohne besondere Berechtigungen zu ermöglichen, ihre Systemkonfiguration zu ändern. Siehe den Webauftritt von AccountsService (`https://www.freedesktop.org/wiki/Software/AccountsService/`) für weitere Informationen.

Das Schlüsselwortargument *accountsservice* gibt das `accountsservice`-Paket an, das als Dienst verfügbar gemacht wird.

`polkit-service` [*#:polkit* `polkit`] *Liefert einen Dienst, der* [Scheme-Prozedur]
Polkit als Dienst zur Verwaltung von Berechtigungen (`https://www.freedesktop.org/wiki/Software/polkit/`) ausführt, wodurch Systemadministratoren auf strukturierte Weise den Zugang zu „privilegierten“ Operationen gewähren können, die erweiterte Berechtigungen erfordern. Indem der Polkit-Dienst angefragt wird, kann eine mit Berechtigungen ausgestattete Systemkomponente die Information erhalten, ob normalen Benutzern Berechtigungen gewährt werden dürfen. Zum Beispiel kann einer normalen Nutzerin die Berechtigung gegeben werden, das System in den Bereitschaftsmodus zu versetzen, unter der Voraussetzung, dass sie lokal vor Ort angemeldet ist.

`polkit-wheel-service` [Scheme-Variable]
Dieser Dienst richtet die `wheel`-Benutzergruppe als Administratoren für den Polkit-Dienst ein. Der Zweck davon ist, dass Benutzer in der `wheel`-Benutzergruppe nach ihren eigenen Passwörtern gefragt werden statt dem Passwort des Administratornutzers `root`, wenn sie administrative Tätigkeiten durchführen, ähnlich dem, wie sich `sudo` verhält.

`upower-service-type` [Scheme-Variable]
Typ des Dienstes, der `upowerd` (`https://upower.freedesktop.org/`) ausführt, ein Programm zur systemweiten Überwachung des Energieverbrauchs und der Akkuladung. Er hat die angegebenen Konfigurationseinstellungen.

Er implementiert die D-Bus-Schnittstelle `org.freedesktop.UPower`. Insbesondere wird UPower auch von GNOME benutzt.

`upower-configuration` [Datentyp]

Repräsentiert die Konfiguration von UPower.

`upower` (Vorgabe: *upower*)
: Das Paket, das für `upower` benutzt werden soll.

`watts-up-pro?` (Vorgabe: `#f`)
: Aktiviert das Watts-Up-Pro-Gerät.

`poll-batteries?` (Vorgabe: `#t`)
: Aktiviert das regelmäßige Abfragen des Kernels bezüglich Änderungen am Stand der Akku-Ladung.

`ignore-lid?` (Vorgabe: `#f`)
: Ignorieren, ob der Rechner zugeklappt ist. Das kann gewünscht sein, wenn Auf- und Zuklappen nicht richtig erkannt werden.

`use-percentage-for-policy?` (Vorgabe: `#t`)
: Ob sich die Richtlinie am Akku-Ladestand in Prozent statt an der verbleibenden Zeit orientieren soll. Eine Richtlinie, die den Prozentstand als Orientierung nimmt, ist in der Regel zuverlässiger.

`percentage-low` (Vorgabe: `20`)
: Wenn `use-percentage-for-policy?` auf `#t` gesetzt ist, wird hiermit der Prozentstand festgelegt, ab dem der Akku-Ladestand als niedrig gilt.

`percentage-critical` (Vorgabe: `5`)
: Wenn `use-percentage-for-policy?` auf `#t` gesetzt ist, wird hiermit der Prozentstand festgelegt, ab dem der Akku-Ladestand als kritisch gilt.

`percentage-action` (Vorgabe: `2`)
: Wenn `use-percentage-for-policy?` auf `#t` gesetzt ist, wird hiermit der Prozentstand festgelegt, ab dem Maßnahmen eingeleitet werden.

`time-low` (Vorgabe: `1200`)
: Wenn `use-percentage-for-policy?` auf `#f` gesetzt ist, wird hiermit die verbleibende Zeit in Sekunden festgelegt, ab der der Akku-Ladestand als niedrig gilt.

`time-critical` (Vorgabe: `300`)
: Wenn `use-percentage-for-policy?` auf `#f` gesetzt ist, wird hiermit die verbleibende Zeit in Sekunden festgelegt, ab der der Akku-Ladestand als kritisch gilt.

`time-action` (Vorgabe: `120`)
: Wenn `use-percentage-for-policy?` auf `#f` gesetzt ist, wird hiermit die verbleibende Zeit in Sekunden festgelegt, ab der Maßnahmen eingeleitet werden.

`critical-power-action` (Vorgabe: `'hybrid-sleep`)
: Welche Maßnahme eingeleitet wird, wenn die `percentage-action` oder `time-action` erreicht wurde (je nachdem, wie `use-percentage-for-policy?` eingestellt wurde).

Mögliche Werte sind:

- `'power-off`
- `'hibernate`
- `'hybrid-sleep`.

`udisks-service` [*#:udisks udisks*] [Scheme-Prozedur]
Liefert einen Dienst für UDisks (`https://udisks.freedesktop.org/docs/latest/`), einen Daemon zur *Datenträgerverwaltung*, der Benutzeroberflächen mit Benachrichtigungen und Möglichkeiten zum Einbinden und Aushängen von Datenträgern versorgt. Zu den Programmen, die mit UDisks kommunizieren, gehört der Befehl `udisksctl`, der Teil von UDisks ist, sowie GNOME Disks. Beachten Sie, dass Udisks über den Befehl `mount` funktioniert; man kann damit also nur die Dateisystemwerkzeuge benutzen, die ins Systemprofil installiert sind. Wenn Sie also NTFS-Dateisysteme zum Lesen und Schreiben öffnen können möchten, müssen Sie `ntfs-3g` systemweit installiert haben.

`colord-service-type` [Scheme-Variable]
Dies ist der Typ des Dienstes, der `colord` ausführt. Dabei handelt es sich um einen Systemdienst mit einer D-Bus-Schnittstelle, um die Farbprofile von Ein- und Ausgabegeräten wie Bildschirmen oder Scannern zu verwalten. Insbesondere wird colord vom grafischen GNOME-Farbverwaltungswerkzeug benutzt. Siehe den Webauftritt von colord (`https://www.freedesktop.org/software/colord/`) für weitere Informationen.

`sane-service-type` [Scheme-Variable]
Mit diesem Dienst wird Zugriff auf Scanner über SANE (`http://www.sane-project.org`) möglich, indem er die nötigen udev-Regeln installiert. Er ist Teil von `%desktop-services` (siehe Abschnitt 12.9.9 [Desktop-Dienste], Seite 366) und verwendet in den Vorgabeeinstellungen das Paket `sane-backends-minimal` (siehe unten) für die Hardwareunterstützung.

`sane-backends-minimal` [Scheme-Variable]
Das vorgegebene Paket, das durch den `sane-service-type` installiert wird. Es unterstützt viele aktuelle Scanner.

`sane-backends` [Scheme-Variable]
Dieses Paket bietet Unterstützung für alle Scanner, die `sane-backends-minimal` unterstützt, und außerdem für ältere Hewlett-Packard-Scanner, die das Paket `hplip` unterstützt. Um es auf einem System zu benutzen, das auf den `%desktop-services` aufbaut, können Sie `modify-services` benutzen (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639), etwa so:

```
(use-modules (gnu))
(use-service-modules
  ...
  desktop)
(use-package-modules
  ...
  scanner)
```

```
(define %my-desktop-services
  ;; Alternative Diensteliste wie %desktop-services mit Unterstützung
  ;; für mehr Scanner.
  (modify-services %desktop-services
    (sane-service-type _ => sane-backends)))

(operating-system
  ...
  (services %my-desktop-services))
```

`geoclue-application` *Name* [*#:allowed? #t*] [*#:system? #f*] [*#:users '()*] [Scheme-Prozedur]

Liefert eine Konfiguration, mit der eine Anwendung auf Ortungsdaten von GeoClue zugreifen kann. Als *Name* wird die Desktop-ID der Anwendung angegeben, ohne die Pfadkomponente mit `.desktop`-Endung. Wenn *allowed?* wahr ist, hat die Anwendung standardmäßig Zugriff auf Ortungsinformationen. Der boolesche Wert *system?* zeigt an, ob die Anwendung eine Systemkomponente ist oder nicht. Zum Schluss wird für *users* eine Liste von Benutzeridentifikatoren (UIDs) aller Benutzerkonten angegeben, für die diese Anwendung Zugriff auf Ortungsinformationen gewährt bekommt. Eine leere Benutzerliste bedeutet, dass dies für alle Benutzer gewährt wird.

`%standard-geoclue-applications` [Scheme-Variable]

Die Standardliste wohlbekannter GeoClue-Anwendungskonfigurationen, mit der das GNOME-Werkzeug für Datum und Uhrzeit die Berechtigung bekommt, den aktuellen Ort abzufragen, um die Zeitzone festzulegen, und die Webbrowser IceCat und Epiphany Ortsinformationen abfragen dürfen. IceCat und Epiphany fragen beide zuerst beim Benutzer nach, bevor sie einer Webseite gestatten, den Ort des Benutzer abzufragen.

`geoclue-service` [*#:colord `colord`*] [*#:whitelist '()*] [*#:wifi-geolocation-url* `"https://location.services.mozilla.com/v1/geolocate?key=geoclue"`] [#:submit-data? #f] [#:wifi-submission-url `"https://location.services.mozilla.com/v1/submit?key=geoclue"`] [#:submission-nick `"geoclue"`] [#:applications %standard-geoclue-applications] [Scheme-Prozedur]

Liefert einen Dienst, der den Ortungsdienst GeoClue ausführt. Dieser Dienst bietet eine D-Bus-Schnittstelle an, mit der Anwendungen Zugriff auf den physischen Ort eines Benutzers anfragen können, und optional Informationen in Online-Ortsdatenbanken eintragen können. Siehe den Webauftritt von GeoClue (`https://wiki.freedesktop.org/www/Software/GeoClue/`) für weitere Informationen.

`bluetooth-service` [*#:bluez `bluez`*] [*#:auto-enable? #f*] [Scheme-Prozedur]

Liefert einen Dienst, der den `bluetoothd`-Daemon ausführt, welcher alle Bluetooth-Geräte verwaltet, und eine Reihe von D-Bus-Schnittstellen zur Verfügung stellt. Wenn AUTO-ENABLE? wahr ist, wird die Bluetooth-Steuerung automatisch beim Hochfahren gestartet, was sich als nützlich erweisen kann, wenn man eine Bluetooth-Tastatur oder -Maus benutzt.

Benutzer müssen zur `lp`-Benutzergruppe gehören, damit sie Zugriff auf den D-Bus-Dienst bekommen.

`bluetooth-service-type` [Scheme-Variable]

Dies ist der Diensttyp für das System zum Linux-Bluetooth-Protokollstapel (`https://bluez.org/`) (BlueZ), das die Konfigurationsdatei `/etc/bluetooth/main.conf` erzeugt. Der Wert für diesen Diensttyp ist ein `bluetooth-configuration`-Verbundsobjekt wie in diesem Beispiel:

```
(service bluetooth-service-type)
```

Siehe unten für Details zur `bluetooth-configuration`.

`bluetooth-configuration` [Datentyp]

Repräsentiert die Konfiguration für den `bluetooth-service-type`.

`bluez` (Vorgabe: `bluez`)
: Zu benutzendes `bluez`-Paket.

`name` (Vorgabe: `"BlueZ"`)
: Welchen Namen Sie für den Adapter als Voreinstellung geben.

`class` (Vorgabe: `#x000000`)
: Welche Geräteklasse Sie als Voreinstellung geben. Nur die Bits für „major device class“ und „minor device class“ werden beachtet.

`discoverable-timeout` (Vorgabe: `180`)
: Wie lange der Adapter erkennbar bleiben soll („discoverable mode“), bevor die Erkennbarkeit endet. Der Wert wird in Sekunden angegeben.

`always-pairable?` (Vorgabe: `#f`)
: Koppeln von Geräten immer zulassen, so dass kein Agent registriert sein muss.

`pairable-timeout` (Vorgabe: `0`)
: Wie lange Koppeln möglich sein soll („pairable mode“), bevor die Erkennbarkeit endet. Der Wert wird in Sekunden angegeben.

`device-id` (Vorgabe: `#f`)
: Dieses Gerät mit dieser Geräteidentifikatorbezugsquelle („vendor id source“, d.h. Assigner), Geräteidentifikator, Produkt- sowie Versionsinformationen im Device-ID-Profil ausweisen. Die Werte für *assigner*, *VID*, *PID* und *version* werden durch `":"` getrennt.

  Mögliche Werte sind:

  - `#f`, um nichts festzulegen.
  - `"assigner:1234:5678:abcd"`, wobei *assigner* entweder `usb` ist (die Voreinstellung) oder `bluetooth`.

`reverse-service-discovery?` (Vorgabe: `#t`)
: Bluetooth-Dienste auf Geräten erkennen, die sich mit unserem Gerät verbinden. Bei BR/EDR braucht man diese Option eigentlich nur für den Qualifizierungsprozess, weil der BITE-Tester in manchen Testfällen etwas gegen inverses SDP hat; bei LE wird hiermit die Funktion als GATT-Client deaktiviert, was sinnvoll ist, wenn unser System nur als Peripheriegerät eingesetzt wird.

`name-resolving?` (Vorgabe: `#t`)
: Namensauflösung bei Gerätesuche („Inquiry“) aktivieren. Legen Sie es auf `#f` fest, wenn Sie die Namen der entfernten Geräte nicht brauchen und Erkennungszyklen beschleunigen möchten.

`debug-keys?` (Vorgabe: `#f`)
: Ob „debug link keys“ für die Laufzeit persistent gespeichert werden. Vorgegeben ist `#f`; sie bleiben also nur gültig, solange die Verbindung mit ihnen anhält.

`controller-mode` (Vorgabe: `'dual`)
: Schränkt die Controller auf das angegebene Transportprotokoll ein. Bei `'dual` werden sowohl BR/EDR als auch LE benutzt (wenn die Hardware sie unterstützt).

  Mögliche Werte sind:

  - `'dual`
  - `'bredr`
  - `'le`

`multi-profile` (Vorgabe: `'off`)
: Unterstützung für Multi Profile Specification aktivieren. Hiermit kann eingestellt werden, ob ein System nur in Konfigurationen von Multiple Profiles Single Device (MPSD) laufen kann oder sowohl mit Konfigurationen in Multiple Profiles Single Device (MPSD) als auch Konfigurationen in Multiple Profiles Multiple Devices (MPMD) umgehen kann.

  Mögliche Werte sind:

  - `'off`
  - `'single`
  - `'multiple`

`fast-connectable?` (Vorgabe: `#f`)
: Dauerhaft eine beschleunigte Verbindung („Fast Connectable“) bei Adaptern, die sie unterstützen, ermöglichen. Ist dies aktiviert, können sich andere Geräte schneller mit unserem verbinden, jedoch steigt der Stromverbrauch. Diese Funktion steht nur auf Kernel-Version 4.1 und neuer vollständig zur Verfügung.

`privacy` (Vorgabe: `'off`)
: Die Voreinstellung zur Verfolgbarkeit.

  - `'off`: Nicht privat stellen.
  - `'network/on`: Ein Gerät nimmt nur Mitteilungen („advertising packets“) von anderen Geräten an, die private Adressen enthalten. Mit manchen alten Geräten funktioniert das vielleicht nicht, weil es voraussetzt, dass für alles RPA(s) benutzt werden.
  - `'device`: Auf „device privacy mode“ gestellte Geräte schützen nur vor Nachverfolgbarkeit des jeweiligen Geräts, aber wenn Mitteilungen von anderen Geräten deren Identity Address preisgeben, werden

sie genauso akzeptiert wie eine private Adresse. Das gilt auch bei anderen Geräten, die in der Vergangenheit ihre IRK mitgeteilt hatten.

Des Weiteren gibt es folgende Möglichkeiten, wenn *controller-mode* auf `'dual` gesetzt ist:

- `'limited-network`: Limited Discoverable Mode für Mitteilungen einsetzen, wodurch wie bei BR/EDR die Identity Address mitgeteilt wird, wenn das Gerät erkennbar ist, aber Network Privacy Mode beim Scannen nach Geräten gilt.
- `'limited-network`: Limited Discoverable Mode für Mitteilungen einsetzen, wodurch wie bei BR/EDR die Identity Address mitgeteilt wird, wenn das Gerät erkennbar ist, aber Device Privacy Mode beim Scannen nach Geräten gilt.

`just-works-repairing` (Vorgabe: `'never`)
: Wie auf einen vom anderen Gerät ausgelösten JUST-WORKS-Vorgang reagiert werden soll. Bei `'always` wird das andere Gerät akzeptiert, bei `'never` abgelehnt und bei `'confirm` nachgefragt.

  Mögliche Werte sind:

  - `'never`
  - `'confirm`
  - `'always`

`temporary-timeout` (Vorgabe: `30`)
: Wie lange ein temporäres Gerät koppelbar bleibt. Der Wert wird in Sekunden angegeben. Bei `0` läuft die Zeit nie aus.

`refresh-discovery?` (Vorgabe: `#t`)
: Zulassen, dass das Gerät eine SDP-Anfrage schickt, um bekannte Dienste zu ermitteln, sobald eine Verbindung hergestellt wurde.

`experimental` (Vorgabe: `#f`)
: Experimentelle Funktionen und Schnittstellen bereitstellen. Sie können auch als Liste von UUIDs angegeben werden.

  Mögliche Werte sind:

  - `#t`
  - `#f`
  - `(list (uuid <uuid-1>) (uuid <uuid-2>) ...)`.

  Die Liste der möglichen 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` (Vorgabe: `300`)
: Wie lange nach einer fehlgeschlagenen Namensauflösung *keine* erneute Auflösung des Namens des anderen Geräts versucht werden soll.

`page-scan-type` (Vorgabe: `#f`)
: Auf welche Art die Erkennung („Scan“) von Verbindungsversuchen („Paging“) bei BR/EDR durchgeführt wird.

`page-scan-interval` (Vorgabe: `#f`)
: Aktivitätsintervall zwischen Anfängen von Erkennungsphasen von Verbindungsversuchen bei BR/EDR.

`page-scan-window` (Vorgabe: `#f`)
: Aktivitätsfenster jeder Erkennungsphase von Verbindungsversuchen bei BR/EDR.

`inquiry-scan-type` (Vorgabe: `#f`)
: Auf welche Art die Erkennung von Gerätesuchen („Inquiry“) bei BR/EDR durchgeführt wird.

`inquiry-scan-interval` (Vorgabe: `#f`)
: Aktivitätsintervall zwischen Anfängen von Erkennungsphasen von Gerätesuchen bei BR/EDR.

`inquiry-scan-window` (Vorgabe: `#f`)
: Aktivitätsfenster jeder Erkennungsphase von Gerätesuchen bei BR/EDR.

`link-supervision-timeout` (Vorgabe: `#f`)
: Zeitbegrenzung, ab der eine inaktive Verbindung bei BR/EDR als getrennt gilt.

`page-timeout` (Vorgabe: `#f`)
: Zeitbegrenzung, ab der ein unbeantworteter Verbindungsversuch bei BR/EDR als gescheitert gilt.

`min-sniff-interval` (Vorgabe: `#f`)
: Minimale Intervalllänge im Sniff-Modus bei BR/EDR.

`max-sniff-interval` (Vorgabe: `#f`)
: Maximale Intervalllänge im Sniff-Modus bei BR/EDR.

`min-advertisement-interval` (Vorgabe: `#f`)
: Minimale Intervalllänge zwischen Mitteilungen bei LE (nur für Legacy Advertisement).

`max-advertisement-interval` (Vorgabe: `#f`)
: Maximale Intervalllänge zwischen Mitteilungen bei LE (nur für Legacy Advertisement).

`multi-advertisement-rotation-interval` (Vorgabe: `#f`)
: Wenn verschiedene Mitteilungen ausgegeben werden, mit welchem Intervall dazwischen rotiert wird, bei LE.

`scan-interval-auto-connect` (Vorgabe: `#f`)
: Intervall zwischen Anfängen von Erkennungsphasen bei passiven Erkennungen zur Unterstützung für autonome Verbindung, bei LE.

`scan-window-auto-connect` (Vorgabe: `#f`)
: Länge jedes Erkennungsfensters bei passiven Erkennungen zur Unterstützung für autonome Verbindung, bei LE.

`scan-interval-suspend` (Vorgabe: `#f`)
: Intervall zwischen Anfängen von Erkennungsphasen bei aktiven Erkennungen zur Unterstützung für Wake-from-Suspend, bei LE.

`scan-window-suspend` (Vorgabe: `#f`)
: Länge jedes Erkennungsfensters bei aktiven Erkennungen zur Unterstützung für Wake-from-Suspend, bei LE.

`scan-interval-discovery` (Vorgabe: `#f`)
: Intervall zwischen Anfängen von Erkennungsphasen bei aktiven Erkennungen von sich mitteilenden Geräten, bei LE.

`scan-window-discovery` (Vorgabe: `#f`)
: Länge jedes Erkennungsfensters bei aktiven Erkennungen von sich mitteilenden Geräten, bei LE.

`scan-interval-adv-monitor` (Vorgabe: `#f`)
: Intervall zwischen Anfängen von Erkennungsphasen bei passiven Erkennungen zur Unterstützung der Advertisement-Monitor-Programmschnittstellen, bei LE.

`scan-window-adv-monitor` (Vorgabe: `#f`)
: Länge jedes Erkennungsfensters bei passiven Erkennungen zur Unterstützung der Advertisement-Monitor-Programmschnittstellen, bei LE.

`scan-interval-connect` (Vorgabe: `#f`)
: Intervall zwischen Anfängen von Erkennungsphasen beim Verbindungsaufbau.

`scan-window-connect` (Vorgabe: `#f`)
: Länge jedes Erkennungsfensters beim Verbindungsaufbau.

`min-connection-interval` (Vorgabe: `#f`)
: Kleinstes gewünschtes Intervall für eine Verbindung, bei LE. Demgegenüber hat Vorrang, wenn ein Wert über die Schnittstelle Load Connection Parameters bestimmt wird.

`max-connection-interval` (Vorgabe: `#f`)
: Größtes gewünschtes Intervall für eine Verbindung, bei LE. Demgegenüber hat Vorrang, wenn ein Wert über die Schnittstelle Load Connection Parameters bestimmt wird.

`connection-latency` (Vorgabe: `#f`)
: Gewünschte Latenz für eine Verbindung, bei LE. Demgegenüber hat Vorrang, wenn ein Wert über die Schnittstelle Load Connection Parameters bestimmt wird.

`connection-supervision-timeout` (Vorgabe: `#f`)
: Gewünschte Zeitbegrenzung, ab der eine inaktive Verbindung als getrennt gilt, bei LE. Demgegenüber hat Vorrang, wenn ein Wert über die Schnittstelle Load Connection Parameters bestimmt wird.

`autoconnect-timeout` (Vorgabe: `#f`)
: Gewünschte Zeitbegrenzung für autonome Verbindungsversuche, bei LE. Demgegenüber hat Vorrang, wenn ein Wert über die Schnittstelle Load Connection Parameters bestimmt wird.

`adv-mon-allowlist-scan-duration` (Vorgabe: `300`)
: Dauer der Erkennungsphasen für Geräte auf der Liste erlaubter Geräte bei verzahnten Arten der Erkennungsphase („Interleaving Scan“). Wird nur bei Erkennungsphasen für Advertisement Monitors benutzt. Gemessen in Millisekunden.

`adv-mon-no-filter-scan-duration` (Vorgabe: `500`)
: Dauer der Erkennungsphasen für ungefiltert alle Geräte bei verzahnten Arten der Erkennungsphase („Interleaving Scan“). Wird nur bei Erkennungsphasen für Advertisement Monitors benutzt. Gemessen in Millisekunden.

`enable-adv-mon-interleave-scan?` (Vorgabe: `#t`)
: Verzahnte Arten der Erkennungsphasen für Advertisement Monitors benutzen, was beim Energiesparen hilft.

`cache` (Vorgabe: `'always`)
: GATT-Attribute-Zwischenspeicher.

  Mögliche Werte sind:

  - `'always`: Immer Attribute zwischenspeichern, selbst von nicht gekoppelten Geräten. So klappt die Zusammenarbeit mit Geräten am besten, die Dauer für eine erneute Verbindung bleibt konsistent und man kann Benachrichtigungen für alle Geräte nachvollziehen.
  - `'yes`: Nur für gekoppelte Geräte deren Attribute speichern.
  - `'no`: Niemals Attribute zwischenspeichern.

`key-size` (Vorgabe: `0`)
: Kleinste von diesem Gerät eingeforderte Schlüsselgröße („Encryption Key Size“), um auf die gesicherten Charakteristika zuzugreifen.

  Mögliche Werte sind:

  - `0`: Keine Anforderungen.
  - `7 <= N <= 16`

`exchange-mtu` (Vorgabe: `517`)
: Wie groß die „Exchange MTU“ für GATT sein soll. Mögliche Werte sind:

  - `23 <= N <= 517`

`att-channels` (Vorgabe: `3`)
: Anzahl der ATT-Kanäle. Mögliche Werte sind:

  - `1`: EATT ist deaktiviert.

- `2 <= N <= 5`

`session-mode` (Vorgabe: `'basic`)
: Für AVDTP der Modus des L2CAP-Signaling-Kanals.

  Mögliche Werte sind:

  - `'basic`: L2CAP Basic Mode benutzen.
  - `'ertm`: L2CAP Enhanced Retransmission Mode benutzen.

`stream-mode` (Vorgabe: `'basic`)
: Für AVDTP der Modus des L2CAP Transport Channel.

  Mögliche Werte sind:

  - `'basic`: L2CAP Basic Mode benutzen.
  - `'streaming`: L2CAP Streaming Mode benutzen.

`reconnect-uuids` (Vorgabe: `'()`)
: Als die ReconnectUUIDs definieren Sie diejenigen Dienste entfernter Geräte, mit denen eine neue Verbindung aufgebaut werden soll, wenn die Verbindung abreißt (durch „link supervision timeout“, d.h. die Zeitbegrenzung, ab der eine inaktive Verbindung als getrennt gilt). Im Policy-Plugin sollte eine vernünftige Voreinstellung zu finden sein. Die Liste hier hat Vorrang. Wenn Sie die leere Liste angeben, werden Verbindungen *nicht* neu aufgebaut.

  Mögliche Werte sind:

  - `'()`
  - `(list (uuid <uuid-1>) (uuid <uuid-2>) ...)`.

`reconnect-attempts` (Vorgabe: `7`)
: Wie oft bei einem Verbindungsverlust versucht werden soll, eine neue Verbindung aufzubauen. Für den Wert 0 wird Neuverbinden deaktiviert.

`reconnect-intervals` (Vorgabe: `'(1 2 4 8 16 32 64)`)
: Definiert eine Liste von Zeitintervallen in Sekunden, wie lange nach jedem Versuch gewartet wird. Wenn die in *reconnect-attempts* festgelegte Anzahl Versuche größer ist als die Liste der Zeitintervalle lang ist, wird das letzte Intervall wiederholt, bis alle Versuche ausgeschöpft sind.

`auto-enable?` (Vorgabe: `#f`)
: Ob alle erkannten Controller sofort aktiviert werden sollen. Dazu zählen Adapter, die beim Hochfahren schon verfügbar sind, wie auch Adapter, die später erst eingesteckt werden.

`resume-delay` (Vorgabe: `2`)
: Audio-Geräte, die durch einen Ruhezustand getrennt wurden, werden beim Aufwecken neu verbunden. Mit *resume-delay* wird festgelegt, wie lange nach dem Aufwecken des Controllers mit einem Neuverbindungsversuch gewartet wird. Wenn mit der Neuverbindung länger gewartet wird, gibt es weniger Probleme mit gleichzeitig genutztem WLAN. Der Wert wird in Sekunden angegeben.

`rssi-sampling-period` (Vorgabe: `#xFF`)
: Voreinstellung für die RSSI-Abtastperiode. Sie wird benutzt, wenn ein Client einen Advertisement Monitor registriert und die RSSISamplingPeriod *nicht* vorgibt.

  Mögliche Werte sind:

  - `#x0`: Alle Mitteilungen melden.
  - `N = #xXX`: Mitteilungen alle N x 100 msec melden (aus dem Bereich: #x01 to #xFE)
  - `#xFF`: In der Beobachtungsphase nur eine Mitteilung pro Gerät melden.

`gnome-keyring-service-type` [Scheme-Variable]
: Dies ist der Typ des Dienstes, der den GNOME-Schlüsselbund (`https://wiki.gnome.org/Projects/GnomeKeyring`) bereitstellt. Sein Wert ist ein `gnome-keyring-configuration`-Objekt (siehe unten).

  Dieser Dienst fügt das `gnome-keyring`-Paket zum Systemprofil hinzu und erweitert PAM um Einträge zur Nutzung von `pam_gnome_keyring.so`, wodurch der Schlüsselbund von Nutzern entsperrt wird, wenn sie sich anmelden, und passwd auch das Passwort des Schlüsselbunds festlegt.

`gnome-keyring-configuration` [Datentyp]
: Verbundsobjekt für die Konfiguration des GNOME-Schlüsselbunddienstes.

  `keyring` (Vorgabe: `gnome-keyring`)
  : Welches GNOME-Schlüsselbund-Paket benutzt werden soll.

  `pam-services`
  : Eine Liste von Paaren aus `(`*`Dienst`*` . `*`Typ`*`)`, die zu erweiternde PAM-Dienste bezeichnen. Dabei steht *Dienst* für den Namen eines bestehenden Dienstes, der erweitert werden soll, und als *Typ* kann `login` oder `passwd` angegeben werden.

    Wenn `login` angegeben wird, wird ein optionales `pam_gnome_keyring.so` zum Auth-Block ohne Argumente und zum Session-Block mit `auto_start` hinzugefügt. Wenn `passwd` angegeben wird, wird ein optionales `pam_gnome_keyring.so` zum Password-Block ohne Argumente hinzugefügt.

    Der vorgegebene Inhalt ist „gdm-password“ mit dem Wert `login` und „passwd“ mit dem Wert `passwd`.

`seatd-service-type` [Scheme-Variable]
: seatd (`https://sr.ht/~kennylevinsen/seatd/`) ist ein minimaler Daemon zur Sitzungsverwaltung.

  Sitzungsverwaltung bedeutet, dass der Zugriff auf gemeinsame Geräte (Grafik, Eingabegeräte) vermittelt wird, ohne dass die Anwendungen, die zugreifen wollen, Administratorrechte brauchen.

```
(append
  (list
```

```
 ;; damit seatd läuft
 (service seatd-service-type))

;; normalerweise zusammen mit %base-services
%base-services)
```

`seatd` funktioniert über einen Unix-Socket. Dabei stellt `libseat` den clientseitigen Teil des Protokolls bereit. Wenn Anwendungen über `seatd` Zugriff auf die gemeinsamen Ressourcen brauchen (z.B. `sway`), dann müssen sie Zugriff auf diesen Socket haben. Um das zu bewerkstelligen, kann man das Benutzerkonto, mit dem sie laufen, zur Gruppe hinzufügen, der der Socket von `seatd` gehört (in der Regel die Gruppe „seat“). Das geht so:

```
(user-account
  (name "alice")
  (group "users")
  (supplementary-groups '("wheel"   ;zur sudo-Nutzung usw. berechtigen
                          "seat"    ;Sitzungsverwaltung
                          "audio"   ;Soundkarte
                          "video"   ;Videogeräte wie Webcams
                          "cdrom")) ;die gute alte CD-ROM
  (comment "Bobs Schwester"))
```

Je nachdem, wie Sie das System einrichten, müssen Sie nicht nur normale Benutzer, sondern auch Systembenutzerkonten, zu dieser Gruppe hinzufügen. Zum Beispiel verlangen manche greetd-Greeter, dass Grafik zur Verfügung steht, also müssen auch diese Benutzerkonten das mit seatd aushandeln können.

`seatd-configuration` [Datentyp]

Verbundsobjekt für die Konfiguration des seatd-Daemon-Dienstes.

`seatd` (Vorgabe: `seatd`)
: Das zu benutzende seatd-Paket.

`group` (Vorgabe: ‘`"seat"`’)
: Die Gruppe, die den seatd-Socket besitzt.

`socket` (Vorgabe: ‘`"/run/seatd.sock"`’)
: Wo der seatd-Socket erzeugt wird.

`logfile` (Vorgabe: ‘`"/var/log/seatd.log"`’)
: In welche Protokolldatei geschrieben wird.

`loglevel` (Vorgabe: ‘`"error"`’)
: Die Protokollierungsstufe, wie ausführlich die Ausgaben ins Protokoll sind. Mögliche Werte: ‘`"silent"`’ (keine Ausgaben), ‘`"error"`’ (nur Fehler), ‘`"info"`’ und ‘`"debug"`’ (zur Fehlersuche).

12.9.10 Tondienste

Das Modul `(gnu services sound)` stellt einen Dienst zur Verfügung, um das Advanced-Linux-Sound-Architecture-System (ALSA) zu konfigurieren, so dass PulseAudio als bevorzugter ALSA-Ausgabetreiber benutzt wird.

`alsa-service-type` [Scheme-Variable]

Dies ist der Typ des Dienstes für das als Advanced Linux Sound Architecture (ALSA) (`https://alsa-project.org/`) bekannte System, das die Konfigurationsdatei `/etc/asound.conf` erzeugt. Der Wert für diesen Diensttyp ist ein `alsa-configuration`-Verbundsobjekt wie in diesem Beispiel:

```
(service alsa-service-type)
```

Siehe die folgenden Details zur `alsa-configuration`.

`alsa-configuration` [Datentyp]

Repräsentiert die Konfiguration für den Dienst `alsa-service`.

`alsa-plugins` (Vorgabe: *alsa-plugins*)
: `alsa-plugins`-Paket, was benutzt werden soll.

`pulseaudio?` (Vorgabe: *#t*)
: Ob ALSA-Anwendungen transparent den PulseAudio-Audioserver (`https://www.pulseaudio.org/`) benutzen sollen.

  Wenn PulseAudio benutzt wird, können Sie gleichzeitig mehrere Anwendungen mit Tonausgabe ausführen und sie unter anderem mit `pavucontrol` einzeln einstellen.

`extra-options` (Vorgabe: `""`)
: Die Zeichenkette, die an die Datei `/etc/asound.conf` angehängt werden soll.

Wenn einzelne Benutzer von ALSAs Systemkonfiguration abweichende Einstellungen vornehmen möchten, können Sie das mit der Konfigurationsdatei `~/.asoundrc` tun:

```
# In Guix müssen wir den absoluten Pfad von Plugins angeben.
pcm_type.jack {
  lib "/home/alice/.guix-profile/lib/alsa-lib/libasound_module_pcm_jack.so"
}

# ALSA an jack weiterleiten:
# <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"
  }
}
```

Siehe `https://www.alsa-project.org/main/index.php/Asoundrc` für die Details.

`pulseaudio-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den PulseAudio-Soundserver (`https://www.pulseaudio.org/`). Mit ihm können die Voreinstellungen systemweit abgeändert werden. Dazu benutzen Sie eine `pulseaudio-configuration`, siehe unten.

> **Warnung:** Durch diesen Dienst werden Einstellungen vorgenommen, die Vorrang vor den Konfigurationsdateien des Benutzers haben. Wenn PulseAudio Konfigurationsdateien in `~/.config/pulse` beachten soll, müssen Sie die Umgebungsvariablen `PULSE_CONFIG` und `PULSE_CLIENTCONFIG` in Ihrer `~/.bash_profile` deaktivieren.
>
> **Warnung:** Dieser Dienst sorgt alleine noch nicht dafür, dass auf ihrer Maschine das `pulseaudio`-Paket vorliegt. Er fügt bloß Konfigurationsdateien dafür hinzu, wie im Folgenden beschrieben. Für den (zugegebenermaßen unwahrscheinlichen) Fall, dass Ihnen ein `pulseaudio`-Paket fehlt, möchten Sie es vielleicht durch den oben genannten `alsa-service-type` aktivieren.

`pulseaudio-configuration` [Datentyp]

Repräsentiert die Konfiguration für den Dienst `pulseaudio-service`.

`client-conf` (Vorgabe: `'()`)
: Eine Liste der Einstellungen, die in `client.conf` vorgenommen werden. Hierfür wird eine Liste von entweder Zeichenketten oder Symbol-Wert-Paaren akzeptiert. Eine Zeichenkette wird so, wie sie ist, eingefügt, mit einem Zeilenumbruch danach. Ein Paar wird als „Schlüssel = Wert“ formatiert, auch hier gefolgt von einem Zeilenumbruch.

`daemon-conf` (Vorgabe: `'((flat-volumes . no))`)
: Eine Liste der Einstellungen, die in `daemon.conf` vorgenommen werden, im gleichen Format wie *client-conf*.

`script-file` (Vorgabe: `(file-append pulseaudio "/etc/pulse/default.pa")`)
: Eine Datei mit dem als `default.pa` zu nutzenden Skript. Wenn Sie das Feld `extra-script-files` von weiter unten benutzen, wird ans Ende des angegebenen Skripts eine `.include`-Direktive angehängt, die auf `/etc/pulse/default.pa.d` zeigt.

`extra-script-files` (Vorgabe: `'()`)
: Eine Liste dateiartiger Objekte, die zusätzliche PulseAudio-Skripte definieren, die bei der Initialisierung des `pulseaudio`-Daemons ausgeführt werden, nachdem das Hauptskript aus `script-file` durchgelaufen ist. Die Skripte werden im Verzeichnis `/etc/pulse/default.pa.d` abgelegt

und sollten als Dateinamenserweiterung ebenso '`.pa`' haben. Eine Referenz der verfügbaren Befehle bekommen Sie durch Ausführen von `man pulse-cli-syntax`.

`system-script-file` (Vorgabe: `(file-append pulseaudio "/etc/pulse/system.pa")`)
Welche Skriptdatei als `system.pa` verwendet werden soll.

Im folgenden Beispiel werden das Standard-PulseAudio-Kartenprofil, das Standard-Ziel und die Standard-Quelle für eine alte SoundBlaster-Audigy-Soundkarte eingerichtet:

```
(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")))))
```

Wir merken an, dass `pulseaudio-service-type` zu `%desktop-services` dazugehört. Wenn Ihre Betriebssystemdeklaration also von einer der „Desktop“-Vorlagen abstammt, dann werden Sie das obige Beispiel anpassen wollen, damit stattdessen der in `%desktop-services` bestehende `pulseaudio-service-type`-Dienst modifiziert wird mit `modify-services` (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639) und kein neuer Dienst angelegt wird.

`ladspa-service-type` [Scheme-Variable]
Der Diensttyp für den Dienst, der die *LADSPA_PATH*-Variable festlegt, damit sie beachtende Programme, z.B. PulseAudio, LADSPA-Plugins laden können.

Das folgende Beispiel wird den Dienst so einrichten, dass Module aus dem `swh-plugins`-Paket aktiviert werden:

```
(service ladspa-service-type
         (ladspa-configuration (plugins (list swh-plugins))))
```

Siehe `http://plugin.org.uk/ladspa-swh/docs/ladspa-swh.html` für die Details.

12.9.11 Datenbankdienste

Das Modul `(gnu services databases)` stellt die folgenden Dienste zur Verfügung.

PostgreSQL

Das folgende Beispiel zeigt einen PostgreSQL-Dienst in seiner Vorgabekonfiguration.

```
(service postgresql-service-type
         (postgresql-configuration
          (postgresql postgresql-10)))
```

Wenn diese Dienste nicht starten können, ist vielleicht bereits ein inkompatibler Datenbankverbund („Cluster“) in *data-directory* vorhanden. Ändern Sie dann den Wert dafür (oder falls Sie den Verbund nicht mehr brauchen, löschen Sie *data-directory* einfach) und starten Sie den Dienst neu.

Nach Voreinstellung müssen sich normale Benutzerkonten des Guix-Systems, die mit PostgreSQL kommunizieren, sogenannte „Peers“, zunächst authentifizieren. Allerdings ist für das `postgres`-Benutzerkonto keine Shell eingestellt, wodurch keine `psql`-Befehle durch diesen Benutzer ausgeführt werden können. Um `psql` benutzen zu können, können Sie sich vorläufig als der `postgres`-Nutzer unter Angabe einer Shell anmelden, ein Konto für einen PostgreSQL-Administrator (einen „Superuser“) mit demselben Namen wie einer der Benutzer des Systems einrichten und dann die zugehörige Datenbank erstellen.

```
sudo -u postgres -s /bin/sh
createuser --interactive
createdb $BENUTZER_ANMELDENAME  # muss angepasst werden
```

`postgresql-configuration` [Datentyp]

Der Datentyp repräsentiert die Konfiguration für den `postgresql-service-type`.

`postgresql`
: Das PostgreSQL-Paket, was für diesen Dienst benutzt werden soll.

`port` (Vorgabe: `5432`)
: Der Port, auf dem PostgreSQL lauschen soll.

`locale` (Vorgabe: `"en_US.utf8"`)
: Welche Regions- und Spracheinstellung („Locale“) beim Erstellen des Datenbankverbunds voreingestellt werden soll.

`config-file` (Vorgabe: `(postgresql-config-file)`)
: Aus welcher Konfigurationsdatei die Laufzeitkonfiguration von PostgreSQL geladen werden soll. Vorgegeben ist, die Einstellungen aus einem postgresql-config-file-Verbundsobjekt mit Vorgabewerten zu nehmen.

`log-directory` (Vorgabe: `"/var/log/postgresql"`)
: In welchem Verzeichnis die Ausgabe von `pg_ctl` in eine Datei namens `"pg_ctl.log"` geschrieben wird. Diese Datei kann zum Beispiel dabei helfen, Fehler in der Konfiguration von PostgreSQL zu finden.

`data-directory` (Vorgabe: `"/var/lib/postgresql/data"`)
: Das Verzeichnis, in dem die Daten gespeichert werden sollen.

`extension-packages` (Vorgabe: `'()`)
: Zusätzliche Erweiterungen werden aus den unter *extension-packages* aufgeführten Paketen geladen. Erweiterungen sind zur Laufzeit verfügbar. Zum Beispiel kann ein Nutzer den postgresql-service-Dienst wie in diesem Beispiel konfigurieren, um eine geografische Datenbank mit Hilfe der `postgis`-Erweiterung zu erzeugen:

```
(use-package-modules databases geo)

(operating-system
  ...
  ;; postgresql wird benötigt, um „psql“ auszuführen, aber postgis ist
  ;; für den Betrieb nicht unbedingt notwendig.
  (packages (cons* postgresql %base-packages))
  (services
```

```
    (cons*
      (service postgresql-service-type
               (postgresql-configuration
                (postgresql postgresql-10)
                (extension-packages (list postgis))))
      %base-services)))
```

Dann wird die Erweiterung sichtbar und Sie können eine leere geografische Datenbak auf diese Weise initialisieren:

```
psql -U postgres
> create database postgistest;
> \connect postgistest;
> create extension postgis;
> create extension postgis_topology;
```

Es ist nicht notwendig, dieses Feld für contrib-Erweiterungen wie hstore oder dblink hinzuzufügen, weil sie bereits durch postgresql geladen werden können. Dieses Feld wird nur benötigt, um Erweiterungen hinzuzufügen, die von anderen Paketen zur Verfügung gestellt werden.

`postgresql-config-file` [Datentyp]

Der Datentyp, der die Konfigurationsdatei von PostgreSQL repräsentiert. Wie im folgenden Beispiel gezeigt, kann er benutzt werden, um die Konfiguration von PostgreSQL anzupassen. Es sei darauf hingewiesen, dass Sie jeden beliebigen G-Ausdruck oder Dateinamen anstelle dieses Verbunds angeben können, etwa wenn Sie lieber eine bestehende Konfigurationsdatei benutzen möchten.

```
(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
             '(("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` (Vorgabe: `"syslog"`)

Welche Protokollierungsmethode für PostgreSQL benutzt werden soll. Hier können mehrere Werte kommagetrennt angegeben werden.

`hba-file` (Vorgabe: `%default-postgres-hba`)
: Ein Dateiname oder G-Ausdruck für die Konfiguration der Authentifizierung durch den Server („Host-based Authentication“).

`ident-file` (Vorgabe: `%default-postgres-ident`)
: Ein Dateiname oder G-Ausdruck für die Konfiguration der Benutzernamensabbildung („User name mapping“).

`socket-directory` (Vorgabe: `"/var/run/postgresql"`)
: Gibt an, in welchem Verzeichnis der oder die Unix-Socket(s) zu finden sind, auf denen PostgreSQL auf Verbindungen von Client-Anwendungen lauscht. Ist dies auf `""` gesetzt, so lauscht PostgreSQL auf gar keinem Unix-Socket, so dass Verbindungen zum Server nur mit TCP/IP-Sockets aufgebaut werden können.

  Der Vorgabewert `#false` bedeutet, dass der voreingestellte Wert von PostgreSQL benutzt wird. Derzeit ist die Voreinstellung ‘`/tmp`’.

`extra-config` (Vorgabe: `'()`)
: Eine Liste zusätzlicher Schlüssel-Wert-Kombinationen, die in die PostgreSQL-Konfigurationsdatei aufgenommen werden sollen. Jeder Eintrag in der Liste muss eine Liste von Listen sein, deren erstes Element dem Schlüssel entspricht und deren übrige Elemente die Werte angeben.

  Als Werte können Zahlen, Boolesche Ausdrücke oder Zeichenketten dienen. Sie werden auf die Parametertypen von PostgreSQL, `Boolean`, `String`, `Numeric`, `Numeric with Unit` und `Enumerated`, abgebildet, die hier (`https://www.postgresql.org/docs/current/config-setting.html`) beschrieben werden.

`postgresql-role-service-type` [Scheme-Variable]

Mit diesem Dienst können PostgreSQL-Rollen und -Datenbanken erzeugt werden, nachdem der PostgreSQL-Dienst gestartet worden ist. Hier ist ein Beispiel, wie man ihn benutzt.

```
(service postgresql-role-service-type
            (postgresql-role-configuration
             (roles
              (list (postgresql-role
                     (name "test")
                     (create-database? #t))))))
```

Dieser Dienst kann mit weiteren Rollen erweitert werden, wie in diesem Beispiel:

```
(service-extension postgresql-role-service-type
                   (const (postgresql-role
                           (name "alice")
                           (create-database? #t))))
```

`postgresql-role` [Datentyp]

PostgreSQL verwaltet Zugriffsrechte auf Datenbanken mit dem Rollenkonzept. Eine Rolle kann als entweder ein Datenbanknutzer oder eine Gruppe von Datenbanknutzern verstanden werden, je nachdem, wie die Rolle eingerichtet wurde. Rollen können

Eigentümer von Datenbankobjekten sein (zum Beispiel von Tabellen) und sie können anderen Rollen Berechtigungen auf diese Objekte zuweisen oder steuern, wer auf welche Objekte Zugriff hat.

`name` Der Rollenname.

`permissions` (Vorgabe: `'(createdb login)`)
Die Liste der Berechtigungen der Rolle. Unterstützte Berechtigungen sind `bypassrls`, `createdb`, `createrole`, `login`, `replication` und `superuser`.

`create-database?` (Vorgabe: `#f`)
Ob eine Datenbank mit demselben Namen wie die Rolle erzeugt werden soll.

`postgresql-role-configuration` [Datentyp]
Der Datentyp, der die Konfiguration des *postgresql-role-service-type* repräsentiert.

`host` (Vorgabe: `"/var/run/postgresql"`)
Mit welchem PostgreSQL-Host eine Verbindung hergestellt werden soll.

`log` (Vorgabe: `"/var/log/postgresql_roles.log"`)
Der Dateiname der Protokolldatei.

`roles` (Vorgabe: `'()`)
Welche PostgreSQL-Rollen von Anfang an erzeugt werden sollen.

MariaDB/MySQL

`mysql-service-type` [Scheme-Variable]
Dies ist der Diensttyp für einen Datenbankserver zu MySQL oder MariaDB. Sein Wert ist ein `mysql-configuration`-Objekt, das das zu nutzende Paket sowie verschiedene Einstellungen für den `mysqld`-Daemon festlegt.

`mysql-configuration` [Datentyp]
Der Datentyp, der die Konfiguration des *mysql-service-type* repräsentiert.

`mysql` (Vorgabe: *mariadb*)
Das Paketobjekt des MySQL-Datenbankservers; es kann entweder *mariadb* oder *mysql* sein.

Für MySQL wird bei der Aktivierung des Dienstes ein temporäres Administratorpasswort („root“-Passwort) angezeigt. Für MariaDB ist das „root“-Passwort leer.

`bind-address` (Vorgabe: `"127.0.0.1"`)
Auf welcher IP-Adresse auf Verbindungen gelauscht wird. Benutzen Sie `"0.0.0.0"`, wenn sich der Server an alle verfügbaren Netzwerkschnittstellen binden soll.

`port` (Vorgabe: `3306`)
Der TCP-Port, auf dem der Datenbankserver auf eingehende Verbindungen lauscht.

`socket` (Vorgabe: `"/run/mysqld/mysqld.sock"`)
: Welche Socket-Datei für lokale Verbindungen (die also nicht über ein Netzwerk laufen) benutzt wird.

`extra-content` (Vorgabe: `""`)
: Weitere Einstellungen für die Konfigurationsdatei `my.cnf`.

`extra-environment` (Vorgabe: `#~'()`)
: Welche Liste von Umgebungsvariablen dem `mysqld`-Prozess mitgegeben wird.

`auto-upgrade?` (Vorgabe: `#t`)
: Ob nach dem Starten des Dienstes `mysql_upgrade` automatisch ausgeführt werden soll. Das ist nötig, um das *Systemschema* nach „großen“ Aktualisierungen auf den neuen Stand zu bringen (etwa beim Umstieg von MariaDB 10.4 auf 10.5), aber Sie können es abschalten, wenn Sie das lieber manuell vornehmen.

Memcached

`memcached-service-type` [Scheme-Variable]
Dies ist der Diensttyp für den Memcached-Dienst (`https://memcached.org/`), der einen verteilten Zwischenspeicher im Arbeitsspeicher (einen „In-Memory-Cache“) zur Verfügung stellt. Der Wert dieses Dienstes ist ein `memcached-configuration`-Objekt.

```
(service memcached-service-type)
```

`memcached-configuration` [Datentyp]
Der Datentyp, der die Konfiguration von memcached repräsentiert.

`memcached` (Vorgabe: `memcached`)
: Das Memcached-Paket, das benutzt werden soll.

`interfaces` (Vorgabe: `'("0.0.0.0")`)
: Auf welchen Netzwerkschnittstellen gelauscht werden soll.

`tcp-port` (Vorgabe: `11211`)
: Der Port, auf dem Verbindungen akzeptiert werden.

`udp-port` (Vorgabe: `11211`)
: Der Port, auf dem UDP-Verbindungen akzeptiert werden. Ist der Wert 0, wird *nicht* auf einem UDP-Socket gelauscht.

`additional-options` (Vorgabe: `'()`)
: Zusätzliche Befehlszeilenoptionen, die an `memcached` übergeben werden.

Redis

`redis-service-type` [Scheme-Variable]
Dies ist der Diensttyp für den Schlüssel-/Wert-Speicher Redis (`https://redis.io/`), dessen Wert ein `redis-configuration`-Objekt ist.

`redis-configuration` [Datentyp]
Der Datentyp, der die Konfiguration von redis repräsentiert.

`redis` (Vorgabe: `redis`)
Das zu benutzende Redis-Paket.

`bind` (Vorgabe: `"127.0.0.1"`)
Die Netzwerkschnittstelle, auf der gelauscht wird.

`port` (Vorgabe: `6379`)
Der Port, auf dem Verbindungen akzeptiert werden. Ist der Wert 0, wird das Lauschen auf einem TCP-Socket deaktiviert.

`working-directory` (Vorgabe: `"/var/lib/redis"`)
Das Verzeichnis, in dem die Datenbank und damit zu tun habende Dateien gespeichert werden.

12.9.12 Mail-Dienste

Das Modul `(gnu services mail)` stellt Guix-Dienstdefinitionen für E-Mail-Dienste zur Verfügung: IMAP-, POP3- und LMTP-Server sowie Mail Transport Agents (MTAs). Jede Menge Akronyme! Auf diese Dienste wird in den folgenden Unterabschnitten im Detail eingegangen.

Dovecot-Dienst

`dovecot-service` [*#:config* (*dovecot-configuration*)] [Scheme-Prozedur]
Liefert einen Dienst, der den IMAP-/POP3-/LMTP-Mailserver Dovecot ausführt.

Normalerweise muss für Dovecot nicht viel eingestellt werden; das vorgegebene Konfigurationsobjekt, das mit `(dovecot-configuration)` erzeugt wird, wird genügen, wenn Ihre Mails in `~/Maildir` gespeichert werden. Ein selbstsigniertes Zertifikat wird für durch TLS geschützte Verbindungen generiert, aber Dovecot lauscht nach Vorgabe auch auf unverschlüsselten Ports. Es gibt jedoch eine Reihe von Optionen, die Mail-Administratoren unter Umständen ändern müssen, was Guix – wie auch bei anderen Diensten – mit einer einheitlichen Scheme-Schnittstelle möglich macht.

Um zum Beispiel anzugeben, dass sich Mails in `maildir~/.mail` befinden, würde man den Dovecot-Dienst wie folgt instanziieren:

```
(dovecot-service #:config
                 (dovecot-configuration
                  (mail-location "maildir:~/.mail")))
```

Im Folgenden sehen Sie die verfügbaren Konfigurationsparameter. Jeder Parameterdefinition ist ihr Typ vorangestellt; zum Beispiel bedeutet ‘`Zeichenketten-Liste foo`’, dass der Parameter `foo` als eine Liste von Zeichenketten angegeben werden sollte. Es ist auch möglich, die Konfiguration als Zeichenkette anzugeben, wenn Sie eine alte `dovecot.conf`-Datei haben, die Sie von einem anderen System übernehmen möchten; am Ende finden Sie mehr Details dazu.

Verfügbare `dovecot-configuration`-Felder sind:

„`package`“ `dovecot` [`dovecot-configuration`-Parameter]
Das Dovecot-Paket.

`Kommagetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`listen`

Eine Liste der IPs oder der Rechnernamen („Hosts“), auf denen auf Verbindungen gelauscht wird. ‘`*`’ bedeutet, auf allen IPv4-Schnittstellen zu lauschen; ‘`::`’ lässt auf allen IPv6-Schnittstellen lauschen. Wenn Sie nicht der Vorgabe entsprechende Ports oder komplexere Einstellungen festlegen möchten, sollten Sie die Adress- und Portfelder des ‘`inet-listener`’ beim jeweiligen Dienst anpassen, für den Sie sich interessieren.

`„protocol-configuration“-Liste` [`dovecot-configuration`-Parameter]
`protocols`

Die Liste der Protokolle, die angeboten werden sollen. Zu den verfügbaren Protokollen gehören ‘`imap`’, ‘`pop3`’ und ‘`lmtp`’.

Verfügbare `protocol-configuration`-Felder sind:

`Zeichenkette name` [`protocol-configuration`-Parameter]

Der Name des Protokolls.

`Zeichenkette auth-socket-path` [`protocol-configuration`-Parameter]

Der Pfad des UNIX-Sockets zum Hauptauthentifizierungsserver, um Benutzer zu finden. Er wird von imap (für geteilte Benutzer) und lda benutzt. Die Vorgabe ist ‘`"/var/run/dovecot/auth-userdb"`’.

`Boolescher-Ausdruck` [`protocol-configuration`-Parameter]
`imap-metadata?`

Ob die Erweiterung `IMAP METADATA`, definiert in RFC 5464 (`https://tools.ietf.org/html/rfc5464`), aktiviert werden soll, mit der Clients einem Postfach („Mailbox“) Metadaten und Annotationen über IMAP zuweisen und sie abfragen können.

Wenn dies ‘`#t`’ ist, müssen Sie auch über das Feld `mail-attribute-dict` das zu nutzende Dictionary (eine Schlüssel-Wert-Datenbank) angeben.

Vorgegeben ist ‘`#f`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`protocol-configuration`-Parameter]
`managesieve-notify-capabilities`

Welche NOTIFY-Capabilitys an Clients gemeldet werden, die sich mit dem ManageSieve-Dienst verbinden, bevor sie sich authentisiert haben. Sie dürfen sich von den Capabilitys unterscheiden, die angemeldeten Nutzern angeboten werden. Wenn dieses Feld leer gelassen wird, wird nach Vorgabe alles angegeben, was der Sieve-Interpretierer unterstützt.

Die Vorgabe ist ‘`()`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`protocol-configuration`-Parameter]
`managesieve-sieve-capability`

Welche SIEVE-Capabilitys an Clients gemeldet werden, die sich mit dem ManageSieve-Dienst verbinden, bevor sie sich authentisiert haben. Sie dürfen sich von den Capabilitys unterscheiden, die angemeldeten Nutzern angeboten werden. Wenn dieses Feld leer gelassen wird, wird nach Vorgabe alles angegeben, was der Sieve-Interpretierer unterstützt.

Die Vorgabe ist ‘`()`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`protocol-configuration`-Parameter]
`mail-plugins`
Leerzeichengetrennte Liste der zu ladenden Plugins.

`Nichtnegative-ganze-Zahl` [`protocol-configuration`-Parameter]
`mail-max-userip-connections`
Die Maximalzahl der IMAP-Verbindungen, die jeder Nutzer von derselben IP-Adresse aus benutzen kann. *Anmerkung*: Beim Vergleichen des Benutzernamens wird Groß- und Kleinschreibung unterschieden. Die Vorgabe ist '`10`'.

`„service-configuration“-Liste` [`dovecot-configuration`-Parameter]
`services`
Die Liste der zu aktivierenden Dienste. Zu den verfügbaren Diensten gehören '`imap`', '`imap-login`', '`pop3`', '`pop3-login`', '`auth`' und '`lmtp`'.

Verfügbare `service-configuration`-Felder sind:

`Zeichenkette kind` [`service-configuration`-Parameter]
Die Dienstart (englisch „kind“). Zu den gültigen Werten gehören `director`, `imap-login`, `pop3-login`, `lmtp`, `imap`, `pop3`, `auth`, `auth-worker`, `dict`, `tcpwrap`, `quota-warning` oder alles andere.

`„listener-configuration“-Liste` [`service-configuration`-Parameter]
`listeners`
„Listener“ für den Dienst, also Lauscher auf neue Verbindungen. Als Listener kann entweder eine `unix-listener-configuration`, eine `fifo-listener-configuration` oder eine `inet-listener-configuration` angegeben werden. Die Vorgabe ist '`()`'.

Verfügbare `unix-listener-configuration`-Felder sind:

`Zeichenkette path` [`unix-listener-configuration`-Parameter]
Der Pfad zur Datei, relativ zum Feld `base-dir`. Er wird auch als der Abschnittsname verwendet.

`Zeichenkette mode` [`unix-listener-configuration`-Parameter]
Der Zugriffsmodus des Sockets. Die Vorgabe ist '`"0600"`'.

`Zeichenkette user` [`unix-listener-configuration`-Parameter]
Der Benutzer, dem der Socket gehört. Die Vorgabe ist '`""`'.

`Zeichenkette group` [`unix-listener-configuration`-Parameter]
Die Gruppe, der der Socket gehört. Die Vorgabe ist '`""`'.

Verfügbare `fifo-listener-configuration`-Felder sind:

`Zeichenkette path` [`fifo-listener-configuration`-Parameter]
Der Pfad zur Datei, relativ zum Feld `base-dir`. Er wird auch als der Abschnittsname verwendet.

`Zeichenkette mode` [`fifo-listener-configuration`-Parameter]
Der Zugriffsmodus des Sockets. Die Vorgabe ist '`"0600"`'.

`Zeichenkette user` [`fifo-listener-configuration`-Parameter]
Der Benutzer, dem der Socket gehört. Die Vorgabe ist '`""`'.

`Zeichenkette group` [`fifo-listener-configuration`-Parameter]
Die Gruppe, der der Socket gehört. Die Vorgabe ist '`""`'.

Verfügbare `inet-listener-configuration`-Felder sind:

`Zeichenkette protocol` [`inet-listener-configuration`-Parameter]
Das Protokoll, auf das gelauscht wird.

`Zeichenkette address` [`inet-listener-configuration`-Parameter]
Die Adresse, auf der gelauscht wird. Bleibt das Feld leer, wird auf allen Adressen gelauscht. Die Vorgabe ist '`""`'.

`Nichtnegative-ganze-Zahl port` [`inet-listener-configuration`-Parameter]
Der Port, auf dem gelauscht werden soll.

`Boolescher-Ausdruck ssl?` [`inet-listener-configuration`-Parameter]
Ob für diesen Dienst SSL benutzt werden kann: '`yes`' für ja, '`no`' für nein oder '`required`' für „verpflichtend“. Die Vorgabe ist '`#t`'.

`Nichtnegative-ganze-Zahl client-limit` [`service-configuration`-Parameter]
Die maximale Anzahl gleichzeitiger Verbindungen mit Clients pro Prozess. Sobald diese Anzahl von Verbindungen eingegangen ist, bewirkt das Eingehen der nächsten Verbindung, dass Dovecot einen weiteren Prozess startet. Ist sie auf 0 gesetzt, benutzt Dovecot stattdessen `default-client-limit`.

Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl service-count` [`service-configuration`-Parameter]
Die Anzahl Verbindungen, die behandelt werden, bevor ein neuer Prozess gestartet wird. Typischerweise sind die einzig sinnvollen Werte 0 (unbeschränkt) oder 1. 1 ist sicherer, aber 0 ist schneller. Siehe <doc/wiki/LoginProcess.txt>.
Die Vorgabe ist '`1`'.

`Nichtnegative-ganze-Zahl process-limit` [`service-configuration`-Parameter]
Die maximale Anzahl von Prozessen, die für diesen Dienst existieren können. Wenn sie auf 0 gesetzt ist, benutzt Dovecot stattdessen `default-process-limit`.

Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl process-min-avail` [`service-configuration`-Parameter]
Die Anzahl der Prozesse, mit denen immer auf neue Verbindungen gewartet wird. Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl` [`service-configuration`-Parameter]
`vsz-limit`
Wenn Sie ‘`service-count 0`’ festlegen, müssen Sie hierfür wahrscheinlich eine größere Zahl wählen. Die Vorgabe ist ‘`256000000`’.

`dict-configuration dict` [`dovecot-configuration`-Parameter]
Die Wörterbuchkonfiguration, wie sie der `dict-configuration`-Konstruktor erzeugt.

Verfügbare `dict-configuration`-Felder sind:

`Formlose-Felder entries` [`dict-configuration`-Parameter]
Eine Liste von Schlüssel-Wert-Paaren, die in diesem Wörterbuch enthalten sein sollen. Die Vorgabe ist ‘`()`’.

`„passdb-configuration“-Liste passdbs` [`dovecot-configuration`-Parameter]
Eine Liste von Passwortdatenbankkonfigurationen, die jeweils mit dem `passdb-configuration`-Konstruktor erzeugt werden.

Verfügbare `passdb-configuration`-Felder sind:

`Zeichenkette driver` [`passdb-configuration`-Parameter]
Der Treiber, den die Passwortdatenbank benutzen soll. Zu den gültigen Werten gehören ‘`pam`’, ‘`passwd`’, ‘`shadow`’, ‘`bsdauth`’ und ‘`static`’. Die Vorgabe ist ‘`"pam"`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`passdb-configuration`-Parameter]
`args`
Leerzeichengetrennte Liste der Argumente an den Passwortdatenbanktreiber. Die Vorgabe ist ‘`""`’.

`„userdb-configuration“-Liste userdbs` [`dovecot-configuration`-Parameter]
Liste der Benutzerdatenbankkonfigurationen, die jeweils mit dem `userdb-configuration`-Konstruktor erzeugt werden.

Verfügbare `userdb-configuration`-Felder sind:

`Zeichenkette driver` [`userdb-configuration`-Parameter]
Der Treiber, den die Benutzerdatenbank benutzen soll. Zu den gültigen Werten gehören ‘`passwd`’ und ‘`static`’. Die Vorgabe ist ‘`"passwd"`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`userdb-configuration`-Parameter]
`args`
Leerzeichengetrennte Liste der Argumente an den Benutzerdatenbanktreiber. Die Vorgabe ist ‘`""`’.

`Formlose-Argumente` [`userdb-configuration`-Parameter]
`override-fields`
Einträge, die Vorrang vor den Feldern aus passwd haben. Die Vorgabe ist ‘`()`’.

`„plugin-configuration“` [`dovecot-configuration`-Parameter]
`plugin-configuration`
Die Plugin-Konfiguration, die vom `plugin-configuration`-Konstruktor erzeugt wird.

„namespace-configuration“-Liste namespaces [dovecot-configuration-Parameter]

Liste der Namensräume. Jedes Objekt in der Liste wird durch den namespace-configuration-Konstruktor erzeugt.

Verfügbare namespace-configuration-Felder sind:

Zeichenkette name [namespace-configuration-Parameter]

Der Name dieses Namensraums.

Zeichenkette type [namespace-configuration-Parameter]

Namensraum-Typ: ‘private’, ‘shared’ oder ‘public’. Die Vorgabe ist ‘"private"’.

Zeichenkette separator [namespace-configuration-Parameter]

Welche Trennzeichen in der Hierarchie von Namensräumen benutzt werden sollen. Sie sollten denselben Trenner für alle Namensräume benutzen, sonst führt es zu Verwirrung bei manchen Clients. Meistens ist ‘/’ eine gute Wahl, die Voreinstellung ist allerdings abhängig vom darunterliegenden Mail-Speicher-Format. Die Vorgabe ist ‘""’.

Zeichenkette prefix [namespace-configuration-Parameter]

Das Präfix, das für Zugang auf diesen Namensraum angegeben werden muss. Es muss für jeden Namensraum ein anderes gewählt werden. Ein Beispiel ist ‘Public/’. Die Vorgabe ist ‘""’.

Zeichenkette location [namespace-configuration-Parameter]

Der physische Ort, an dem sich dieses Postfach („Mailbox“) befindet. Das Format ist dasselbe wie beim Mail-Ort („mail location“), der auch als Voreinstellung hierfür benutzt wird. Die Vorgabe ist ‘""’.

Boolescher-Ausdruck inbox? [namespace-configuration-Parameter]

Es kann nur eine INBOX geben; hiermit wird festgelegt, zu welchem Namensraum sie gehört. Die Vorgabe ist ‘#f’.

Boolescher-Ausdruck hidden? [namespace-configuration-Parameter]

Wenn der Namensraum versteckt ist, wird er Clients gegenüber *nicht* über die NAMESPACE-Erweiterung mitgeteilt. Wahrscheinlich möchten Sie auch ‘list? #f’ festlegen. Das ist in erster Linie dann nützlich, wenn Sie von einem anderen Server mit anderen Namensräumen umziehen, die Sie ersetzen möchten, die aber trotzdem noch weiterhin funktionieren sollen. Zum Beispiel können Sie versteckte Namensräume mit Präfixen ‘~/mail/’, ‘~%u/mail/’ und ‘mail/’ anlegen. Die Vorgabe ist ‘#f’.

Boolescher-Ausdruck list? [namespace-configuration-Parameter]

Ob die Postfächer („Mailboxes“) unter diesem Namensraum mit dem LIST-Befehl angezeigt werden können. Dadurch wird der Namensraum für Clients sichtbar, die die NAMESPACE-Erweiterung nicht unterstützen. Mit dem besonderen Wert children werden auch Kind-Postfächer aufgelistet, aber das Namensraumpräfix verborgen. Die Vorgabe ist ‘#t’.

`Boolescher-Ausdruck` [`namespace-configuration`-Parameter]
`subscriptions?`

Die Abonnements werden im Namensraum selbst behandelt. Wenn es auf `#f` gesetzt ist, werden sie im Elternnamensraum behandelt. Das leere Präfix sollte hier immer `#t` haben. Die Vorgabe ist '`#t`'.

`„mailbox-configuration“-Liste` [`namespace-configuration`-Parameter]
`mailboxes`

Die Liste der in diesem Namensraum vordefinierten Postfächer. Die Vorgabe ist '`()`'.

Verfügbare `mailbox-configuration`-Felder sind:

`Zeichenkette name` [`mailbox-configuration`-Parameter]

Der Name dieses Postfachs (dieser „Mailbox“).

`Zeichenkette auto` [`mailbox-configuration`-Parameter]

Bei '`create`' wird dieses Postfach automatisch erzeugt. Bei '`subscribe`' wird dieses Postfach sowohl automatisch erzeugt als auch automatisch abonniert. Die Vorgabe ist '`"no"`'.

`Leerzeichengetrennte-Zeichenketten-Liste` [`mailbox-configuration`-Parameter]
`special-use`

Liste der `SPECIAL-USE`-Attribute von IMAP, wie sie im RFC 6154 festgelegt wurden. Gültige Werte sind `\All`, `\Archive`, `\Drafts`, `\Flagged`, `\Junk`, `\Sent` und `\Trash`. Die Vorgabe ist '`()`'.

`Dateiname base-dir` [`dovecot-configuration`-Parameter]

Das Basisverzeichnis, in dem Laufzeitdaten gespeichert werden. Die Vorgabe ist '`"/var/run/dovecot/"`'.

`Zeichenkette login-greeting` [`dovecot-configuration`-Parameter]

Begrüßungsnachricht für Clients. Die Vorgabe ist '`"Dovecot ready."`'.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`login-trusted-networks`

Die Liste der Netzwerkbereiche, denen vertraut wird. Für Verbindungen von diesen IP-Adressen können abweichende IP-Adressen und Ports angegeben werden (zur Protokollierung und zur Authentifizierung). '`disable-plaintext-auth`' wird für diese Netzwerke außerdem ignoriert. Normalerweise würden Sie hier Ihre IMAP-Proxy-Server eintragen. Die Vorgabe ist '`()`'.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`login-access-sockets`

Die Liste der Sockets zur Zugriffsprüfung beim Anmelden (z.B. tcpwrap). Die Vorgabe ist '`()`'.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`verbose-proctitle?`

Ausführlichere Prozessnamen anzeigen (mit „ps“). Nach Voreinstellung werden Benutzername und IP-Adresse angezeigt. Die Einstellung ist nützlich, wenn man sehen können will, wer tatsächlich IMAP-Prozesse benutzt (z.B. gemeinsam genutzte

Postfächer oder wenn derselbe Benutzeridentifikator/„UID“ für mehrere Konten benutzt wird). Die Vorgabe ist ‘`#f`’.

`Boolescher-Ausdruck shutdown-clients?` [`dovecot-configuration`-Parameter]
Ob alle Prozesse abgewürgt werden sollen, wenn der Haupt-Dovecot-Prozess terminiert. Ist dies auf `#f` gesetzt, kann Dovecot aktualisiert werden, ohne dass bestehende Client-Verbindungen zwangsweise geschlossen werden (jedoch kann das problematisch sein, wenn die Aktualisierung z.B. eine Sicherheitslücke schließen soll). Die Vorgabe ist ‘`#t`’.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`doveadm-worker-count`
Ist dies *nicht* null, werden Mail-Befehle über die angegebene Anzahl von Verbindungen an den doveadm-Server geschickt, statt sie direkt im selben Prozess auszuführen. Die Vorgabe ist ‘`0`’.

`Zeichenkette doveadm-socket-path` [`dovecot-configuration`-Parameter]
Der UNIX-Socket oder das „Host:Port“-Paar, mit dem Verbindungen zum doveadm-Server hergestellt werden. Die Vorgabe ist ‘`"doveadm-server"`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`import-environment`
Die Liste der Umgebungsvariablen, die beim Starten von Dovecot erhalten bleiben und allen Kindprozessen davon mitgegeben werden. Sie können auch „Schlüssel=Wert“-Paare angeben, um wie immer bestimmte Zuweisungen festzulegen.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`disable-plaintext-auth?`
Deaktiviert den LOGIN-Befehl und alle anderen Klartext-Authentisierungsverfahren, solange kein SSL/TLS benutzt wird (die LOGINDISABLED-Capability). Beachten Sie, dass, wenn die entfernte IP-Adresse mit der lokalen IP-Adresse identisch ist (Sie sich also vom selben Rechner aus verbinden), die Verbindung als sicher aufgefasst und Klartext-Authentisierung möglich ist. Siehe auch die „ssl=required“-Einstellung. Die Vorgabe ist ‘`#t`’.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`auth-cache-size`
Die Größe des Zwischenspeichers für Authentifizierungen (z.B. ‘`#e10e6`’). Bei 0 ist er deaktiviert. Beachten Sie, dass für bsdauth, PAM und vpopmail die Einstellung ‘`cache-key`’ festgelegt sein muss, damit ein Zwischenspeicher benutzt wird. Die Vorgabe ist ‘`0`’.

`Zeichenkette auth-cache-ttl` [`dovecot-configuration`-Parameter]
Wie lange Daten im Zwischenspeicher gültig bleiben („Time to live“). Nachdem die TTL ausläuft, wird der zwischengespeicherte Eintrag nicht mehr benutzt, *außer* wenn eine Auflösung über die Hauptdatenbank zu einem internen Fehler führt. Dovecot versucht zudem, Passwortänderungen automatisch zu behandeln: Wenn die letzte Authentisierung erfolgreich war, diese jetzt aber nicht, wird der Zwischenspeicher *nicht* benutzt. Derzeit funktioniert dies nur bei Klartext-Authentisierung. Die Vorgabe ist ‘`"1 hour"`’ für 1 Stunde.

`Zeichenkette auth-cache-negative-ttl` [`dovecot-configuration`-Parameter]
TTL beim Zwischenspeichern negativer Ergebnisse („negative Hits“, z.B. wenn der Benutzer nicht gefunden wurde oder das Passwort nicht stimmt). 0 deaktiviert das Zwischenspeichern davon vollständig. Die Vorgabe ist ‘`"1 hour"`’ für 1 Stunde.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`auth-realms`
Die Liste der Administrationsbereiche („Realms“) für SASL-Authentisierungsmechanismen, die solche benötigen. Sie können dieses Feld leer lassen, wenn Sie *keine* Unterstützung für mehrere Administrationsbereiche wollen. Viele Clients benutzen den ersten hier aufgelisteten Administrationsbereich, also sollte der als Voreinstellung gewünschte Bereich vorne stehen. Die Vorgabe ist ‘`()`’.

`Zeichenkette auth-default-realm` [`dovecot-configuration`-Parameter]
Der voreingestellte Administrationsbereich bzw. die Domain, falls keiner angegeben wurde. Dies wird sowohl für SASL-Administrationsbereiche als auch zum Anhängen von @domain an den Benutzernamen bei Klartextanmeldungen benutzt. Die Vorgabe ist ‘`""`’.

`Zeichenkette auth-username-chars` [`dovecot-configuration`-Parameter]
Die Liste der in Benutzernamen zulässigen Zeichen. Wenn der vom Benutzer angegebene Benutzername ein hier nicht aufgelistetes Zeichen enthält, wird die Authentisierung automatisch abgelehnt. Dies dient bloß als eine weitere Überprüfung, um zu gewährleisten, dass mögliche Schwachstellen bei der Maskierung von Anführungszeichen in SQL-/LDAP-Datenbanken nicht ausgenutzt werden können. Wenn Sie alle Zeichen zulassen möchten, setzen Sie dieses Feld auf eine leere Zeichenkette. Die Vorgabe ist ‘`"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890.-_@"`’.

`Zeichenkette` [`dovecot-configuration`-Parameter]
`auth-username-translation`
Wie Zeichen in Benutzernamen ersetzt werden sollen, bevor der Name mit Datenbanken aufgelöst wird. Der Wert besteht aus einer Reihe von Angaben, welches Zeichen durch welches zu ersetzen ist. Zum Beispiel werden für ‘`#@/@`’ die Zeichen ‘`#`’ und ‘`/`’ beide durch ‘`@`’ ersetzt. Die Vorgabe ist ‘`""`’.

`Zeichenkette auth-username-format` [`dovecot-configuration`-Parameter]
Formatierungsanweisungen, die auf jeden Benutzernamen angewandt werden, bevor er mit einer Datenbank aufgelöst wird. Sie können hierzu die Standardvariablen verwenden, z.B. würde %Lu den Benutzernamen in Kleinbuchstaben umschreiben („lowercase“), %n würde den Domainnamen weglassen, wenn einer angegeben wurde, und ‘`%n-AT-%d`’ würde alle ‘`@`’ durch ‘`-AT-`’ ersetzen. Diese Übersetzung wird durchgeführt, nachdem die Änderungen aus ‘`auth-username-translation`’ angewandt wurden. Die Vorgabe ist ‘`"%Lu"`’.

`Zeichenkette` [`dovecot-configuration`-Parameter]
`auth-master-user-separator`
Wenn Sie es für „Master“-Benutzer erlauben möchten, sich durch Angeben des Master-Benutzernamens als Teil einer normalen Benutzernamens-Zeichenkette als jemand

anders anzumelden (also ohne Verwendung der Unterstützung davon durch den SASL-Mechanismus), können Sie hier die Trennzeichen dazwischen angeben. Das Format ist dann <Benutzername><Trenner><Master-Benutzername>. UW-IMAP benutzt '*' als Trennzeichen, also könnte das eine gute Wahl sein. Die Vorgabe ist '""'.

Zeichenkette auth-anonymous-username [dovecot-configuration-Parameter]
Der Benutzername, der verwendet wird, wenn sich Benutzer mit dem SASL-Mechanismus „ANONYMOUS" anmelden. Die Vorgabe ist '"anonymous"'.

Nichtnegative-ganze-Zahl auth-worker-max-count [dovecot-configuration-Parameter]
Die maximale Anzahl von dovecot-auth-Arbeiterprozessen. Diese werden benutzt, um blockierende Anfragen an die Passwortdatenbank („passdb") und an die Benutzerdatenbank („userdb") zu stellen (z.B. MySQL und PAM). Sie werden automatisch erzeugt und gelöscht, je nachdem, wann sie gebraucht werden. Die Vorgabe ist '30'.

Zeichenkette auth-gssapi-hostname [dovecot-configuration-Parameter]
Der Rechnername, der in GSSAPI-Prinzipalnamen benutzt wird. Nach Voreinstellung wird der durch gethostname() zurückgelieferte Name verwendet. Benutzen Sie '$ALL' (mit Anführungszeichen), damit alle Einträge in der Schlüsseltabelle („Keytab") akzeptiert werden. Die Vorgabe ist '""'.

Zeichenkette auth-krb5-keytab [dovecot-configuration-Parameter]
Kerberos-Schlüsseltabelle („Keytab"), die für den GSSAPI-Mechanismus benutzt werden soll. Wenn sie nicht angegeben wird, wird die Voreinstellung des Systems benutzt (in der Regel /etc/krb5.keytab). Eventuell müssen Sie den Auth-Dienst als Administratornutzer „root" ausführen lassen, um Lesezugriffe auf diese Datei zu ermöglichen. Die Vorgabe ist '""'.

Boolescher-Ausdruck auth-use-winbind? [dovecot-configuration-Parameter]
NTLM-Authentifizierung und GSS-SPNEGO-Authentifizierung mit dem winbind-Daemon und dem 'ntlm-auth'-Hilfsprogramm von Samba durchführen. Siehe <doc/wiki/Authentication/Mechanisms/Winbind.txt>. Die Vorgabe ist '#f'.

Dateiname auth-winbind-helper-path [dovecot-configuration-Parameter]
Der Pfad zur Binärdatei 'ntlm-auth' von Samba. Die Vorgabe ist '"/usr/bin/ntlm_auth"'.

Zeichenkette auth-failure-delay [dovecot-configuration-Parameter]
Die Zeit, wie lange vor der Antwort auf eine fehlgeschlagene Authentisierung gewartet wird. Die Vorgabe ist '"2 secs"' für 2 Sekunden.

Boolescher-Ausdruck auth-ssl-require-client-cert? [dovecot-configuration-Parameter]
Es wird ein gültiges SSL-Client-Zertifikat verlangt, andernfalls schlägt die Authentisierung fehl. Die Vorgabe ist '#f'.

Boolescher-Ausdruck auth-ssl-username-from-cert? [dovecot-configuration-Parameter]
Ob der Benutzername aus dem SSL-Zertifikat des Clients ausgelesen werden soll, indem X509_NAME_get_text_by_NID() benutzt wird, um den Distinguished Name

(„DN“) als Gebräuchlicher Name („CommonName“) des Zertifikatinhabers („Subject“) auszulesen. Die Vorgabe ist ‘`#f`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`auth-mechanisms`
Die Liste der erwünschten Authentisierungsmechanismen. Unterstützte Mechanismen sind: ‘`plain`’, ‘`login`’, ‘`digest-md5`’, ‘`cram-md5`’, ‘`ntlm`’, ‘`rpa`’, ‘`apop`’, ‘`anonymous`’, ‘`gssapi`’, ‘`otp`’, ‘`skey`’ und ‘`gss-spnego`’. *Anmerkung*: Siehe auch die Einstellung zu ‘`disable-plaintext-auth`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`director-servers`
Die Liste der IP-Adressen oder Rechnernamen („Hostnames“) für alle Direktorserver, einschließlich dieses Servers selbst. Ports können wie in „IP:Port“ angegeben werden. Der voreingestellte Port ist derselbe wie der, der beim ‘`inet-listener`’ des Direktordienstes benutzt wird. Die Vorgabe ist ‘`()`’.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`director-mail-servers`
Die Liste der IP-Adressen oder Rechnernamen („Hostnames“), um alle Hintergrund-Mailserver zu erreichen. Auch Bereiche können angegeben werden, wie 10.0.0.10-10.0.0.30. Die Vorgabe ist ‘`()`’.

`Zeichenkette director-user-expire` [`dovecot-configuration`-Parameter]
Wie lange Benutzer zum selben Server weitergeleitet werden, sobald dieser keine Verbindungen mehr hat. Die Vorgabe ist ‘`"15 min"`’.

`Zeichenkette director-username-hash` [`dovecot-configuration`-Parameter]
Wie der Benutzername umgeschrieben wird, bevor er gehasht wird. Zu den sinnvollen Werten gehören %Ln, wenn der Nutzer sich mit oder ohne @domain anmelden kann, oder %Ld, wenn Postfächer innerhalb der Domain gemeinsam genutzt werden. Die Vorgabe ist ‘`"%Lu"`’.

`Zeichenkette log-path` [`dovecot-configuration`-Parameter]
Die Protokolldatei, die für Fehlermeldungen benutzt werden soll. Bei ‘`syslog`’ wird das Protokoll an syslog geschrieben, bei ‘`/dev/stderr`’ an die Standardfehlerausgabe. Die Vorgabe ist ‘`"syslog"`’.

`Zeichenkette info-log-path` [`dovecot-configuration`-Parameter]
Die Protokolldatei, die für Informationsmeldungen benutzt werden soll. Die Voreinstellung ist ‘`log-path`’. Die Vorgabe ist ‘`""`’.

`Zeichenkette debug-log-path` [`dovecot-configuration`-Parameter]
Die Protokolldatei, die für Meldungen zur Fehlersuche benutzt werden soll. Die Voreinstellung ist ‘`info-log-path`’. Die Vorgabe ist ‘`""`’.

`Zeichenkette syslog-facility` [`dovecot-configuration`-Parameter]
Als welche Syslog-Einrichtung Protokolle an Syslog übermittelt werden sollen. Falls Sie ‘`mail`’ hierbei *nicht* benutzen wollen, eignen sich normalerweise local0–local7. Andere Standardeinrichtungen werden auch unterstützt. Die Vorgabe ist ‘`"mail"`’.

`Boolescher-Ausdruck auth-verbose?` [`dovecot-configuration`-Parameter]
Ob gescheiterte Anmeldeversuche und die Gründe, warum diese nicht erfolgreich waren, protokolliert werden sollen. Die Vorgabe ist '`#f`'.

`Zeichenkette auth-verbose-passwords` [`dovecot-configuration`-Parameter]
Ob bei falschen Passwörtern das versuchte falsche Passwort ins Protokoll geschrieben werden soll. Gültige Werte sind `"no"` („nein"), `"plain"` (als Klartext) und `"sha1"`. Den SHA1-Hash zu speichern kann nützlich sein, um zu erkennen, wenn jemand versucht, sehr viele Passwörter durchzuprobieren (ein „Brute-Force"-Angriff) oder ob Benutzer einfach nur dasselbe Passwort immer wieder probieren. Sie können auch nur die ersten n Zeichen des Wertes protokollieren, indem Sie `":n"` anhängen (z.B. sha1:6). Die Vorgabe ist '`"no"`'.

`Boolescher-Ausdruck auth-debug?` [`dovecot-configuration`-Parameter]
Ob zur Fehlersuche noch ausführlichere Protokolle geschrieben werden sollen. Zum Beispiel werden SQL-Anfragen protokolliert. Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck auth-debug-passwords?` [`dovecot-configuration`-Parameter]
Ob bei falschen Passwörtern das versuchte falsche Passwort und das benutzte Passwortschema ins Protokoll geschrieben werden soll, damit das Problem untersucht werden kann. Wenn dies aktiviert wird, wird auch '`auth-debug`' aktiviert. Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck mail-debug?` [`dovecot-configuration`-Parameter]
Ob die Fehlersuche beim Mail-Prozess ermöglicht werden soll. Dies kann Ihnen dabei helfen, herauszufinden, warum Dovecot Ihre E-Mails nicht findet. Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck verbose-ssl?` [`dovecot-configuration`-Parameter]
SSL-Fehler auf Protokollebene anzeigen. Die Vorgabe ist '`#f`'.

`Zeichenkette log-timestamp` [`dovecot-configuration`-Parameter]
Das Präfix für jede Zeile, die ins Protokoll geschrieben wird. %-Codes sind im Format von strftime(3). Die Vorgabe ist '`"\"%b %d %H:%M:%S \""`'.

`Leerzeichengetrennte-Zeichenketten-Liste login-log-format-elements` [`dovecot-configuration`-Parameter]
Eine Liste der Elemente, die protokolliert werden sollen. Die Elemente, deren Variablenwerte nicht leer sind, werden zu einer kommagetrennten Zeichenkette zusammengefügt.

`Zeichenkette login-log-format` [`dovecot-configuration`-Parameter]
Das Format des Anmeldeprogramms. %s umfasst die Zeichenkette '`login-log-format-elements`', %$ enthält die Daten, die man protokollieren lassen möchte. Die Vorgabe ist '`"%$: %s"`'.

`Zeichenkette mail-log-prefix` [`dovecot-configuration`-Parameter]
Das Präfix, das Protokollen für Mailprozesse vorangestellt wird. Siehe doc/wiki/Variables.txt für eine Liste der benutzbaren Variablen. Die Vorgabe ist '`"\"%s(%u)<%{pid}><%{session}>: \""`'.

`Zeichenkette deliver-log-format` [`dovecot-configuration`-Parameter]
Welches Format zur Protokollierung von Mailzustellungen verwendet werden soll. Sie können die folgenden Variablen benutzen:

`%$`	Zustellungsstatusnachricht (z.B. '`saved to INBOX`')
`%m`	Nachrichtenidentifikator („Message-ID“)
`%s`	Betreff („Subject“)
`%f`	Absendeadresse („From“)
`%p`	Physische Größe
`%w`	Virtuelle Größe.

Die Vorgabe ist '`"msgid=%m: %$"`'.

`Zeichenkette mail-location` [`dovecot-configuration`-Parameter]
Wo die Postfächer (die „Mailboxes“) der Benutzer gespeichert sind. Die Vorgabe ist die leere Zeichenkette, was bedeutet, dass Dovecot die Postfächer automatisch zu finden versucht. Das funktioniert nur, wenn der Nutzer bereits E-Mails gespeichert hat, also sollten Sie Dovecot den vollständigen Pfad mitteilen.

Wenn Sie das mbox-Format benutzen, genügt es *nicht*, den Pfad zur INBOX-Datei (z.B. `/var/mail/%u`) anzugeben. Sie müssen Dovecot *auch* mitteilen, wo die anderen Postfächer gespeichert sind. Dieses Verzeichnis nennt sich *Wurzelmailverzeichnis* („Root Mail Directory“) und es muss als erster Pfad in der '`mail-location`'-Einstellung angegeben werden.

Es gibt ein paar besondere Variable, die Sie verwenden können, z.B.:

'`%u`'	Benutzername
'`%n`'	Benutzerteil in Benutzer@Domain; sonst dasselbe wie %u, wenn es keine Domain gibt
'`%d`'	Domainteil in Benutzer@Domain; sonst leer, wenn es keine Domain gibt
'`%h`'	Persönliches Verzeichnis

Siehe doc/wiki/Variables.txt für die vollständige Liste. Einige Beispiele:

'`maildir:~/Maildir`'
'`mbox:~/mail:INBOX=/var/mail/%u`'
'`mbox:/var/mail/%d/%1n/%n:INDEX=/var/indexes/%d/%1n/%`'

Die Vorgabe ist '`""`'.

`Zeichenkette mail-uid` [`dovecot-configuration`-Parameter]
Systembenutzer und -gruppe, die benutzt werden sollen, um auf Mails zuzugreifen. Wenn Sie mehrere Benutzerkonten verwenden, kann auch die Benutzerdatenbank „userdb“ vorrangig verwendet werden, indem sie zu Benutzer- oder Gruppenidentifikatoren (UIDs und GIDs) auflöst. Sie können hier Zahlen oder Namen angeben. Siehe <doc/wiki/UserIds.txt>. Die Vorgabe ist '`""`'.

`Zeichenkette mail-gid` [`dovecot-configuration`-Parameter]
Die Vorgabe ist '`""`'.

`Zeichenkette mail-privileged-group` [`dovecot-configuration`-Parameter]
Die Benutzergruppe, die zwischenzeitlich benutzt wird, um Operationen mit besonderen Berechtigungen auszuführen. Derzeit wird dies nur mit dem INBOX-Postfach benutzt, wenn dessen anfängliche Erzeugung oder Sperrung per „Dotlocking“-Datei fehlschlägt. Typischerweise wird es auf '`"mail"`' gesetzt, damit Zugriffe auf `/var/mail` möglich sind. Die Vorgabe ist '`""`'.

`Zeichenkette mail-access-groups` [`dovecot-configuration`-Parameter]
Mail-Prozessen Zugriff auf diese zusätzlichen Benutzergruppen gewähren. Typischerweise werden sie benutzt, um gemeinsam genutzte Postfächer („Mailboxes“) so einzurichten, dass alle aus der Gruppe zugreifen können. Beachten Sie, dass es gefährlich sein kann, dies zu erlauben, wenn Benutzer symbolische Verknüpfungen einrichten können (z.B. kann jeder, wenn hier die '`mail`'-Gruppe festgelegt wurde, `ln -s /var/mail ~/mail/var` benutzen, um die Postfächer der anderen zu löschen, oder `ln -s /secret/shared/box ~/mail/mybox`, um sie zu lesen). Die Vorgabe ist '`""`'.

`Zeichenkette mail-attribute-dict` [`dovecot-configuration`-Parameter]
Der Ort, wo ein Dictionary (eine Schlüssel-Wert-Datenbank) zu finden ist, mit dem IMAP-Metadaten gespeichert werden, entsprechend deren Definition im RFC 5464 (`https://tools.ietf.org/html/rfc5464`).

Die IMAP-METADATA-Befehle stehen nur zur Verfügung, wenn in der Protokollkonfiguration von „imap“ das Feld `imap-metadata?` auf '`#t`' gesetzt ist.

Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`mail-full-filesystem-access?`
Clients vollen Dateisystemzugriff gestatten. Damit gibt es keine Zugriffsüberprüfungen mehr, abgesehen von denen, die das Betriebssystem für die aktiven Benutzer- und Gruppenidentifikatoren (UID und GID) durchführt. Es ist sowohl für maildir- als auch mbox-Formate verwendbar und Sie können dadurch für Namen von Postfächern („Mailboxes“) Präfixe wie z.B. `/pfad/` oder `~benutzer/` wählen. Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck mmap-disable?` [`dovecot-configuration`-Parameter]
Überhaupt kein `mmap()` benutzen. Das ist erforderlich, wenn Sie Indexe auf geteilten Dateisystemen speichern (wie NFS oder Cluster-Dateisystemen). Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck dotlock-use-excl?` [`dovecot-configuration`-Parameter]
Ob sich Dovecot darauf verlassen kann, dass '`O_EXCL`' funktioniert, wenn es Sperrdateien als „Dotlock“ erstellt. NFS unterstützt '`O_EXCL`' seit Version 3, also sollte es heutzutage kein Problem mehr sein, dies als Voreinstellung zu benutzen. Die Vorgabe ist '`#t`'.

`Zeichenkette mail-fsync` [`dovecot-configuration`-Parameter]
Wann fsync() oder fdatasync() aufgerufen werden soll:

`optimized`
Wann immer es nötig ist, um keine wichtigen Daten zu verlieren

`always` Praktisch bei z.B. NFS, wenn Schreibzugriffe mit `write()` verzögert sind

`never` Niemals benutzen (ist am schnellsten, aber Abstürze können zu Datenverlusten führen)

Die Vorgabe ist '`"optimized"`'.

`Boolescher-Ausdruck mail-nfs-storage?` [`dovecot-configuration`-Parameter]
Mails werden in NFS gespeichert. Setzen Sie dies auf ja, damit Dovecot NFS-Zwischenspeicher zurückschreiben kann, wann immer es nötig ist. Wenn Sie nur einen einzigen Mail-Server benutzen, brauchen Sie es *nicht*. Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck mail-nfs-index?` [`dovecot-configuration`-Parameter]
Ob die Index-Dateien für Mails auch in NFS gespeichert sind. Wenn dies auf ja gesetzt ist, muss '`mmap-disable? #t`' und '`fsync-disable? #f`' gesetzt sein. Die Vorgabe ist '`#f`'.

`Zeichenkette lock-method` [`dovecot-configuration`-Parameter]
Die Sperrmethode für Indexdateien. Die Alternativen sind fcntl, flock und dotlock. Bei Dotlocking werden ein paar Tricks benutzt, die mehr Plattenein- und -ausgaben als andere Sperrmethoden zur Folge haben. Für NFS-Benutzer gilt: flock funktioniert nicht, also denken Sie bitte daran, '`mmap-disable`' zu ändern. Die Vorgabe ist '`"fcntl"`'.

`Dateiname mail-temp-dir` [`dovecot-configuration`-Parameter]
Das Verzeichnis, in dem LDA/LMTP zwischenzeitlich eingehende E-Mails >128 kB speichert. Die Vorgabe ist '`"/tmp"`'.

`Nichtnegative-ganze-Zahl first-valid-uid` [`dovecot-configuration`-Parameter]
Der Bereich, in dem die Benutzerkennungen („UIDs“) von sich bei Dovecot anmeldenden Benutzern liegen müssen. Das dient hauptsächlich dazu, sicherzustellen, dass sich Anwender nicht mit den Benutzerkonten von Daemons oder sonstigen Systembenutzerkonten anmelden können. Beachten Sie, dass eine Anmeldung als Administrator „root“ grundsätzlich vom Code des Dovecot-Programms verboten wird und selbst dann *nicht* möglich ist, wenn Sie '`first-valid-uid`' auf 0 setzen. Die Vorgabe ist '`500`'.

`Nichtnegative-ganze-Zahl last-valid-uid` [`dovecot-configuration`-Parameter]
Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl first-valid-gid` [`dovecot-configuration`-Parameter]
Der Bereich, in dem die Gruppenkennungen („GIDs“) von sich bei Dovecot anmeldenden Benutzern liegen müssen. Benutzerkonten, deren primäre Gruppe keine gültige GID hat, können sich nicht anmelden. Wenn das Benutzerkonto zu zusätzlichen Gruppen mit ungültiger GID gehört, werden diese Gruppen-Berechtigungen von Dovecot wieder abgegeben. Die Vorgabe ist '`1`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`last-valid-gid`
Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`mail-max-keyword-length`
Die maximale zulässige Länge eines Mail-Schlüsselwort-Namens. Sie wirkt sich nur aus, wenn Sie neue Schlüsselwörter anzulegen versuchen. Die Vorgabe ist '`50`'.

`Doppelpunktgetrennte-Dateinamen-Liste` [`dovecot-configuration`-Parameter]
`valid-chroot-dirs`
Die Liste der Verzeichnisse, in die Mail-Prozesse per „chroot“ das Wurzelverzeichnis wechseln können (d.h. für `/var/mail` wird auch ein chroot nach `/var/mail/foo/bar` möglich). Diese Einstellung hat keinen Einfluss auf '`login-chroot`', '`mail-chroot`' oder Authentifizierungs-„chroot“-Einstellungen. Wenn diese Einstellung leer gelassen wird, werden chroots per '`/./`' in Persönlichen Verzeichnissen ignoriert. *Warnung*: Fügen Sie niemals Verzeichnisse hinzu, die lokale Benutzer verändern können, weil diese dann eventuell über eine Rechteausweitung Administratorrechte an sich reißen können. In der Regel sollte man ein solches Verzeichnis nur eintragen, wenn Nutzer keinen Shell-Zugriff erhalten können. Siehe <doc/wiki/Chrooting.txt>. Die Vorgabe ist '`()`'.

`Zeichenkette mail-chroot` [`dovecot-configuration`-Parameter]
Das voreingestellte „chroot“-Verzeichnis für Mail-Prozesse. Es kann für einzelne Nutzer in der Benutzerdatenbank außer Kraft gesetzt werden, indem '`/./`' als Teil der Angabe zum Persönlichen Verzeichnis des Benutzers verwendet wird (z.B. lässt '`/home/./benutzer`' das Wurzelverzeichnis per „chroot“ nach `/home` wechseln). Beachten Sie, dass es in der Regel *nicht* unbedingt notwendig ist, Chrooting zu betreiben, weil Dovecot es Benutzern ohnehin nicht erlaubt, auf Dateien außerhalb ihres Mail-Verzeichnisses zuzugreifen. Wenn Ihren Persönlichen Verzeichnissen das Chroot-Verzeichnis vorangestellt ist, sollten Sie '`/.`' an '`mail-chroot`' anhängen. Siehe <doc/wiki/Chrooting.txt>. Die Vorgabe ist '`""`'.

`Dateiname auth-socket-path` [`dovecot-configuration`-Parameter]
Der UNIX-Socket-Pfad, unter dem der Hauptauthentifizierungsserver zu finden ist, mit dem Nutzer gefunden werden können. Er wird von IMAP (für gemeinsame Benutzerkonten) und von LDA benutzt. Die Vorgabe ist '`"/var/run/dovecot/auth-userdb"`'.

`Dateiname mail-plugin-dir` [`dovecot-configuration`-Parameter]
Das Verzeichnis, in dem Mailplugins zu finden sind. Die Vorgabe ist '`"/usr/lib/dovecot"`'.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`mail-plugins`
Die Liste der Plugins, die für alle Dienste geladen werden sollen. Plugins, die nur für IMAP, LDA, etc. gedacht sind, werden in ihren eigenen .conf-Dateien zu dieser Liste hinzugefügt. Die Vorgabe ist '`()`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`mail-cache-min-mail-count`

Die kleinste Anzahl an Mails in einem Postfach, bevor Aktualisierungen an der Zwischenspeicherdatei vorgenommen werden. Damit kann das Verhalten von Dovecot optimiert werden, um weniger Schreibzugriffe auf die Platte durchzuführen, wofür allerdings mehr Lesezugriffe notwendig werden. Die Vorgabe ist '`0`'.

`Zeichenkette` [`dovecot-configuration`-Parameter]
`mailbox-idle-check-interval`

Während der IDLE-Befehl läuft, wird ab und zu im Postfach (der „Mailbox“) nachgeschaut, ob es neue Mails oder andere Änderungen gibt. Mit dieser Einstellung wird festgelegt, wie lange zwischen diesen Überprüfungen höchstens gewartet wird. Dovecot kann auch dnotify, inotify und kqueue benutzen, um sofort über Änderungen informiert zu werden. Die Vorgabe ist '`"30 secs"`'.

`Boolescher-Ausdruck mail-save-crlf?` [`dovecot-configuration`-Parameter]

Ob Mails mit CR+LF-Kodierung für Zeilenumbrüche statt einfacher LF gespeichert werden sollen. Dadurch wird das Versenden dieser Mails den Prozessor weniger beanspruchen, dies gilt besonders für den Systemaufruf sendfile() unter Linux und FreeBSD. Allerdings werden auch ein bisschen mehr Ein- und Ausgaben auf der Platte notwendig, wodurch es insgesamt langsamer werden könnte. Beachten Sie außerdem, dass andere Software, die die mboxes/maildirs ausliest, mit den CRs falsch umgehen und Probleme verursachen könnte. Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`maildir-stat-dirs?`

Nach Voreinstellung liefert der LIST-Befehl alle Einträge im Mailverzeichnis („Maildir“), die mit einem Punkt beginnen. Wenn diese Option aktiviert wird, liefert Dovecot nur solche Einträge, die für Verzeichnisse stehen. Dazu wird auf jedem Eintrag stat() aufgerufen, wodurch mehr Ein- und Ausgaben auf der Platte anfallen. (Bei Systemen, die einen Struktureintrag '`dirent->d_type`' machen, ist diese Überprüfung unnötig, daher werden dort nur Verzeichnisse geliefert, egal was hier eingestellt ist.) Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`maildir-copy-with-hardlinks?`

Ob zum Kopieren einer Nachricht statt einer Kopie so weit möglich harte Verknüpfungen („Hard Links“) verwendet werden sollen. Dadurch wird das System wesentlich weniger ausgelastet und Nebenwirkungen sind unwahrscheinlich. Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`maildir-very-dirty-syncs?`

Ob Dovecot annehmen darf, dass es der einzige MUA ist, der auf Maildir zugreift. Dann wird das cur/-Verzeichnis nur bei unerwarteten Änderungen an seiner mtime durchsucht oder wenn die Mail sonst nicht gefunden werden kann. Die Vorgabe ist '`#f`'.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`mbox-read-locks`

Welche Sperrmethoden zum Sperren des mbox-Postfachs (der „Mailbox“) benutzt werden. Es stehen vier Methoden zur Auswahl:

`dotlock` Hier wird eine Datei namens <Postfach>.lock erzeugt. Es handelt sich um die älteste und am ehesten mit NFS verwendbare Lösung. Wenn Sie ein Verzeichnis wie /var/mail/ benutzen, müssen die Benutzer Schreibzugriff darauf haben.

`dotlock-try`
Genau wie dotlock, aber wenn es mangels Berechtigungen fehlschlägt oder nicht genug Plattenplatz verfügbar ist, wird einfach nicht gesperrt.

`fcntl` Benutzen Sie diese Einstellung wenn möglich. Sie funktioniert auch mit NFS, sofern lockd benutzt wird.

`flock` Existiert vielleicht nicht auf allen Systemen. Funktioniert nicht mit NFS.

`lockf` Existiert vielleicht nicht auf allen Systemen. Funktioniert nicht mit NFS.

Sie können mehrere Sperrmethoden angeben; wenn ja, dann ist deren Reihenfolge entscheidend, um Verklemmungen („Deadlocks“) zu vermeiden, wenn andere MTAs/MUAs auch mehrere Sperrmethoden benutzen. Manche Betriebssysteme erlauben es nicht, manche davon gleichzeitig zu benutzen.

`Leerzeichengetrennte-Zeichenketten-Liste` [`dovecot-configuration`-Parameter]
`mbox-write-locks`

`Zeichenkette mbox-lock-timeout` [`dovecot-configuration`-Parameter]
Wie lange höchstens auf Sperren (irgendeiner Art) gewartet wird, bevor abgebrochen wird. Die Vorgabe ist ‘`"5 mins"`’.

`Zeichenkette` [`dovecot-configuration`-Parameter]
`mbox-dotlock-change-timeout`
Wenn eine Dotlock-Sperrdatei existiert, das Postfach (die „Mailbox“) aber auf keine Weise geändert wurde, wird die Sperrdatei nach der hier angegebenen Zeit außer Kraft gesetzt. Die Vorgabe ist ‘`"2 mins"`’.

`Boolescher-Ausdruck mbox-dirty-syncs?` [`dovecot-configuration`-Parameter]
Wenn sich das mbox-Postfach unerwartet ändert, müssen wir es gänzlich neu einlesen, um herauszufinden, was sich geändert hat. Wenn die mbox groß ist, kann das viel Zeit in Anspruch nehmen. Weil es sich bei der Änderung meistens nur um eine neu angefügte Mail handelt, wäre es schneller, nur die neuen Mails zu lesen. Wenn diese Einstellung hier aktiviert ist, arbeitet Dovecot nach dem eben beschriebenen Prinzip, liest aber doch die ganze mbox-Datei neu ein, sobald es etwas nicht wie erwartet vorfindet. Der einzige wirkliche Nachteil bei dieser Einstellung ist, dass es Dovecot nicht sofort erkennt, wenn ein anderer MUA die Statusindikatoren („Flags“) ändert. Beachten Sie, dass eine komplette Synchronisation bei den Befehlen SELECT, EXAMINE, EXPUNGE und CHECK durchgeführt wird. Die Vorgabe ist ‘`#t`’.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`mbox-very-dirty-syncs?`

Wie '`mbox-dirty-syncs`', aber ohne dass komplette Synchronisationen selbst bei den Befehlen SELECT, EXAMINE, EXPUNGE oder CHECK durchgeführt werden. Wenn dies hier aktiviert ist, wird '`mbox-dirty-syncs`' ignoriert. Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck mbox-lazy-writes?` [`dovecot-configuration`-Parameter]

Ob das Schreiben von mbox-Kopfzeilen hinausgezögert wird, bis eine komplette Schreibsynchronisation durchgeführt wird (bei den Befehlen EXPUNGE und CHECK, und beim Schließen des Postfachs, d.h. der „Mailbox“). Das wird besonders nützlich, wenn Clients POP3 verwenden, wo es oft vorkommt, dass die Clients alle Mails löschen. Der Nachteil ist, dass Dovecots Änderungen nicht sofort für andere MUAs sichtbar werden. Die Vorgabe ist '`#t`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`mbox-min-index-size`

Wenn die Größe des mbox-Postfaches kleiner als die angegebene Größe (z.B. 100k) ist, werden keine Index-Dateien geschrieben. Wenn bereits eine Index-Datei existiert, wird sie weiterhin gelesen und nur nicht aktualisiert. Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`mdbox-rotate-size`

Die maximale Größe der dbox-Datei, bis sie rotiert wird. Die Vorgabe ist '`10000000`'.

`Zeichenkette mdbox-rotate-interval` [`dovecot-configuration`-Parameter]

Das maximale Alter der dbox-Datei, bis sie rotiert wird. Typischerweise wird es in Tagen angegeben. Der Tag beginnt um Mitternacht, also steht 1d für heute, 2d für gestern, etc. 0 heißt, die Überprüfung ist abgeschaltet. Die Vorgabe ist '`"1d"`'.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`mdbox-preallocate-space?`

Ob beim Erstellen neuer mdbox-Postfachdateien gleich am Anfang eine Datei der Größe '`mdbox-rotate-size`' vorab angelegt werden soll. Diese Einstellung funktioniert derzeit nur mit Linux auf manchen Dateisystemen (ext4, xfs). Die Vorgabe ist '`#f`'.

`Zeichenkette mail-attachment-dir` [`dovecot-configuration`-Parameter]

Postfächer in den Formaten sdbox und mdbox unterstützen es, Mail-Anhänge in externen Dateien zu speichern, wodurch sie mit Einzelinstanz-Speicherung („Single-Instance Storage“) dedupliziert werden können. Andere Hintergrundsysteme („Backends“) bieten dafür noch keine Unterstützung.

Warnung: Diese Funktionalität wurde noch nicht ausgiebig getestet. Benutzen Sie sie auf eigene Gefahr.

Das Wurzelverzeichnis, in dem Mail-Anhänge gespeichert werden. Wenn es leer gelassen wird, ist es deaktiviert. Die Vorgabe ist '`""`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`mail-attachment-min-size`

Anhänge, die kleiner sind als hier angegeben, werden *nicht* extern gespeichert. Es ist auch möglich, ein Plugin zu schreiben, das externes Speichern bestimmter Anhänge deaktiviert. Die Vorgabe ist '`128000`'.

`Zeichenkette mail-attachment-fs` [`dovecot-configuration`-Parameter]
Ein Dateisystemhintergrundprogramm, mit dem Anhänge gespeichert werden:

`posix` Dovecot führt keine Einzelinstanzspeicherung durch (aber das Dateisystem kann so leichter selbst deduplizieren)

`sis posix` Einzelinstanzspeicherung wird durch einen sofortigen Byte-für-Byte-Vergleich beim Speichern umgesetzt

`sis-queue posix`
Einzelinstanzspeicherung mit verzögertem Vergleich und Deduplizierung.

Die Vorgabe ist '`"sis posix"`'.

`Zeichenkette mail-attachment-hash` [`dovecot-configuration`-Parameter]
Welches Hash-Format die Dateinamen von Anhängen bestimmt. Sie können beliebigen Text und Variable beifügen: `%{md4}`, `%{md5}`, `%{sha1}`, `%{sha256}`, `%{sha512}`, `%{size}`. Es können auch nur Teile der Variablen benutzt werden, z.B. liefert `%{sha256:80}` nur die ersten 80 Bits. Die Vorgabe ist '`"%{sha1}"`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`default-process-limit`
Die Vorgabe ist '`100`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`default-client-limit`
Die Vorgabe ist '`1000`'.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`default-vsz-limit`
Die vorgegebene Beschränkung der VSZ („Virtual Memory Size“, virtuelle Speichergröße) für Dienstprozesse. Dies ist hauptsächlich dafür gedacht, Prozessen, die ein Speicherleck aufweisen, rechtzeitig Einhalt zu gebieten und sie abzuwürgen, bevor sie allen Speicher aufbrauchen. Die Vorgabe ist '`256000000`'.

`Zeichenkette default-login-user` [`dovecot-configuration`-Parameter]
Der Anmeldebenutzer, der intern von Anmeldeprozessen benutzt wird. Der Anmeldebenutzer ist derjenige Benutzer im Dovecot-System, dem am wenigsten Vertrauen zugeschrieben wird. Er sollte auf überhaupt nichts Zugriff haben. Die Vorgabe ist '`"dovenull"`'.

`Zeichenkette default-internal-user` [`dovecot-configuration`-Parameter]
Der interne Benutzer, der von Prozessen ohne besondere Berechtigungen benutzt wird. Er sollte sich vom Anmeldebenutzer unterscheiden, damit Anmeldeprozesse keine anderen Prozesse stören können. Die Vorgabe ist '`"dovecot"`'.

`Zeichenkette ssl?` [`dovecot-configuration`-Parameter]
SSL/TLS-Unterstützung: yes für ja, no für nein, oder required, wenn SSL/TLS verpflichtend benutzt werden muss. Siehe <doc/wiki/SSL.txt>. Die Vorgabe ist '`"required"`'.

`Zeichenkette ssl-cert` [`dovecot-configuration`-Parameter]
Das PEM-kodierte X.509-SSL/TLS-Zertifikat (der öffentliche Schlüssel). Die Vorgabe ist '`"</etc/dovecot/default.pem"`'.

`Zeichenkette ssl-key` [`dovecot-configuration`-Parameter]
Der PEM-kodierte private Schlüssel für SSL/TLS. Der Schlüssel wird geöffnet, bevor Administratorrechte abgegeben werden, damit niemand außer dem Administratornutzer „root“ Lesezugriff auf die Schlüsseldatei hat. Die Vorgabe ist '`"</etc/dovecot/private/default.pem"`'.

`Zeichenkette ssl-key-password` [`dovecot-configuration`-Parameter]
Wenn die Schlüsseldatei passwortgeschützt ist, geben Sie hier das Passwort an. Alternativ können Sie es angeben, wenn sie Dovecot starten, indem Sie es mit dem Parameter -p übergeben. Da die Konfigurationsdatei oftmals allgemein lesbar ist, möchten Sie es vielleicht in einer anderen Datei ablegen. Die Vorgabe ist '`""`'.

`Zeichenkette ssl-ca` [`dovecot-configuration`-Parameter]
Die PEM-kodierte Zertifikatsautorität, die als vertrauenswürdig eingestuft wird. Legen Sie sie nur dann fest, wenn Sie '`ssl-verify-client-cert? #t`' setzen möchten. Die Datei sollte das oder die Zertifikat(e) der Zertifikatsautorität („Certificate Authority“, kurz CA) enthalten, gefolgt von den entsprechenden Zertifikatsperrlisten (CRLs), z.B. '`ssl-ca </etc/ssl/certs/ca.pem`'. Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck ssl-require-crl?` [`dovecot-configuration`-Parameter]
Ob die Prüfung der Client-Zertifikate gegen die Zertifikatsperrlisten (CRLs) erfolgreich sein muss. Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`ssl-verify-client-cert?`
Ob der Client gebeten wird, ein Zertifikat zu schicken. Wenn Sie es auch verpflichtend machen wollen, setzen Sie '`auth-ssl-require-client-cert? #t`' im Autorisierungsabschnitt. Die Vorgabe ist '`#f`'.

`Zeichenkette ssl-cert-username-field` [`dovecot-configuration`-Parameter]
Welches Feld im Zertifikat den Benutzernamen angibt. In der Regel wählt man den Gebräuchlichen Namen „commonName“ oder den Eindeutigen Identifikator „x500UniqueIdentifier“ als Benutzernamen, wenn man Client-Zertifikate benutzt. Sie müssen dann auch '`auth-ssl-username-from-cert? #t`' setzen. Die Vorgabe ist '`"commonName"`'.

`Zeichenkette ssl-min-protocol` [`dovecot-configuration`-Parameter]
Die kleinste Version des SSL-Protokolls, die noch akzeptiert werden soll. Die Vorgabe ist '`"TLSv1"`'.

`Zeichenkette ssl-cipher-list` [`dovecot-configuration`-Parameter]
Welche SSL-Ciphers benutzt werden dürfen. Die Vorgabe ist '`"ALL:!kRSA:!SRP:!kDHd:!DSS:!aNULL:!e`

`Zeichenkette ssl-crypto-device` [`dovecot-configuration`-Parameter]
Das SSL-Verschlüsselungsgerät („Crypto Device“), das benutzt werden soll. Gültige Werte bekommen Sie gezeigt, wenn Sie „openssl engine“ ausführen. Die Vorgabe ist ‘`""`’.

`Zeichenkette postmaster-address` [`dovecot-configuration`-Parameter]
An welche Adresse Mails versandt werden sollen, die über die Zurückweisung einer Mail informieren. %d wird zur Domain des Empfängers umgeschrieben. Die Vorgabe ist ‘`"postmaster@%d"`’.

`Zeichenkette hostname` [`dovecot-configuration`-Parameter]
Der Rechnername, der an mehreren Stellen in versandten E-Mails (z.B. im Nachrichtenidentifikator „Message-Id“) und in LMTP-Antworten benutzt wird. Die Voreinstellung entspricht dem wirklichen Rechnernamen des Systems. Die Vorgabe ist ‘`""`’.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`quota-full-tempfail?`
Ob bei einem Nutzer, der sein Kontingent überschreitet, ein temporärer Fehler gemeldet werden soll, statt Nachrichten zurück zu versenden (zu „bouncen“). Die Vorgabe ist ‘`#f`’.

`Dateiname sendmail-path` [`dovecot-configuration`-Parameter]
Welche Binärdatei zum Versenden von Mails benutzt werden soll. Die Vorgabe ist ‘`"/usr/sbin/sendmail"`’.

`Zeichenkette submission-host` [`dovecot-configuration`-Parameter]
Wenn dieses Feld nicht leer ist, werden Mails an den SMTP-Server auf dem angegebenen „Rechner[:Port]“ statt an sendmail geschickt. Die Vothabe ist ‘`""`’.

`Zeichenkette rejection-subject` [`dovecot-configuration`-Parameter]
Die Betreffkopfzeile („Subject:“), die für Mails benutzt werden soll, die über die Zurückweisung einer Mail informieren. Sie können dieselben Variablen wie beim hierunter beschriebenen Zurückweisungsgrund ‘`rejection-reason`’ benutzen. Die Vorgabe ist ‘`"Rejected: %s"`’.

`Zeichenkette rejection-reason` [`dovecot-configuration`-Parameter]
Die menschenlesbare Fehlermeldung in Mails, die über die Zurückweisung einer Mail informieren. Sie können diese Variablen benutzen:

`%n`	CRLF-Zeilenumbruch
`%r`	Begründung („Reason“)
`%s`	Ursprünglicher Betreff („Subject“)
`%t`	Empfänger („To“)

Die Vorgabe ist ‘`"Your message to <%t> was automatically rejected:%n%r"`’.

`Zeichenkette recipient-delimiter` [`dovecot-configuration`-Parameter]
Trennzeichen zwischen dem eigentlichen Lokalteil („local-part“) und Detailangaben in der E-Mail-Adresse. Die Vorgabe ist ‘`"+"`’.

`Zeichenkette` [`dovecot-configuration`-Parameter]
`lda-original-recipient-header`
Aus welcher Kopfzeile die Adresse des Ursprünglichen Empfängers (SMTPs „RCPT TO:“-Adresse) genommen wird, wenn sie nicht anderweitig eingetragen ist. Wird die Befehlszeilenoption `-a` von dovecot-lda angegeben, hat sie Vorrang vor diesem Feld. Oft wird die Kopfzeile X-Original-To hierfür verwendet. Die Vorgabe ist ‘`""`’.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`lda-mailbox-autocreate?`
Ob ein nicht existierendes Postfach (eine „Mailbox“) automatisch erzeugt werden soll, wenn eine Mail darin abgespeichert wird. Die Vorgabe ist ‘`#f`’.

`Boolescher-Ausdruck` [`dovecot-configuration`-Parameter]
`lda-mailbox-autosubscribe?`
Ob automatisch erzeugte Postfächer („Mailboxes“) auch automatisch abonniert werden sollen. Die Vorgabe ist ‘`#f`’.

`Nichtnegative-ganze-Zahl` [`dovecot-configuration`-Parameter]
`imap-max-line-length`
Die maximale Länge einer IMAP-Befehlszeile. Manche Clients erzeugen sehr lange Befehlszeilen bei riesigen Postfächern, daher müssen Sie diesen Wert gegebenenfalls anheben, wenn Sie Fehlermeldungen wie „Too long argument“ oder „IMAP command line too large“ häufig sehen. Die Vorgabe ist ‘`64000`’.

`Zeichenkette imap-logout-format` [`dovecot-configuration`-Parameter]
Formatzeichenkette für das Abmelden bei IMAP:

`%i` Gesamtzahl vom Client empfangener Bytes

`%o` Gesamtzahl zum Client versandter Bytes

Siehe `doc/wiki/Variables.txt` für eine Liste aller Variablen, die Sie benutzen können. Die Vorgabe ist ‘`"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}"`’.

`Zeichenkette imap-capability` [`dovecot-configuration`-Parameter]
Ersetzt die Antworten auf IMAP-CAPABILITY-Anfragen. Wenn der Wert mit „`+`“ beginnt, werden die angegebenen Capabilitys zu den voreingestellten hinzugefügt (z.B. +XFOO XBAR). Die Vorgabe ist ‘`""`’.

`Zeichenkette` [`dovecot-configuration`-Parameter]
`imap-idle-notify-interval`
Wie lange zwischen „OK Still here“-Benachrichtigungen gewartet wird, wenn der Client auf IDLE steht. Die Vorgabe ist ‘`"2 mins"`’.

`Zeichenkette imap-id-send` [`dovecot-configuration`-Parameter]
ID-Feldnamen und -werte, die an Clients versandt werden sollen. Wenn * als der Wert angegeben wird, benutzt Dovecot dafür den voreingestellten Wert. Die folgenden Felder verfügen derzeit über voreingestellte Werte: name, version, os, os-version, support-url, support-email. Die Vorgabe ist ‘`""`’.

`Zeichenkette imap-id-log` [`dovecot-configuration`-Parameter]
Welche vom Client übermittelten ID-Felder protokolliert werden. * bedeutet alle. Die Vorgabe ist '`""`'.

`Leerzeichengetrennte-Zeichenketten-Liste imap-client-workarounds` [`dovecot-configuration`-Parameter]
Maßnahmen, um verschiedene Fehler in Clients zu umgehen:

`delay-newmail`
Benachrichtigungen über neue Mails mit EXISTS/RECENT nur als Antwort auf NOOP- und CHECK-Befehle versenden. Manche Clients ignorieren diese ansonsten, zum Beispiel OSX Mail (<v2.1). Outlook Express verhält sich noch problematischer, denn ohne diese Maßnahme können dem Anwender Fehlermeldungen wie „Die Nachricht steht nicht mehr auf dem Server zur Verfügung“ („Message no longer in server“) angezeigt werden. Beachten Sie, dass OE6 auch mit dieser Maßnahme immer noch Probleme macht, wenn die Synchronisation auf „nur Kopfzeilen“ („Headers Only“) eingestellt ist.

`tb-extra-mailbox-sep`
Thunderbird kommt aus irgendeinem Grund durcheinander bei LAYOUT=fs (mbox und dbox) und fügt überzählige '`/`'-Suffixe an Postfachnamen („Mailbox“-Namen) an. Mit dieser Maßnahme ignoriert Dovecot zusätzliche '`/`', statt sie als ungültige Postfachnamen zu behandeln.

`tb-lsub-flags`
Ob \Noselect-Flags für LSUB-Antworten mit LAYOUT=fs (z.B. mbox) geliefert werden. Dadurch merkt Thunderbird, dass man Postfächer nicht auswählen kann, und zeigt sie ausgegraut an, statt erst nach einiger Zeit eine Fehlermeldung einzublenden, sie seien nicht auswählbar.

Die Vorgabe ist '`()`'.

`Zeichenkette imap-urlauth-host` [`dovecot-configuration`-Parameter]
Welcher Rechner in vom Client übermittelten URLAUTH-URLs zugelassen wird. Bei „*“ wird jeder zugelassen. Die Vorgabe ist '`""`'.

Uff! Das waren viele Konfigurationsoptionen. Das Schöne daran ist aber, dass Guix eine vollständige Schnittstelle für alles bietet, was man in Dovecots Konfigurationssprache ausdrücken kann. Damit können Sie Konfigurationen nicht nur auf schöne Art aufschreiben, sondern kann auch reflektiven Code schreiben, der Konfigurationen aus Scheme heraus auslesen und umschreiben kann.

Vielleicht haben Sie aber auch einfach schon eine `dovecot.conf`, die Sie mit Guix zum Laufen bringen möchten. In diesem Fall können Sie eine `opaque-dovecot-configuration` im `#:config`-Parameter an `dovecot-service` übergeben. Wie der Name schon sagt, bietet eine opake Konfiguration keinerlei Unterstützung für Reflexion.

Verfügbare `opaque-dovecot-configuration`-Felder sind:

`„package“ dovecot` [`opaque-dovecot-configuration`-Parameter]
Das Dovecot-Paket.

`Zeichenkette string` [`opaque-dovecot-configuration`-Parameter]
Der Inhalt der `dovecot.conf` als eine Zeichenkette.

Wenn Ihre `dovecot.conf` zum Beispiel nur aus der leeren Zeichenkette bestünde, könnten Sie einen Dovecot-Dienst wie folgt instanziieren:

```
(dovecot-service #:config
                 (opaque-dovecot-configuration
                  (string "")))
```

OpenSMTPD-Dienst

`opensmtpd-service-type` [Scheme-Variable]
Dies ist der Diensttyp des OpenSMTPD-Dienstes (`https://www.opensmtpd.org`), dessen Wert ein `opensmtpd-configuration`-Objekt sein sollte, wie in diesem Beispiel:

```
(service opensmtpd-service-type
         (opensmtpd-configuration
           (config-file (local-file "./my-smtpd.conf"))))
```

`opensmtpd-configuration` [Datentyp]
Datentyp, der die Konfiguration von opensmtpd repräsentiert.

`package` (Vorgabe: *opensmtpd*)
Das Paketobjekt des SMTP-Servers OpenSMTPD.

`config-file` (Vorgabe: `%default-opensmtpd-config-file`)
Ein dateiartiges Objekt der OpenSMTPD-Konfigurationsdatei, die benutzt werden soll. Nach Vorgabe lauscht OpenSMTPD auf der Loopback-Netzwerkschnittstelle und ist so eingerichtet, dass Mail von Nutzern und Daemons auf der lokalen Maschine sowie E-Mails an entfernte Server versandt werden können. Führen Sie `man smtpd.conf` aus, wenn Sie mehr erfahren möchten.

`setgid-commands?` (Vorgabe: `#t`)
Dadurch werden die folgenden Befehle setgid als `smtpq`, damit sie ausgeführt werden können: `smtpctl`, `sendmail`, `send-mail`, `makemap`, `mailq` und `newaliases`. Siehe Abschnitt 12.10 [Setuid-Programme], Seite 604, für mehr Informationen zu setgid-Programmen.

Exim-Dienst

`exim-service-type` [Scheme-Variable]
Dies ist der Diensttyp für den Mail Transfer Agent (MTA) namens Exim (`https://exim.org`), dessen Wert ein `exim-configuration`-Objekt sein sollte, wie in diesem Beispiel:

```
(service exim-service-type
         (exim-configuration
           (config-file (local-file "./my-exim.conf"))))
```

Um einen Dienst vom Typ `exim-service-type` zu benutzen, müssen Sie auch einen Dienst `mail-aliases-service-type` in Ihrer `operating-system`-Deklaration stehen haben (selbst wenn darin keine Alias-Namen eingerichtet sind).

`exim-configuration` [Datentyp]
Der Datentyp, der die Konfiguration von Exim repräsentiert.

`package` (Vorgabe: *exim*)
Das Paketobjekt des Exim-Servers.

`config-file` (Vorgabe: `#f`)
Ein dateiartiges Objekt der Exim-Konfigurationsdatei. Wenn sein Wert `#f` ist, wird die vorgegebene Konfigurationsdatei aus dem als `package` angegebenen Paket benutzt. Die sich ergebende Konfigurationsdatei wird geladen, nachdem die Konfigurationsvariablen `exim_user` und `exim_group` gesetzt wurden.

Getmail-Dienst

`getmail-service-type` [Scheme-Variable]
Dies ist der Diensttyp des Mail-Retrievers Getmail (`http://pyropus.ca/software/getmail/`), der als Wert ein `getmail-configuration`-Objekt hat.

Verfügbare `getmail-configuration`-Felder sind:

`Zeichenkette name` [`getmail-configuration`-Parameter]
Ein Symbol, das den getmail-Dienst identifiziert.

Die Vorgabe ist ‘`"unset"`’.

`„package“ package` [`getmail-configuration`-Parameter]
Das getmail-Paket, das benutzt werden soll.

`Zeichenkette user` [`getmail-configuration`-Parameter]
Das Benutzerkonto, mit dem getmail ausgeführt wird.

Die Vorgabe ist ‘`"getmail"`’.

`Zeichenkette group` [`getmail-configuration`-Parameter]
Die Benutzergruppe, mit der getmail ausgeführt wird.

Die Vorgabe ist ‘`"getmail"`’.

`Zeichenkette directory` [`getmail-configuration`-Parameter]
Welches getmail-Verzeichnis benutzt werden soll.

Die Vorgabe ist ‘`"/var/lib/getmail/default"`’.

`„getmail-configuration-file“ rcfile` [`getmail-configuration`-Parameter]
Die zu benutzende getmail-Konfigurationsdatei.

Verfügbare `getmail-configuration-file`-Felder sind:

,,getmail-retriever-configuration [getmail-configuration-file-Parameter]
retriever

Von welchem E-Mail-Konto Mails bezogen werden sollen und wie auf dieses zugegriffen werden kann.

Verfügbare getmail-retriever-configuration-Felder sind:

Zeichenkette type [getmail-retriever-configuration-Parameter]
Welche Art von Mail-Retriever benutzt werden soll. Zu den gültigen Werten gehören ‘passwd’ und ‘static’.

Die Vorgabe ist ‘"SimpleIMAPSSLRetriever"’.

Zeichenkette server [getmail-retriever-configuration-Parameter]
Der Benutzername, mit dem man sich beim Mailserver anmeldet.

Die Vorgabe ist ‘unset’.

Zeichenkette [getmail-retriever-configuration-Parameter]
username
Der Benutzername, mit dem man sich beim Mailserver anmeldet.

Die Vorgabe ist ‘unset’.

Nichtnegative-ganze-Zahl [getmail-retriever-configuration-Parameter]
port
Die Portnummer, mit der eine Verbindung hergestellt wird.

Vorgegeben ist ‘#f’.

Zeichenkette [getmail-retriever-configuration-Parameter]
password
Einträge, die Vorrang vor den Feldern aus passwd haben.

Die Vorgabe ist ‘""’.

Liste [getmail-retriever-configuration-Parameter]
password-command
Einträge, die Vorrang vor den Feldern aus passwd haben.

Die Vorgabe ist ‘()’.

Zeichenkette [getmail-retriever-configuration-Parameter]
keyfile
Der Schlüssel im PEM-Format, der für das Aufbauen der TLS-Verbindung genutzt werden soll.

Die Vorgabe ist ‘""’.

Zeichenkette [getmail-retriever-configuration-Parameter]
certfile
Die Zertifikatsdatei im PEM-Format, die für das Aufbauen der TLS-Verbindung genutzt werden soll.

Die Vorgabe ist ‘""’.

`Zeichenkette` [`getmail-retriever-configuration`-Parameter]
`ca-certs`
Welche Zertifikate von Zertifikatsautoritäten („CA Certificates“) benutzt werden sollen.

Die Vorgabe ist ‘`""`’.

`Parameter-Assoziative-Liste` [`getmail-retriever-configuration`-Parameter]
`extra-parameters`
Weitere Parameter für den Retriever.

Die Vorgabe ist ‘`()`’.

`,,getmail-destination-configuration` [`getmail-configuration-file`-Parameter]
`destination`
Was mit geholten Nachrichten geschehen soll.

Verfügbare `getmail-destination-configuration`-Felder sind:

`Zeichenkette type` [`getmail-destination-configuration`-Parameter]
Die Art des Empfängers der Mail. Zu den gültigen Werten gehören ‘`Maildir`’, ‘`Mboxrd`’ und ‘`MDA_external`’.

Die Vorgabe ist ‘`unset`’.

`Zeichenkette-oder-Datei-like-Objekt` [`getmail-destination-configuration`-Parameter]
`path`
Entspricht der path-Option für den Mailempfänger („Destination“). Was hiermit bewirkt wird, hängt von der gewählten Empfängerart ab.

Die Vorgabe ist ‘`""`’.

`Parameter-Assoziative-Liste` [`getmail-destination-configuration`-Parameter]
`extra-parameters`
Weitere Empfängerparameter.

Die Vorgabe ist ‘`()`’.

`,,getmail-options-configuration` [`getmail-configuration-file`-Parameter]
`options`
getmail konfigurieren.

Verfügbare `getmail-options-configuration`-Felder sind:

`Nichtnegative-ganze-Zahl` [`getmail-options-configuration`-Parameter]
`verbose`
Wenn es auf ‘`0`’ gesetzt ist, wird getmail nur Warnungen und Fehler ausgeben. Ein Wert von ‘`1`’ bedeutet, dass Meldungen über das Holen und Löschen von Nachrichten ausgegeben werden. Wenn es auf ‘`2`’ gesetzt ist, wird getmail Meldungen über jede durchgeführte Aktion ausgeben.

Die Vorgabe ist ‘`1`’.

`Boolescher-Ausdruck` [`getmail-options-configuration`-Parameter]
`read-all`

Wenn es auf wahr gesetzt ist, wird getmail alle verfügbaren Nachrichten holen. Andernfalls wird es nur solche Nachrichten holen, die es nicht bereits gesehen hat.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck` [`getmail-options-configuration`-Parameter]
`delete`

Wenn es auf wahr gesetzt ist, werden Mitteilungen vom Server gelöscht, nachdem sie erfolgreich geholt und zugestellt wurden. Andernfalls werden Nachrichten auf dem Server gelassen.

Vorgegeben ist '`#f`'.

`Nichtnegative-ganze-Zahl` [`getmail-options-configuration`-Parameter]
`delete-after`

Getmail wird nach der hier angegebenen Anzahl von Tagen Nachrichten löschen, die es gesehen hat, wenn sie zugestellt wurden. Dadurch werden Nachrichten diese Anzahl von Tagen lang auf dem Server gelassen, nachdem sie zugestellt wurden. Ein Wert von '`0`' deaktiviert diese Funktionalität.

Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl` [`getmail-options-configuration`-Parameter]
`delete-bigger-than`

Nachrichten, die größer als die angegebene Anzahl von Bytes sind, nach dem Holen löschen, selbst wenn die Optionen delete und delete-after abgeschaltet sind. Ein Wert von '`0`' deaktiviert diese Funktionalität.

Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl` [`getmail-options-configuration`-Parameter]
`max-bytes-per-session`

Nachrichten, die höchstens die angegebene Anzahl von Bytes groß sind, vor dem Beenden der Serversitzung von dort holen. Ein Wert von '`0`' deaktiviert diese Funktionalität.

Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl` [`getmail-options-configuration`-Parameter]
`max-message-size`

Keine Nachrichten holen, deren Größe die angegebene Anzahl von Bytes überschreitet. Ein Wert von '`0`' deaktiviert diese Funktionalität.

Die Vorgabe ist '`0`'.

`Boolescher-Ausdruck` [`getmail-options-configuration`-Parameter]
`delivered-to`

Wenn dies auf wahr gesetzt ist, fügt getmail eine Delivered-To-Kopfzeile an die Nachrichten an.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck` [`getmail-options-configuration`-Parameter]
`received`
: Wenn dies gesetzt ist, fügt getmail eine Received-Kopfzeile an die Nachrichten an.

  Die Vorgabe ist '`#t`'.

`Zeichenkette` [`getmail-options-configuration`-Parameter]
`message-log`
: Getmail wird seine Aktionen in die genannte Datei protokollieren. Wenn als Wert '`""`' angegeben wird, wird diese Funktionalität deaktiviert.

  Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck` [`getmail-options-configuration`-Parameter]
`message-log-syslog`
: Wenn es auf wahr gesetzt ist, wird getmail ein Protokoll seiner Aktionen an den Systemprotokolldienst übergeben.

  Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck` [`getmail-options-configuration`-Parameter]
`message-log-verbose`
: Wenn dies auf wahr gesetzt ist, wird getmail Informationen über *nicht* geholte Nachrichten und den jeweiligen Grund dafür sowie Anfang und Ende des Holvorgangs in Informationszeilen protokollieren.

  Vorgegeben ist '`#f`'.

`Parameter-Assoziative-Liste` [`getmail-options-configuration`-Parameter]
`extra-parameters`
: Weitere geltende Optionen.

  Die Vorgabe ist '`()`'.

`Liste idle` [`getmail-configuration`-Parameter]
: Eine Liste der Postfächer, für die getmail beim Server auf Benachrichtigungen wegen neuer Mails warten soll. Diese Funktionalität setzt voraus, dass der Server die IDLE-Erweiterung unterstützt.

  Die Vorgabe ist '`()`'.

`Liste environment-variables` [`getmail-configuration`-Parameter]
: Umgebungsvariable, die für getmail gelten sollen.

  Die Vorgabe ist '`()`'.

Dienst für Mail-Alias-Namen

`mail-aliases-service-type` [Scheme-Variable]
: Das ist der Typ des Dienstes, der `/etc/aliases` zur Verfügung stellt, wo festgelegt wird, wie Mail-Nachrichten an Benutzer des Systems geliefert werden.

```
(service mail-aliases-service-type
         '(("postmaster" "bob")
           ("bob" "bob@example.com" "bob@example2.com")))
```

Die Konfiguration für einen Dienst vom Typ `mail-aliases-service-type` ist eine assoziative Liste, die angibt, wie beim System ankommende Mail-Nachrichten zuzustellen sind. Jeder Eintrag hat die Form `(Alias Adressen ...)`, wobei das `Alias` den lokalen Alias-Namen angibt und `Adressen` angibt, wo die Mail-Nachrichten für diesen Benutzer ankommen sollen.

Die Alias-Namen müssen nicht als Benutzerkonten auf dem lokalen System existieren. Im Beispiel oben muss es also keinen Eintrag für `postmaster` unter den `user-accounts` in der `operating-system`-Deklaration geben, um die `postmaster`-Mails an `bob` weiterzuleiten (von wo diese dann an `bob@example.com` und `bob@example2.com` weitergeschickt würden).

GNU-Mailutils-IMAP4-Daemon

`imap4d-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den IMAP4-Daemon aus den GNU Mailutils (siehe Abschnitt "imap4d" in *GNU Mailutils Manual*), dessen Wert ein `imap4d-configuration`-Objekt sein sollte, wie in diesem Beispiel:

```
(service imap4d-service-type
         (imap4d-configuration
           (config-file (local-file "imap4d.conf"))))
```

`imap4d-configuration` [Datentyp]

Datentyp, der die Konfiguration von `imap4d` repräsentiert.

`package` (Vorgabe: `mailutils`)
: Das Paket, das `imap4d` zur Verfügung stellt.

`config-file` (Vorgabe: `%default-imap4d-config-file`)
: Dateiartiges Objekt der zu nutzenden Konfigurationsdatei. Nach Vorgabe lauscht IMAP4D auf TCP-Port 143 vom lokalen Rechner `localhost`. Siehe Abschnitt "Conf-imap4d" in *GNU Mailutils Manual* für Details.

Radicale-Dienst

`radicale-service-type` [Scheme-Variable]

Dies ist der Diensttyp des CalDAV- und CardDAV-Servers Radicale (`https://radicale.org`), der als Wert ein `radicale-configuration`-Objekt hat.

`radicale-configuration` [Datentyp]

Datentyp, der die Konfiguration von `radicale` repräsentiert.

`package` (Vorgabe: `radicale`)
: Das Paket, das `radicale` zur Verfügung stellt.

`config-file` (Vorgabe: `%default-radicale-config-file`)
: Dateiartiges Objekt der zu nutzenden Konfigurationsdatei. Nach Vorgabe lauscht Radicale auf TCP-Port 5232 vom lokalen Rechner `localhost` und benutzt die `htpasswd`-Datei unter `/var/lib/radicale/users` mit Passwörtern im Klartext („plain“).

12.9.13 Kurznachrichtendienste

Das Modul `(gnu services messaging)` stellt Guix-Dienstdefinitionen für Kurznachrichtendienste, d.h. „Instant Messaging“, zur Verfügung. Zurzeit werden folgende Dienste unterstützt:

Prosody-Dienst

`prosody-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den XMPP-Kommunikationsserver Prosody (`https://prosody.im`). Sein Wert muss ein `prosody-configuration`-Verbundsobjekt wie in diesem Beispiel sein:

```
(service prosody-service-type
         (prosody-configuration
          (modules-enabled (cons* "groups" "mam" %default-modules-enabled))
          (int-components
           (list
            (int-component-configuration
             (hostname "conference.example.net")
             (plugin "muc")
             (mod-muc (mod-muc-configuration)))))
          (virtualhosts
           (list
            (virtualhost-configuration
             (domain "example.net"))))))
```

Siehe im Folgenden Details über die `prosody-configuration`.

Prosody kann mit den Vorgabeeinstellungen ohne viel weitere Konfiguration benutzt werden. Nur ein `virtualhosts`-Feld wird gebraucht: Es legt die Domain fest, um die sich Prosody kümmert.

Sie können die Korrektheit der generierten Konfigurationsdatei überprüfen, indem Sie den Befehl `prosodyctl check` ausführen.

Prosodyctl hilft auch dabei, Zertifikate aus dem `letsencrypt`-Verzeichnis zu importieren, so dass das `prosody`-Benutzerkonto auf sie Zugriff hat. Siehe `https://prosody.im/doc/letsencrypt`.

```
prosodyctl --root cert import /etc/letsencrypt/live
```

Im Folgenden finden Sie die verfügbaren Konfigurationsparameter. Jeder Parameterdefinition ist ihr Typ vorangestellt; zum Beispiel bedeutet ‘`Zeichenketten-Liste foo`’, dass der Parameter `foo` als Liste von Zeichenketten angegeben werden sollte. Typangaben, die mit `Vielleicht-` beginnen, stehen für Parameter, die in `prosody.cfg.lua` *nicht* vorkommen, falls deren Wert *nicht* angegeben wurde.

Sie können die Konfiguration auch als eine Zeichenkette festlegen, wenn Sie über eine alte `prosody.cfg.lua`-Datei verfügen, die Sie von einem anderen System übernehmen möchten; siehe das Ende dieses Abschnitts für Details.

Der Typ `Dateiobjekt` bezeichnet hierbei entweder ein dateiartiges Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) oder einen Dateinamen.

Verfügbare `prosody-configuration`-Felder sind:

„package“ prosody [prosody-configuration-Parameter]
Das Prosody-Paket.

Dateiname data-path [prosody-configuration-Parameter]
Der Ort, wo sich Prosodys Verzeichnis zum Speichern von Daten befinden soll. Siehe https://prosody.im/doc/configure. Die Vorgabe ist '"/var/lib/prosody"'.

Dateiobjekt-Liste plugin-paths [prosody-configuration-Parameter]
Zusätzliche Plugin-Verzeichnisse. Plugins werden der Reihe nach in allen festgelegten Pfaden gesucht. Siehe https://prosody.im/doc/plugins_directory. Die Vorgabe ist '()'.

Dateiname certificates [prosody-configuration-Parameter]
Jeder virtuellte Rechner und jede Komponente braucht ein Zertifikat, mit dem Clients und Server ihre Identität sicher verifizieren können. Prosody lädt automatisch Zertifikate bzw. Schlüssel aus dem hier angegebenen Verzeichnis. Die Vorgabe ist '"/etc/prosody/certs"'.

Zeichenketten-Liste admins [prosody-configuration-Parameter]
Dies ist eine Liste der Benutzerkonten, die auf diesem Server Administratoren sind. Beachten Sie, dass Sie die Benutzerkonten noch separat als Nutzer erzeugen lassen müssen. Siehe https://prosody.im/doc/admins and https://prosody.im/doc/creating_accounts. Ein Beispiel: (admins '("user1@example.com" "user2@example.net")) Die Vorgabe ist '()'.

Boolescher-Ausdruck use-libevent? [prosody-configuration-Parameter]
Die Nutzung von libevent aktivieren, damit bessere Leistungsfähigkeit auch unter hoher Last gewährleistet wird. Siehe https://prosody.im/doc/libevent. Die Vorgabe ist '#f'.

Modul-Liste modules-enabled [prosody-configuration-Parameter]
Die Liste der Module, die Prosody beim Starten lädt. Es sucht nach mod_modulename.lua im Plugin-Verzeichnis, also sollten Sie sicherstellen, dass es dort auch existiert. Dokumentation über Module können Sie hier finden: https://prosody.im/doc/modules. Die Vorgabe ist '("roster" "saslauth" "tls" "dialback" "disco" "carbons" "private" "blocklist" "vcard" "version" "uptime" "time" "ping" "pep" "register" "admin_adhoc")'.

Zeichenketten-Liste modules-disabled [prosody-configuration-Parameter]
'"offline"', '"c2s"' und '"s2s"' werden automatisch geladen, aber wenn Sie sie deaktivieren möchten, tragen Sie sie einfach in die Liste ein. Die Vorgabe ist '()'.

Dateiobjekt groups-file [prosody-configuration-Parameter]
Der Pfad zu einer Textdatei, in der gemeinsame Gruppen definiert werden. Wenn dieser Pfad leer ist, dann tut 'mod_groups' nichts. Siehe https://prosody.im/doc/modules/mod_groups. Die Vorgabe ist '"/var/lib/prosody/sharedgroups.txt"'.

`Boolescher-Ausdruck` [`prosody-configuration`-Parameter]
`allow-registration?`
Ob nach Voreinstellung *keine* neuen Benutzerkonten angelegt werden können, aus Sicherheitsgründen. Siehe `https://prosody.im/doc/creating_accounts`. Die Vorgabe ist '`#f`'.

`Vielleicht-„ssl-configuration“ ssl` [`prosody-configuration`-Parameter]
Dies ist der Teil der Einstellungen, der mit SSL/TLS zu tun hat. Der Großteil davon ist deaktiviert, damit die Voreinstellungen von Prosody verwendet werden. Wenn Sie diese Optionen hier nicht völlig verstehen, sollten Sie sie *nicht* in Ihrer Konfiguration verwenden. Es passiert leicht, dass Sie die Sicherheit Ihres Servers absenken, indem Sie sie falsch benutzen. Siehe `https://prosody.im/doc/advanced_ssl_config`.

Verfügbare `ssl-configuration`-Felder sind:

`Vielleicht-Zeichenkette protocol` [`ssl-configuration`-Parameter]
Dadurch wird entschieden, was für ein Handshake benutzt wird.

`Vielleicht-Dateiname key` [`ssl-configuration`-Parameter]
Der Pfad zur Datei mit Ihrem privaten Schlüssel.

`Vielleicht-Dateiname certificate` [`ssl-configuration`-Parameter]
Der Pfad zur Datei mit Ihrem Zertifikat.

`Dateiobjekt capath` [`ssl-configuration`-Parameter]
Der Pfad zum Verzeichnis, das die Wurzelzertifikate enthält, die Prosody zur Verifikation der Zertifikate entfernter Server benutzen soll. Die Vorgabe ist '`"/etc/ssl/certs"`'.

`Vielleicht-Dateiobjekt cafile` [`ssl-configuration`-Parameter]
Der Pfad zu einer Datei, in der Wurzelzertifikate enthalten sind, denen Prosody vertrauen soll. Er verhält sich ähnlich wie `capath`, aber alle Zertifikate stehen hintereinander in der Datei.

`Vielleicht-Zeichenketten-Liste` [`ssl-configuration`-Parameter]
`verify`
Eine Liste von Verifikationsoptionen. (Die meisten bilden auf die `set_verify()`-Flags von OpenSSL ab.)

`Vielleicht-Zeichenketten-Liste` [`ssl-configuration`-Parameter]
`options`
Eine Liste allgemeiner Optionen, die mit SSL/TLS zu tun haben. Diese bilden auf OpenSSLs `set_options()` ab. Eine vollständige Liste der in LuaSec verfügbaren Optionen finden Sie im Quellcode von LuaSec.

`Vielleicht-Nichtnegative-ganze-Zahl` [`ssl-configuration`-Parameter]
`depth`
Wie lange eine Kette von Zertifikatsautoritäten („Certificate Authorities“) nach einem passenden Wurzelzertifikat durchsucht wird, dem vertraut wird.

`Vielleicht-Zeichenkette ciphers` [`ssl-configuration`-Parameter]
Eine Zeichenkette mit OpenSSL-Ciphers. Damit wird ausgewählt, welche Ciphers Prosody seinen Clients anbietet, und in welcher Reihenfolge.

`Vielleicht-Dateiname dhparam` [`ssl-configuration`-Parameter]
Ein Pfad zu einer Datei, in der Parameter für Diffie-Hellman-Schlüsselaustausche stehen. Sie können so eine Datei mit diesem Befehl erzeugen: `openssl dhparam -out /etc/prosody/certs/dh-2048.pem 2048`

`Vielleicht-Zeichenkette curve` [`ssl-configuration`-Parameter]
Die Kurve, die für Diffie-Hellman mit elliptischen Kurven verwendet werden soll. Prosodys Voreinstellung ist '`"secp384r1"`'.

`Vielleicht-Zeichenketten-Liste` [`ssl-configuration`-Parameter]
`verifyext`
Eine Liste von zusätzlichen Verifikationsoptionen.

`Vielleicht-Zeichenkette password` [`ssl-configuration`-Parameter]
Das Passwort fÜr verschlüsselte private Schlüssel.

`Boolescher-Ausdruck` [`prosody-configuration`-Parameter]
`c2s-require-encryption?`
Ob alle Verbindungen von Client zu Server zwangsweise verschlüsselt sein müssen. Siehe `https://prosody.im/doc/modules/mod_tls`. Die Vorgabe ist '`#f`'.

`Zeichenketten-Liste` [`prosody-configuration`-Parameter]
`disable-sasl-mechanisms`
Welche Mechanismen angeboten werden. Siehe `https://prosody.im/doc/modules/mod_saslauth`. Die Vorgabe ist '`("DIGEST-MD5")`'.

`Boolescher-Ausdruck` [`prosody-configuration`-Parameter]
`s2s-require-encryption?`
Ob alle Verbindungen von Server zu Server zwangsweise verschlüsselt sein müssen. Siehe `https://prosody.im/doc/modules/mod_tls`. Die Vorgabe ist '`#f`'.

`Boolescher-Ausdruck s2s-secure-auth?` [`prosody-configuration`-Parameter]
Ob Verschlüsselung und Zertifikatsauthentifizierung verpflichtend durchgeführt werden müssen. Das bietet das ideale Maß an Sicherheit, jedoch müssen dann auch die Server, mit denen Sie kommunizieren, Verschlüsselung unterstützen *und* gültige Zertifikate vorweisen, denen Sie auch vertrauen. Siehe `https://prosody.im/doc/s2s#security`. Die Vorgabe ist '`#f`'.

`Zeichenketten-Liste` [`prosody-configuration`-Parameter]
`s2s-insecure-domains`
Viele Server bieten keine Unterstützung für Verschlüsselung oder ihre Zertifikate sind ungültig oder selbstsigniert. Hier können Sie Domains eintragen, die von der Pflicht zur Authentisierung mit Zertifikaten ausgenommen werden. Diese werden dann über DNS authentifiziert. Siehe `https://prosody.im/doc/s2s#security`. Die Vorgabe ist '`()`'.

`Zeichenketten-Liste` [`prosody-configuration`-Parameter]
`s2s-secure-domains`
Selbst wenn Sie `s2s-secure-auth?` deaktiviert lassen, können Sie noch immer gültige Zertifikate bei manchen Domains verlangen, indem Sie diese hier auflisten. Siehe `https://prosody.im/doc/s2s#security`. Die Vorgabe ist '`()`'.

`Zeichenkette authentication` [`prosody-configuration`-Parameter]
Wählen Sie aus, welcher Hintergrunddienst („Provider“) zur Authentifizierung benutzt werden soll. Das vorgegebene System speichert Passwörter im Klartext ab und benutzt dafür den in Prosody eingestellten Datenspeicher, um Authentifizierungsdaten zu speichern. Wenn Sie Ihrem Server kein Vertrauen entgegenbringen, siehe `https://prosody.im/doc/modules/mod_auth_internal_hashed` für Informationen, wie Sie den gehashten Hintergrunddienst benutzen. Siehe auch `https://prosody.im/doc/authentication`. Die Vorgabe ist '`"internal_plain"`'.

`Vielleicht-Zeichenkette log` [`prosody-configuration`-Parameter]
Hiermit werden die Protokollierungsoptionen festgelegt. Fortgeschrittene Protokollierungskonfigurationen werden vom Prosody-Dienst noch nicht unterstützt. Siehe `https://prosody.im/doc/logging`. Die Vorgabe ist '`"*syslog"`'.

`Dateiname pidfile` [`prosody-configuration`-Parameter]
Die Datei, in der Prosodys Prozessidentifikator („PID“) abgelegt wird. Siehe `https://prosody.im/doc/modules/mod_posix`. Die Vorgabe ist '`"/var/run/prosody/prosody.pid"`'.

`Vielleicht-Nichtnegative-ganze-Zahl` [`prosody-configuration`-Parameter]
`http-max-content-size`
Die maximal zulässige Größe des HTTP-Rumpfs (in Bytes).

`Vielleicht-Zeichenkette` [`prosody-configuration`-Parameter]
`http-external-url`
Manche Module machen auf verschiedene Arten ihre eigene URL verfügbar. Diese URL setzt sich aus dem benutzten Protokoll, Rechnernamen und Port zusammen. Wenn Prosody hinter einem Proxy ausgeführt wird, ist die öffentliche URL stattdessen die `http-external-url`. Siehe `https://prosody.im/doc/http#external_url`.

`„virtualhost-configuration“-Liste` [`prosody-configuration`-Parameter]
`virtualhosts`
Der Name eines Rechners („Host“) in Prosody bezeichnet eine Domain, auf der Benutzerkonten angelegt werden können. Wenn Sie zum Beispiel möchten, dass Nutzer Adressen haben wie '`"john.smith@example.com"`', dann müssen Sie einen Rechnernamen '`"example.com"`' hinzufügen. Alle Optionen in dieser Liste gelten nur für diesen Rechnernamen.

> **Anmerkung:** Die Bezeichnung *virtueller* Rechnername wird in der Konfiguration verwendet, damit es nicht zu Verwechslungen mit dem tatsächlichen physischen Rechner kommt, auf dem Prosody installiert ist. Eine einzelne Prosody-Instanz kann mehrere Domains bedienen, jede definiert mit ihrem eigenen VirtualHost-Eintrag in der Konfiguration

von Prosody. Im Gegensatz dazu hätte ein Server, der nur eine Domain anbietet, nur einen einzigen VirtualHost-Eintrag.

Siehe `https://prosody.im/doc/configure#virtual_host_settings`.

Verfügbare `virtualhost-configuration`-Felder sind:

Alle folgenden Felder, wie sie auch die `prosody-configuration` hat: `admins`, `use-libevent?`, `modules-enabled`, `modules-disabled`, `groups-file`, `allow-registration?`, `ssl`, `c2s-require-encryption?`, `disable-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`, und außerdem:

`Zeichenkette domain` [`virtualhost-configuration`-Parameter]
Die Domain, auf der man Prosody erreichen soll.

`„int-component-configuration“-Liste int-components` [`prosody-configuration`-Parameter]
Komponenten sind zusätzliche Dienste auf einem Server, die Clients zur Verfügung stehen. Sie sind normalerweise auf einer Subdomain des Hauptservers verfügbar (wie zum Beispiel ‘`"mycomponent.example.com"`’). Beispiele für Komponenten könnten Server für Chaträume, Benutzerverzeichnisse oder Zugänge („Gateways“) zu anderen Protokollen sein.

Interne Komponenten werden über Prosody-spezifische Plugins implementiert. Um eine interne Komponente hinzuzufügen, tragen Sie einfach das `hostname`-Feld für den Rechnernamen und die Plugins ein, die Sie für die Komponente benutzen möchten.

Siehe `https://prosody.im/doc/components`. Die Vorgabe ist ‘`()`’.

Verfügbare `int-component-configuration`-Felder sind:

Alle folgenden Felder, wie sie auch die `prosody-configuration` hat: `admins`, `use-libevent?`, `modules-enabled`, `modules-disabled`, `groups-file`, `allow-registration?`, `ssl`, `c2s-require-encryption?`, `disable-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`, und außerdem:

`Zeichenkette hostname` [`int-component-configuration`-Parameter]
Der Rechnername für diese Komponente.

`Zeichenkette plugin` [`int-component-configuration`-Parameter]
Das Plugin, das Sie für diese Komponente benutzen möchten.

`Vielleicht-„mod-muc-configuration“ mod-muc` [`int-component-configuration`-Parameter]
Multi-User Chat (MUC) ist der Name von Prosodys Modul, womit Sie Chaträume/Konferenzen für XMPP-Benutzer anbieten lassen können.

Allgemeine Informationen über das Einrichten und Benutzen von Multi-User-Chaträumen können Sie in der Dokumentation über Chaträume finden (`https://prosody.im/doc/chatrooms`), die Sie lesen sollten, wenn Ihnen XMPP-Chaträume neu sind.

Siehe auch `https://prosody.im/doc/modules/mod_muc`.

Verfügbare `mod-muc-configuration`-Felder sind:

`Zeichenkette name` [`mod-muc-configuration`-Parameter]
Der Name, der in Antworten auf die Diensteermittlung benutzt. Die Vorgabe ist '`"Prosody Chatrooms"`'.

`Zeichenkette-oder-Boolescher-Ausdruck restrict-room-creation` [`mod-muc-configuration`-Parameter]
Für '`#t`' können nur Administratoren neue Chaträume anlegen. Andernfalls kann jeder einen Raum anlegen. Der Wert '`"local"`' schränkt das Anlegen neuer Räume auf solche Nutzer ein, die zur Eltern-Domain des Dienstes gehören. Z.B. kann '`user@example.com`' Räume auf '`rooms.example.com`' anlegen. Für den Wert '`"admin"`' können nur Dienstadministratoren Chaträume anlegen. Die Vorgabe ist '`#f`'.

`Nichtnegative-ganze-Zahl max-history-messages` [`mod-muc-configuration`-Parameter]
Die Maximalzahl der Nachrichten aus dem Chat-Verlauf, die an ein Mitglied nachversandt werden, das gerade erst dem Raum beigetreten ist. Die Vorgabe ist '`20`'.

`„ext-component-configuration“-Liste ext-components` [`prosody-configuration`-Parameter]
Externe Komponenten benutzen XEP-0114, das von den meisten eigenständigen Komponenten unterstützt wird. Um eine externe Komponente hinzuzufügen, tragen Sie einfach den Rechnernamen ins `hostname`-Feld ein. Siehe `https://prosody.im/doc/components`. Die Vorgabe ist '`()`'.

Verfügbare `ext-component-configuration`-Felder sind:

Alle folgenden Felder, wie sie auch die `prosody-configuration` hat: `admins`, `use-libevent?`, `modules-enabled`, `modules-disabled`, `groups-file`, `allow-registration?`, `ssl`, `c2s-require-encryption?`, `disable-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`, und außerdem:

`Zeichenkette component-secret` [`ext-component-configuration`-Parameter]
Das Passwort, das die Komponente für die Anmeldung benutzt.

`Zeichenkette hostname` [`ext-component-configuration`-Parameter]
Der Rechnername für diese Komponente.

`Nichtnegative-ganze-Zahl-Liste component-ports` [`prosody-configuration`-Parameter]
Der/Die Port(s), wo Prosody auf Verbindungen zu Komponenten lauscht. Die Vorgabe ist '`(5347)`'.

`Zeichenkette component-interface` [`prosody-configuration`-Parameter]
Die Schnittstelle, auf der Prosody auf Verbindungen zu Komponenten lauscht. Die Vorgabe ist '`"127.0.0.1"`'.

`Vielleicht-Roher-Inhalt raw-content` [`prosody-configuration`-Parameter]
„Roher Inhalt", der so, wie er ist, an die Konfigurationsdatei angefügt wird.

Möglicherweise möchten Sie einfach nur eine bestehende `prosody.cfg.lua` zum Laufen bringen. In diesem Fall können Sie ein `opaque-prosody-configuration`-Verbundsobjekt als der Wert des `prosody-service-type` übergeben. Wie der Name schon sagt, bietet eine opake Konfiguration keinerlei Unterstützung für Reflexion. Verfügbare `opaque-prosody-configuration`-Felder sind:

`„package" prosody` [`opaque-prosody-configuration`-Parameter]
Das Prosody-Paket.

`Zeichenkette prosody.cfg.lua` [`opaque-prosody-configuration`-Parameter]
Der Inhalt, der als `prosody.cfg.lua` benutzt werden soll.

Wenn Ihre `prosody.cfg.lua` zum Beispiel nur aus der leeren Zeichenkette bestünde, könnten Sie einen Prosody-Dienst wie folgt instanziieren:

```
(service prosody-service-type
         (opaque-prosody-configuration
          (prosody.cfg.lua "")))
```

BitlBee-Dienst

BitlBee (`https://bitlbee.org`) ist ein Zugang („Gateway"), der eine IRC-Schnittstelle für verschiedene Kurznachrichtenprotokolle wie XMPP verfügbar macht.

`bitlbee-service-type` [Scheme-Variable]
Dies ist der Diensttyp für den BitlBee-IRC-Zugangsdaemon (`https://bitlbee.org`) (englisch „IRC Gateway Daemon"). Sein Wert ist eine `bitlbee-configuration` (siehe unten).

Damit BitlBee auf Port 6667 vom lokalen Rechner („localhost") lauscht, fügen Sie diese Zeile zu Ihrem „services"-Feld hinzu:

```
(service bitlbee-service-type)
```

`bitlbee-configuration` [Datentyp]
Dies ist die Konfiguration für BitlBee. Sie hat folgende Felder:

`interface` (Vorgabe: `"127.0.0.1"`)
`port` (Vorgabe: `6667`)
Lauscht auf der Netzwerkschnittstelle, die der als *interface* angegebenen IP-Adresse entspricht, auf dem angegebenen *port*.

Wenn als *interface* `127.0.0.1` angegeben wurde, können sich nur lokale Clients verbinden; bei `0.0.0.0` können die Verbindungen von jeder Netzwerkschnittstelle aus hergestellt werden.

`bitlbee` (Vorgabe: `bitlbee`)
Das zu benutzende BitlBee-Paket.

`plugins` (Vorgabe: `'()`)
: Die Liste zu verwendender Plugin-Pakete – z.B. `bitlbee-discord`.

`extra-settings` (Vorgabe: `""`)
: Ein Konfigurationsschnipsel, das wortwörtlich in die BitlBee-Konfigurationsdatei eingefügt wird.

Quassel-Dienst

Quassel (`https://quassel-irc.org/`) ist ein verteilter IRC-Client, was bedeutet, dass sich ein oder mehr Clients mit dem zentralen Kern verbinden und die Verbindung wieder trennen können.

`quassel-service-type` [Scheme-Variable]
: Dies ist der Diensttyp für den Daemon zum IRC-Hintergrundsystem („Backend“) Quassel (`https://quassel-irc.org/`). Sein Wert ist eine `quassel-configuration` (siehe unten).

`quassel-configuration` [Datentyp]
: Die Konfiguration für Quassel mit den folgenden Feldern:

  `quassel` (Vorgabe: `quassel`)
  : Das zu verwendende Quassel-Paket.

  `interface` (Vorgabe: `"::,0.0.0.0"`)
  `port` (Vorgabe: `4242`)
  : Lauscht auf der/den Netzwerkschnittstelle(n), die den in der kommagetrennten *interface*-Liste festgelegten IPv4- oder IPv6-Schnittstellen entsprechen, auf dem angegebenen *port*.

  `loglevel` (Vorgabe: `"Info"`)
  : Die gewünschte Detailstufe der Protokollierung. Akzeptiert werden die Werte Debug (ausführlich zur Fehlersuche), Info, Warning (nur Warnungen und Fehler) und Error (nur Fehler).

12.9.14 Telefondienste

Das Modul `(gnu services telephony)` stellt Guix-Dienstdefinitionen für Telefoniedienste zur Verfügung. Zurzeit werden folgende Dienste unterstützt:

Jami

In diesem Abschnitt wird beschrieben, wie Sie einen Server für Jami einrichten, mit dem Video- oder Audiokonferenzen bereitgestellt werden können, neben anderen Nutzungsmöglichkeiten. Das folgende Beispiel zeigt, wie Jami-Kontenarchive (Sicherungskopien) automatisch eingerichtet werden:

```
(service jami-service-type
         (jami-configuration
          (accounts
           (list (jami-account
                  (archive "/etc/jami/unverschlüsseltes-konto-1.gz"))
                 (jami-account
```

```
                    (archive "/etc/jami/unverschlüsseltes-konto-2.gz"))))))
```

Wenn etwas für das `accounts`-Feld angegeben wird, werden die Jami-Kontendateien des Dienstes unter `/var/lib/jami` bei jedem Neustart des Dienstes neu erzeugt.

Jami-Konten und die zugehörigen Archive mit Sicherungskopien können mit dem Jami-Client `jami` oder auch mit dem Client `jami-gnome` angelegt werden. Die Konten sollten *nicht* mit einem Passwort geschützt werden, aber es ist ratsam, sie nur für den Administratornutzer '`root`' lesbar zu machen.

Im nächsten Beispiel sehen Sie, wie deklariert wird, dass nur manche Kontakte mit einem bestimmten Konto kommunizieren dürfen:

```
(service jami-service-type
         (jami-configuration
          (accounts
           (list (jami-account
                  (archive "/etc/jami/unverschlüsseltes-konto-1.gz")
                  (peer-discovery? #t)
                  (rendezvous-point? #t)
                  (allowed-contacts
                   '("1dbcb0f5f37324228235564b79f2b9737e9a008f"
                     "2dbcb0f5f37324228235564b79f2b9737e9a008f")))))))
```

In diesem Modus können nur die als `allowed-contacts` deklarierten Kontakte eine Kommunikation mit diesem Jami-Konto einleiten. So etwas kann zum Beispiel benutzt werden, um Konten als „Treffpunkt“ zur Erstellung von privaten Videokonferenzräumen einzurichten.

Um dem Systemadministrator die volle Kontrolle über von deren System angebotene Konferenzen zu geben, unterstützt der Jami-Dienst die folgenden Aktionen:

```
# 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)
```

Das Ziel ist, mit den obigen Aktionen die für die Moderation wertvollsten Aktionen zur Verfügung zu stellen; wir versuchen *nicht*, die gesamte Programmierschnittstelle von Jami abzudecken. Benutzer, die aus Guile heraus mit dem Jami-Daemon interagieren möchten, interessieren sich vielleicht für das Modul `(gnu build jami-service)`, womit die obigen Shepherd-Aktionen implementiert wurden.

Die Aktionen `add-moderator` und `ban-contact` nehmen den *Fingerabdruck* (einen 40-zeichigen Hash) als erstes Argument und einen Konto-Fingerabdruck oder Benutzernamen als zweites Argument an.

```
# herd add-moderator jami 1dbcb0f5f37324228235564b79f2b9737e9a008f \
  f3345f2775ddfe07a4b0d95daea111d15fbc1199
```

```
# herd list-moderators jami
Moderators for account f3345f2775ddfe07a4b0d95daea111d15fbc1199:
  - 1dbcb0f5f37324228235564b79f2b9737e9a008f
```

Im Fall von `ban-contact` ist das zweite Benutzernamen-Argument optional; wenn Sie es weglassen, wird das Konto gegenüber allen Jami-Konten gesperrt.

```
# herd ban-contact jami 1dbcb0f5f37324228235564b79f2b9737e9a008f

# herd list-banned-contacts jami
Banned contacts for account f3345f2775ddfe07a4b0d95daea111d15fbc1199:
  - 1dbcb0f5f37324228235564b79f2b9737e9a008f
```

Gesperrten Kontakten werden auch ihre Moderatorrechte entzogen.

Die Aktion `disable-account` ermöglicht es, ein Konto gänzlich vom Netzwerk abzutrennen, also unerreichbar zu machen. Dagegen bewirkt `enable-account` das Gegenteil. Sie nehmen einen einzelnen Kontobenutzernamen oder -fingerabdruck als erstes Argument:

```
# herd disable-account jami f3345f2775ddfe07a4b0d95daea111d15fbc1199

# herd list-accounts jami
The following Jami accounts are available:
  - f3345f2775ddfe07a4b0d95daea111d15fbc1199 (dummy) [disabled]
```

Die Aktion `list-account-details` zeigt die Parameter jedes Kontos ausführlich im Recutils-Format an, d.h. der Befehl `recsel` kann benutzt werden, um die relevanten Konten auszuwählen (siehe Abschnitt "Selection Expressions" in *Handbuch der GNU recutils*). Beachten Sie, dass anstelle der Punkt-Zeichen ('`.`') in den Schlüsseln der Kontoparameter Unterstriche ('`_`') angezeigt werden, um den Anforderungen des Recutils-Formats zu genügen. Folgendes Beispiel zeigt, wie man den Fingerabdruck jedes im Treffpunktmodus befindlichen Kontos angezeigt bekommt:

```
# herd list-account-details jami | \
  recsel -p Account.username -e 'Account.rendezVous ~ "true"'
Account_username: f3345f2775ddfe07a4b0d95daea111d15fbc1199
```

Die anderen Aktionen sollten selbsterklärend sein.

Nun folgt eine vollständige Übersicht über die verfügbaren Konfigurationsoptionen.

`jami-configuration` [Datentyp]

Verfügbare `jami-configuration`-Felder sind:

`libjami` (Vorgabe: `libjami`) (Typ: „package“)
: Das Jami-Daemon-Paket, was benutzt werden soll.

`dbus` (Vorgabe: `dbus-for-jami`) (Typ: „package“)
: Das D-Bus-Paket, mit dem die benötigte D-Bus-Sitzung gestartet wird.

`nss-certs` (Vorgabe: `nss-certs`) (Typ: „package“)
: Das nss-certs-Paket, was die TLS-Zertifikate bereitstellt.

`enable-logging?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Ob Protokollierung in Syslog aktiviert sein soll.

`debug?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob Meldungen der Fehlersuch-Ausführlichkeitsstufe aktiviert sein sollen.

`auto-answer?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob Anrufe erzwungen automatisch entgegengenommen werden sollen.

`accounts` (Typ: Vielleicht-„jami-account“-Liste)
: Eine Liste der Jami-Konten, die jedes Mal (neu) eingerichtet werden, wenn der Jami-Daemon-Dienst startet. Wenn Sie dieses Feld angeben, werden die Kontoverzeichnisse unter `/var/lib/jami/` bei jedem Start des Dienstes aufs Neue erzeugt, was einen konsistenten Zustand gewährleistet.

`jami-account` [Datentyp]

Verfügbare `jami-account`-Felder sind:

`archive` (Typ: Zeichenkette-oder-„computed-file“)
: Der Dateiname des Kontenarchivs mit den Sicherungskopien des Kontos. Damit werden die Konten neu eingerichtet, sobald der Dienst startet. Das Kontenarchiv muss *unverschlüsselt* sein. Es wird dringend empfohlen, diese Dateien nur für den Administratornutzer ‘`root`’ lesbar zu machen (sie also *nicht* im Store zu speichern), damit die darin enthaltenen geheimen Schlüssel des Jami-Kontos nicht bekannt werden.

`allowed-contacts` (Typ: Vielleicht-Kontofingerabdruck-Liste)
: Die Liste der erlaubten Kontakte des Kontos in Form deren 40 Zeichen langen Fingerabdrucks. Nachrichten und eingehende Anrufe von Konten, die in der Liste fehlen, werden abgewiesen. Wenn keine Liste angegeben wird, wird die gesamte Konfiguration des Kontenarchivs als Kontakte und für öffentliche eingehende Anrufe und Nachrichten zugelassen, was normalerweise bedeutet, dass jeder Kontakt mit dem Konto kommunizieren kann.

`moderators` (Typ: Vielleicht-Kontofingerabdruck-Liste)
: Die Liste der Kontakte, die Moderationsrecht (andere Nutzer sperren, stummschalten usw.) in Treffpunkt-Konferenzen („Rendezvous“) haben. Kontakte werden in Form ihres 40 Zeichen langen Fingerabdrucks angegeben. Wird nichts festgelegt, werden Moderationsrechte an die gesamte Konfiguration des Kontenarchivs übertragen, was normalerweise bedeutet, dass jeder die Moderatorrolle einnehmen kann.

`rendezvous-point?` (Typ: Vielleicht-Boolescher-Ausdruck)
: Ob das Konto im Treffpunktmodus (Rendezvous-Modus) arbeiten soll. In diesem Modus werden alle eingehenden Audio-/Videoanrufe in eine Konferenz gemischt. Wenn nichts angegeben wird, gilt der Wert im Kontenarchiv.

`peer-discovery?` (Typ: Vielleicht-Boolescher-Ausdruck)
: Ob sich der Jami-Daemon am Multicast-Mechanismus zum Finden lokaler Netzwerkteilnehmer beteiligen soll. Damit werden andere OpenDHT-

Knoten im lokalen Netzwerk erkannt, wodurch die Kommunikation zwischen Geräten in einem solchen Netzwerk aufrechterhalten werden kann, selbst wenn die Internetverbindung verloren geht. Wenn nichts angegeben wird, gilt der Wert im Kontenarchiv.

`bootstrap-hostnames` (Typ: Vielleicht-Zeichenketten-Liste)
Eine Liste von Rechnernamen oder IP-Adressen, die auf OpenDHT-Knoten verweisen, über die beim Start eine Verbindung zum OpenDHT-Netzwerk aufgebaut wird. Wenn nichts angegeben wird, gilt der Wert im Kontenarchiv.

`name-server-uri` (Typ: Vielleicht-Zeichenkette)
Die URI des zu nutzenden Namens-Servers, mit dem der Konto-Fingerabdruck eines registrierten Nutzers erfragt werden kann.

Mumble-Server

Dieser Abschnitt beschreibt, wie Sie einen Server für Mumble (`https://mumble.info`) einrichten und ausführen. (Die Server-Komponente war ehemals bekannt unter dem Namen Murmur.)

`mumble-server-configuration` [Datentyp]
Der Diensttyp für den Mumble-Server. Eine Beispielkonfiguration kann so aussehen:

```
(service mumble-server-service-type
         (mumble-server-configuration
          (welcome-text
            "Willkommen zu diesem mit Guix betriebenen Mumble-Server!")
          (cert-required? #t) ;Anmeldungen mit Textpasswort deaktivieren
          (ssl-cert "/etc/letsencrypt/live/mumble.example.com/fullchain.pem")
          (ssl-key "/etc/letsencrypt/live/mumble.example.com/privkey.pem")))
```

Nachdem Sie Ihr System rekonfiguriert haben, können Sie das Passwort des Administratornutzers `SuperUser` auf dem Mumble-Server mit Hilfe des Befehls von Hand festlegen, der Ihnen in der Aktivierungsphase des Mumble-Servers angezeigt wird.

Es wird empfohlen, ein normales Mumble-Benutzerkonto zu registrieren und mit Administrator- oder Moderatorrechten auszustatten. Sie können auch das Clientprogramm `mumble` benutzen, um sich als neuer normaler Benutzer anzumelden und zu registrieren, und sich dann abmelden. Im nächsten Schritt melden Sie sich mit dem Benutzernamen `SuperUser` mit dem vorher festgelegten `SuperUser`-Passwort an und statten Ihren registrierten Mumble-Benutzer mit Administrator- oder Moderatorrechten aus oder erzeugen ein paar Kanäle.

Verfügbare `mumble-server-configuration`-Felder sind:

`package` (Vorgabe: `mumble`)
Das Paket, das `bin/mumble-server` enthält.

`user` (Vorgabe: `"mumble-server"`)
Der Benutzer, der den Mumble-Server ausführt.

`group` (Vorgabe: `"mumble-server"`)
Die Gruppe des Benutzers, der den Mumble-Server ausführt.

`port` (Vorgabe: `64738`)
: Der Port, auf dem der Server lauschen wird.

`welcome-text` (Vorgabe: `""`)
: Der Willkommenstext, der an Clients geschickt wird, sobald sie eine Verbindung aufgebaut haben.

`server-password` (Vorgabe: `""`)
: Das Passwort, das Clients eingeben müssen, um sich verbinden zu können.

`max-users` (Vorgabe: `100`)
: Die Maximalzahl von Nutzern, die gleichzeitig mit dem Server verbunden sein können.

`max-user-bandwidth` (Vorgabe: `#f`)
: Wie viele Stimmdaten ein Benutzer pro Sekunde versenden kann.

`database-file` (Vorgabe: `"/var/lib/mumble-server/db.sqlite"`)
: Der Dateiname der SQLite-Datenbank. Das Benutzerkonto für den Dienst wird Besitzer des Verzeichnisses.

`log-file` (Vorgabe: `"/var/log/mumble-server/mumble-server.log"`)
: Der Dateiname der Protokolldatei. Das Benutzerkonto für den Dienst wird Besitzer des Verzeichnisses.

`autoban-attempts` (Vorgabe: `10`)
: Wie oft sich ein Benutzer innerhalb des in `autoban-timeframe` angegebenen Zeitrahmens verbinden kann, ohne automatisch für die in `autoban-time` angegebene Zeit vom Server verbannt zu werden.

`autoban-timeframe` (Vorgabe: `120`)
: Der Zeitrahmen für automatisches Bannen in Sekunden.

`autoban-time` (Vorgabe: `300`)
: Wie lange in Sekunden ein Client gebannt wird, wenn er die Autobann-Beschränkungen überschreitet.

`opus-threshold` (Vorgabe: `100`)
: Der Prozentanteil der Clients, die Opus unterstützen müssen, bevor der Opus-Audiocodec verwendet wird.

`channel-nesting-limit` (Vorgabe: `10`)
: Wie tief Kanäle höchstens ineinander verschachtelt sein können.

`channelname-regex` (Vorgabe: `#f`)
: Eine Zeichenkette in Form eines regulären Ausdrucks von Qt, zu dem Kanalnamen passen müssen.

`username-regex` (Vorgabe: `#f`)
: Eine Zeichenkette in Form eines regulären Ausdrucks von Qt, zu dem Nutzernamen passen müssen.

`text-message-length` (Vorgabe: `5000`)
: Wie viele Bytes ein Benutzer höchstens in einer Textchatnachricht verschicken kann.

`image-message-length` (Vorgabe: `(* 128 1024)`)
: Wie viele Bytes ein Benutzer höchstens in einer Bildnachricht verschicken kann.

`cert-required?` (Vorgabe: `#f`)
: Falls dies auf `#t` gesetzt ist, werden Clients abgelehnt, die sich bloß mit Passwörtern authentisieren. Benutzer müssen den Zertifikatsassistenten abgeschlossen haben, bevor sie sich verbinden können.

`remember-channel?` (Vorgabe: `#f`)
: Ob sich `mumble-server` für jeden Nutzer den Kanal merken soll, auf dem er sich zuletzt befunden hat, als er die Verbindung getrennt hat, so dass er wieder auf dem gemerkten Kanal landet, wenn er dem Server wieder beitritt.

`allow-html?` (Vorgabe: `#f`)
: Ob HTML in Textnachrichten, Nutzerkommentaren und Kanalbeschreibungen zugelassen wird.

`allow-ping?` (Vorgabe: `#f`)
: Wenn es auf wahr gesetzt ist, wird an nicht angemeldete Anwender die momentane Benutzerzahl, die maximale Benutzerzahl und die maximale Bandbreite pro Benutzer übermittelt. Im Mumble-Client werden diese Informationen im Verbinden-Dialog angezeigt.

  Wenn diese Einstellung deaktiviert ist, wird der Server nicht in der öffentlichen Serverliste aufgeführt.

`bonjour?` (Vorgabe: `#f`)
: Ob der Server im lokalen Netzwerk anderen über das Bonjour-Protokoll mitgeteilt werden soll.

`send-version?` (Vorgabe: `#f`)
: Ob die `mumble-server`-Serverversion Clients gegenüber in Ping-Anfragen mitgeteilt werden soll.

`log-days` (Vorgabe: `31`)
: Mumble führt in der Datenbank Protokolle, auf die über entfernte Prozeduraufrufe („Remote Procedure Calls“, kurz RPC) zugegriffen werden kann. Nach Vorgabe bleiben diese 31 Tage lang erhalten, aber sie können diese Einstellung auf 0 setzen, damit sie ewig gespeichert werden, oder auf -1, um *keine* Protokolle in die Datenbank zu schreiben.

`obfuscate-ips?` (Vorgabe: `#t`)
: Ob IP-Adressen in Protokollen anonymisiert werden sollen, um die Privatsphäre von Nutzern zu schützen.

`ssl-cert` (Vorgabe: `#f`)
: Der Dateiname des SSL-/TLS-Zertifikats, das für verschlüsselte Verbindungen benutzt werden soll.

```
(ssl-cert "/etc/letsencrypt/live/example.com/fullchain.pem")
```

`ssl-key` (Vorgabe: `#f`)
: Dateipfad zum privaten Schlüssel für SSL, was für verschlüsselte Verbindungen benutzt wird.

```
(ssl-key "/etc/letsencrypt/live/example.com/privkey.pem")
```

`ssl-dh-params` (Vorgabe: `#f`)
: Dateiname einer PEM-kodierten Datei mit Diffie-Hellman-Parametern für die SSL-/TLS-Verschlüsselung. Alternativ setzen Sie ihn auf `"@ffdhe2048"`, `"@ffdhe3072"`, `"@ffdhe4096"`, `"@ffdhe6144"` oder `"@ffdhe8192"`, wodurch die mitgelieferten Parameter aus RFC 7919 genutzt werden.

`ssl-ciphers` (Vorgabe: `#f`)
: Die Option `ssl-ciphers` wählt aus, welche Cipher-Suites zur Verwendung in SSL/TLS verfügbar sein sollen.

  Diese Option wird in der OpenSSL-Notation für Cipher-Listen (`https://www.openssl.org/docs/apps/ciphers.html#CIPHER-LIST-FORMAT`) angegeben.

  Es wird empfohlen, dass Sie Ihre Cipher-Zeichenkette mit „openssl ciphers <Zeichenkette>“ prüfen, bevor Sie sie hier einsetzen, um ein Gefühl dafür zu bekommen, was für eine Cipher-Suite sie damit bekommen. Nachdem Sie diese Option festgelegt haben, wird empfohlen, dass Sie das Protokoll Ihres Mumble-Servers durchsehen und sicherstellen, dass Mumble auch wirklich die Cipher-Suites benutzt, die Sie erwarten.

  > **Anmerkung:** Änderungen hieran können die Rückwärtskompatibilität Ihres Mumble-Servers beeinträchtigen; dadurch kann es für ältere Mumblie-Clients unmöglich werden, sich damit zu verbinden.

`public-registration` (Vorgabe: `#f`)
: Hier muss ein `<mumble-server-public-registration-configuration>`-Verbundsobjekt oder `#f` angegeben werden.

  Sie können Ihren Server optional in die öffentliche Serverliste eintragen lassen, die der Mumble-Client `mumble` beim Start anzeigt. Sie können Ihren Server nicht registrieren, wenn Sie ein `server-password` festgelegt oder `allow-ping` auf `#f` gesetzt haben.

  Es könnte ein paar Stunden dauern, bis er in der öffentlichen Liste zu finden ist.

`file` (Vorgabe: `#f`)
: Optional kann hier eine vorrangig benutzte alternative Konfiguration festgelegt werden.

`mumble-server-public-registration-configuration` [Datentyp]
Konfiguration für das öffentliche Registrieren eines `mumble-server`-Dienstes.

`name` Dies ist ein Anzeigename für Ihren Server. Er ist nicht zu verwechseln mit dem Rechnernamen („Hostname“).

`password` Ein Passwort, um Ihre Registrierung zu identifizieren. Nachfolgende Aktualisierungen müssen dasselbe Passwort benutzen. Verlieren Sie Ihr Passwort nicht.

`url` Dies sollte ein Link mit `http://` oder `https://` auf Ihren Webauftritt sein.

`hostname` (Vorgabe: `#f`)
Nach Vorgabe wird Ihr Server über seine IP-Adresse aufgeführt. Wenn dies gesetzt ist, wird er stattdessen mit diesem Rechnernamen verknüpft.

Hinweis: Folgendes wird demnächst verschwinden: Aus historischen Gründen werden alle oben genannten `mumble-server`-Prozeduren auch unter dem Namenspräfix `murmur-` exportiert. Sie sollten für die Zukunft auf `mumble-server-` umsteigen.

12.9.15 Dateientauschdienste

Im Modul `(gnu services file-sharing)` werden Dienste bereitgestellt, die beim Übertragen von Dateien zwischen Netzwerkteilnehmern untereinander helfen („peer-to-peer“).

Transmission-Daemon-Dienst

Transmission (`https://transmissionbt.com/`) ist ein vielseitiger BitTorrent-Client, der eine Vielzahl grafischer und befehlszeilenbasierter Benutzeroberflächen bereitstellt. Ein Dienst vom Typ `transmission-daemon-service-type` macht die Variante für die Nutzung ohne Oberfläche zugänglich, nämlich `transmission-daemon`, als ein Systemdienst, mit dem Benutzer Dateien über BitTorrent teilen können, selbst wenn sie gerade nicht angemeldet sind.

`transmission-daemon-service-type` [Scheme-Variable]
Der Diensttyp für den BitTorrent-Client „Transmission Daemon“. Sein Wert muss ein `transmission-daemon-configuration`-Verbundsobjekt wie in diesem Beispiel sein:

```
(service transmission-daemon-service-type
         (transmission-daemon-configuration
          ;; Zugriff auf die Steuerungsschnittstelle über
          ;; entfernte Prozeduraufrufe (Remote Procedure Calls)
          (rpc-authentication-required? #t)
          (rpc-username "transmission")
          (rpc-password
           (transmission-password-hash
            "transmission" ; gewünschtes Passwort
            "uKd1uMs9"))   ; beliebiges „Salt“

          ;; Anfragen von diesem Rechner und Rechnern im
          ;; lokalen Netzwerk erlauben
          (rpc-whitelist-enabled? #t)
          (rpc-whitelist '("::1" "127.0.0.1" "192.168.0.*"))
```

```
;; Während der Arbeitszeit die Bandbreite begrenzen
(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 morgens
(alt-speed-time-end
 (+ (* 60 (+ 12 5)) 30))))  ; 5:30 nachmittags / abends
```

Sobald der Dienst gestartet wurde, können Benutzer mit dem Daemon über seine Webschnittstelle interagieren (auf `http://localhost:9091/`) oder indem sie das Werkzeug `transmission-remote` auf der Befehlszeile aufrufen. Es ist Teil des Pakets `transmission`. (Emacs-Nutzer möchten vielleicht einen Blick auf das Paket `emacs-transmission` werfen.) Beide kommunizieren mit dem Daemon über seine Schnittstelle für entfernte Prozeduraufrufe (Remote Procedure Calls, RPC), was nach Vorgabe jedem Nutzer auf dem System zur Verfügung steht. Sie könnten das ändern wollen, indem Sie die Einstellungen `rpc-authentication-required?`, `rpc-username` und `rpc-password` anpassen, wie oben gezeigt und im Folgenden beschrieben.

Als Wert für `rpc-password` muss ein Passwort in der Art angegeben werden, die Transmission-Clients erzeugen und nutzen. Sie können es aus einer bestehenden Datei `settings.json` exakt kopieren, wenn bereits ein anderer Transmission-Client benutzt wurde. Andernfalls können Sie mit den Prozeduren `transmission-password-hash` und `transmission-random-salt` aus diesem Modul einen geeigneten Hash-Wert bestimmen.

`transmission-password-hash` *Passwort Salt* [Scheme-Prozedur]
: Liefert eine Zeichenkette mit dem Hash von *Passwort* zusammen mit dem *Salt* in dem Format, das Clients für Transmission für deren `rpc-password`-Einstellung erkennen.

  Das *Salt* muss eine acht Zeichen lange Zeichenkette sein. Die Prozedur `transmission-random-salt` kann benutzt werden, um einen geeigneten Salt-Wert zufällig zu erzeugen.

`transmission-random-salt` [Scheme-Prozedur]
: Liefert eine Zeichenkette mit einem zufälligen, acht Zeichen langen Salt-Wert von der Art, wie sie Clients für Transmission erzeugen und benutzen. Sie ist dafür geeignet, an die Prozedur `transmission-password-hash` übergeben zu werden.

Diese Prozeduren sind aus einer über den Befehl `guix repl` gestarteten Guile-REPL heraus zugänglich (siehe Abschnitt 9.13 [Aufruf von guix repl], Seite 185). Sie eignen sich dafür, einen zufälligen Salt-Wert zu bekommen, der als der zweite Parameter an „transmission-password-hash“ übergeben werden kann. Zum Beispiel:

```
$ guix repl
scheme@(guix-user)> ,use (gnu services file-sharing)
scheme@(guix-user)> (transmission-random-salt)
$1 = "uKd1uMs9"
```

Alternativ kann ein vollständiger Passwort-Hash in einem einzigen Schritt erzeugt werden:

```
scheme@(guix-user)> (transmission-password-hash "transmission"
(transmission-random-salt))
$2 = "{c8bbc6d1740cd8dc819a6e25563b67812c1c19c9VtFPfdsX"
```

Die sich ergebende Zeichenkette kann so, wie sie ist, als der Wert von `rpc-password` eingesetzt werden. Dadurch kann das Passwort geheim gehalten werden, selbst in einer Betriebssystemkonfiguration.

Vom Daemon heruntergeladene Torrent-Dateien sind nur für die Benutzer in der Benutzergruppe „transmission“ direkt zugänglich. Sie haben auf das in der Einstellung `download-dir` angegebene Verzeichnis nur Lesezugriff (und auf das in `incomplete-dir` angegebene Verzeichnis, sofern `incomplete-dir-enabled?` auf `#t` gesetzt ist). Heruntergeladene Dateien können in beliebige Verzeichnisse verschoben oder ganz gelöscht werden, indem Sie `transmission-remote` mit den Befehlszeilenoptionen `--move` zum Verschieben und `--remove-and-delete` zum Löschen benutzen.

Wenn die Einstellung `watch-dir-enabled?` auf `#t` gesetzt ist, haben die Benutzer in der Gruppe „transmission“ auch die Möglichkeit, `.torrent`-Dateien in dem über `watch-dir` angegebenen Verzeichnis zu platzieren. Dadurch fügt der Daemon die entsprechenden Torrents hinzu. (Mit der Einstellung `trash-original-torrent-files?` wird festgelegt, ob der Daemon diese Dateien löschen soll, wenn er mit ihnen fertig ist.)

Manche der Einstellungen des Daemons können zeitweilig über `transmission-remote` und ähnliche Werkzeuge geändert werden. Sie können solche Änderungen rückgängig machen, indem Sie die `reload`-Aktion des Dienstes aufrufen. Dann lädt der Daemon seine Einstellungen neu von der Platte:

```
# herd reload transmission-daemon
```

Die vollständige Menge der Einstellmöglichkeiten finden Sie in der Definition des Datentyps `transmission-daemon-configuration`.

`transmission-daemon-configuration` [Datentyp]
: Dieser Datentyp repräsentiert die Einstellungen für den Transmission-Daemon. Sie entsprechen direkt den Einstellungen, die Clients für Transmission aus deren Datei `settings.json` erkennen können.

Verfügbare `transmission-daemon-configuration`-Felder sind:

`„package“ transmission` [`transmission-daemon-configuration`-Parameter]
: Das Transmission-Paket, das benutzt werden soll.

`Nichtnegative-ganze-Zahl stop-wait-period` [`transmission-daemon-configuration`-Parameter]
: Wie viele Sekunden gewartet werden soll, wenn der Dienst für `transmission-daemon` gestoppt wird, bis dieser sich beendet, bevor sein Prozess zwangsweise abgewürgt wird. Dadurch wird dem Daemon etwas Zeit gegeben, seine Daten in Ordnung zu bringen und seinen Trackern ein letztes Mal Bescheid zu geben, wenn er herunterfährt. Auf langsamen Rechnern oder Rechnern mit einer langsamen Netzwerkanbindung könnte es besser sein, diesen Wert zu erhöhen.

  Die Vorgabe ist ‘`10`’.

`Zeichenkette download-dir` [`transmission-daemon-configuration`-Parameter]
Das Verzeichnis, wohin Torrent-Dateien heruntergeladen werden.

Die Vorgabe ist '`"/var/lib/transmission-daemon/downloads"`'.

`Boolescher-Ausdruck incomplete-dir-enabled?` [`transmission-daemon-configuration`-Parameter]
Wenn es `#t` ist, werden Dateien in `incomplete-dir` zwischengespeichert, während ihr Torrent heruntergeladen wird, und nach Abschluss des Torrents nach `download-dir` verschoben. Andernfalls werden Dateien für alle Torrents gleich in `download-dir` gespeichert (einschließlich derer, bei denen das Herunterladen noch im Gange ist).

Vorgegeben ist '`#f`'.

`Vielleicht-Zeichenkette incomplete-dir` [`transmission-daemon-configuration`-Parameter]
In welchem Verzeichnis die Dateien bei noch nicht abgeschlossenem Herunterladen zwischengespeichert werden, falls `incomplete-dir-enabled?` auf `#t` gesetzt ist.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`umask umask` [`transmission-daemon-configuration`-Parameter]
Die Dateimodusmaske, mit der heruntergeladene Dateien erzeugt werden. (Siehe die Handbuchseite von `umask` für mehr Informationen.)

Die Vorgabe ist '`18`'.

`Boolescher-Ausdruck rename-partial-files?` [`transmission-daemon-configuration`-Parameter]
Wenn es `#t` ist, wird an die Namen teilweise heruntergeladener Dateien „.part“ angehängt.

Die Vorgabe ist '`#t`'.

`Zuweisungsmodus preallocation` [`transmission-daemon-configuration`-Parameter]
Auf welche Art vorab Speicher für heruntergeladene Dateien zugewiesen werden soll. Entweder `none` (gar nicht), `fast` (schnell) bzw. gleichbedeutend `sparse` (dünnbesetzt/kompakt) oder `full` (gänzlich). Bei `full` fragmentiert die Platte am wenigsten, aber es dauert länger, die Datei zu erzeugen.

Vorgegeben ist '`fast`'.

`Boolescher-Ausdruck watch-dir-enabled?` [`transmission-daemon-configuration`-Parameter]
Ist es `#t`, wird das in `watch-dir` angegebene Verzeichnis beobachtet, ob dort neue `.torrent`-Dateien eingefügt werden. Die darin beschriebenen Torrents werden automatisch hinzugefügt (und die ursprünglichen Dateien werden entfernt, wenn `trash-original-torrent-files?` auf `#t` steht).

Vorgegeben ist '`#f`'.

`Vielleicht-Zeichenkette watch-dir` [`transmission-daemon-configuration`-Parameter]
Welches Verzeichnis beobachtet wird, ob dort neue `.torrent`-Dateien eingefügt werden, die neu hinzuzufügende Torrents anzeigen, falls `watch-dir-enabled` auf `#t` steht.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`trash-original-torrent-files?`
Wenn es `#t` ist, werden `.torrent`-Dateien aus dem beobachteten Verzeichnis gelöscht, sobald ihr Torrent hinzugefügt worden ist (siehe `watch-directory-enabled?`).

Vorgegeben ist ‘`#f`’.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`speed-limit-down-enabled?`
Wenn es `#t` ist, wird die Geschwindigkeit beim Herunterladen durch den Daemon durch die in `speed-limit-down` angegebene Rate beschränkt.

Vorgegeben ist ‘`#f`’.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`speed-limit-down`
Die standardmäßige globale Höchstgeschwindigkeit für das Herunterladen, in Kilobyte pro Sekunde.

Die Vorgabe ist ‘`100`’.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`speed-limit-up-enabled?`
Wenn es `#t` ist, wird die Geschwindigkeit des Daemons beim Hochladen durch die in `speed-limit-up` eingestellte Rate beschränkt.

Vorgegeben ist ‘`#f`’.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`speed-limit-up`
Die standardmäßige globale Höchstgeschwindigkeit für das Hochladen, in Kilobyte pro Sekunde.

Die Vorgabe ist ‘`100`’.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`alt-speed-enabled?`
Wenn es `#t` ist, gelten die alternativen Höchstgeschwindigkeiten `alt-speed-down` und `alt-speed-up` (statt `speed-limit-down` und `speed-limit-up`, wenn sie aktiviert sind), um die Bandbreitennutzung des Daemons einzuschränken. Sie können einen Plan festlegen, zu welchen Zeiten in der Woche sie automatisch aktiviert werden; siehe `alt-speed-time-enabled?`.

Vorgegeben ist ‘`#f`’.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`alt-speed-down`
Die alternative globale Höchstgeschwindigkeit für das Herunterladen, in Kilobyte pro Sekunde.

Die Vorgabe ist ‘`50`’.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`alt-speed-up`

Die alternative globale Höchstgeschwindigkeit für das Hochladen, in Kilobyte pro Sekunde.

Die Vorgabe ist '`50`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`alt-speed-time-enabled?`

Wenn es `#t` ist, werden die alternativen Höchstgeschwindigkeiten `alt-speed-down` und `alt-speed-up` automatisch während bestimmter Zeitperioden aktiviert, entsprechend der Einstellungen für `alt-speed-time-day`, `alt-speed-time-begin` und `alt-speed-time-end`.

Vorgegeben ist '`#f`'.

`Tage-Liste` [`transmission-daemon-configuration`-Parameter]
`alt-speed-time-day`

An welchen Wochentagen die alternative Höchstgeschwindigkeitsplanung benutzt werden soll. Anzugeben ist entweder eine Liste der englischen Namen der Wochentage (`sunday`, `monday` und so weiter) oder eines der Symbole `weekdays` (unter der Woche), `weekends` (an Wochenenden) oder `all`.

Die Vorgabe ist '`all`'.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`alt-speed-time-begin`

Zu welcher Uhrzeit die alternativen Höchstgeschwindigkeiten in Kraft treten, als Anzahl der Minuten seit Mitternacht.

Die Vorgabe ist '`540`'.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`alt-speed-time-end`

Ab welcher Uhrzeit die alternativen Höchstgeschwindigkeiten nicht mehr gelten, als Anzahl der Minuten seit Mitternacht.

Die Vorgabe ist '`1020`'.

`Zeichenkette` [`transmission-daemon-configuration`-Parameter]
`bind-address-ipv4`

Auf welcher IP-Adresse auf Verbindungen von anderen Netzwerkteilnehmern (Peers) gelauscht wird. Benutzen Sie „`0.0.0.0`“, wenn auf allen verfügbaren IP-Adressen gelauscht werden soll.

Die Vorgabe ist '`"0.0.0.0"`'.

`Zeichenkette` [`transmission-daemon-configuration`-Parameter]
`bind-address-ipv6`

Auf welcher IPv6-Adresse auf Verbindungen von anderen Netzwerkteilnehmern (Peers) gelauscht wird. Benutzen Sie „`::`“, wenn auf allen verfügbaren IPv6-Adressen gelauscht werden soll.

Die Vorgabe ist '`"::"`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`peer-port-random-on-start?`

Wenn es `#t` ist, wählt der Daemon bei seinem Start einen Port mit zufälliger Portnummer, auf dem er auf Verbindungen durch andere Netzwerkteilnehmer lauscht. Die Nummer liegt zwischen (einschließlich) den Angaben in `peer-port-random-low` und `peer-port-random-high`. Ansonsten wird er auf dem in `peer-port` angegebenen Port lauschen.

Vorgegeben ist '`#f`'.

`Portnummer` [`transmission-daemon-configuration`-Parameter]
`peer-port-random-low`

Die niedrigste Portnummer, die ausgewählt werden kann, wenn `peer-port-random-on-start?` auf `#t` steht.

Die Vorgabe ist '`49152`'.

`Portnummer` [`transmission-daemon-configuration`-Parameter]
`peer-port-random-high`

Die höchste Portnummer, die ausgewählt werden kann, wenn `peer-port-random-on-start?` auf `#t` steht.

Die Vorgabe ist '`65535`'.

`Portnummer peer-port` [`transmission-daemon-configuration`-Parameter]

Auf welchem Port auf Verbindungsversuche anderer Netzwerkteilnehmer gelauscht wird, wenn `peer-port-random-on-start?` auf `#f` steht.

Die Vorgabe ist '`51413`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`port-forwarding-enabled?`

Wenn es `#t` ist, wird der Daemon versuchen, eine Weiterleitung benötigter Ports beim Internetzugang (Gateway), an dem der Rechner angeschlossen ist, automatisch über UPnP und NAT-PMP einzurichten.

Die Vorgabe ist '`#t`'.

`Verschlüsselungsmodus` [`transmission-daemon-configuration`-Parameter]
`encryption`

Ob Verbindungen zu anderen Netzwerkteilnehmern (Peers) verschlüsselt werden sollen; entweder `prefer-unencrypted-connections` (unverschlüsselte Verbindungen bevorzugen), `prefer-encrypted-connections` (verschlüsselte Verbindungen bevorzugen) oder `require-encrypted-connections` (nur verschlüsselte Verbindungen benutzen).

Die Vorgabe ist '`prefer-encrypted-connections`'.

`Vielleicht-Zeichenkette` [`transmission-daemon-configuration`-Parameter]
`peer-congestion-algorithm`

Der Algorithmus zur TCP-Staukontrolle, der für Verbindungen zu anderen Netzwerkteilnehmern (Peers) verwendet werden soll. Anzugeben ist eine Zeichenkette, die das Betriebssystem für Aufrufe an `setsockopt` zulässt. Wenn nichts angegeben wird, wird die Voreinstellung des Betriebssystems angewandt.

Es ist anzumerken, dass der Kernel auf GNU/Linux-Systemen so konfiguriert werden muss, dass Prozesse zur Nutzung eines nicht standardmäßigen Algorithmus zur Staukontrolle berechtigt sind, andernfalls wird er solche Anfragen mit „Die Operation ist nicht erlaubt“ ablehnen. Um die auf Ihrem System verfügbaren und zur Nutzung freigegebenen Algorithmen zu sehen, schauen Sie sich den Inhalt der Dateien `tcp_available_congestion_control` und `tcp_allowed_congestion_control` im Verzeichnis `/proc/sys/net/ipv4` an.

Wenn Sie zum Beispiel den Transmission-Daemon zusammen mit dem TCP-LP-Algorithmus (`http://www-ece.rice.edu/networks/TCP-LP/`) („TCP Low Priority“) zur Staukontrolle benutzen möchten, müssen Sie Ihre Kernelkonfiguration anpassen, damit Unterstützung für den Algorithmus eingebaut wird. Anschließend müssen Sie Ihre Betriebssystemkonfiguration anpassen, um dessen Nutzung zu erlauben, indem Sie einen Dienst des Typs `sysctl-service-type` hinzufügen (bzw. die Konfiguration des bestehenden Dienstes anpassen). Fügen Sie Zeilen ähnlich der folgenden hinzu:

```
(service sysctl-service-type
         (sysctl-configuration
          (settings
           ("net.ipv4.tcp_allowed_congestion_control" .
            "reno cubic lp"))))
```

Dann können Sie die Konfiguration des Transmission-Daemons aktualisieren zu

```
(peer-congestion-algorithm "lp")
```

und das System rekonfigurieren, damit die Änderungen in Kraft treten.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`TCP-Type-Of-Service` [`transmission-daemon-configuration`-Parameter]
`peer-socket-tos`

Welcher Type-Of-Service in ausgehenden TCP-Paketen angefragt werden soll; entweder `default`, `low-cost`, `throughput`, `low-delay` oder `reliability`.

Die Vorgabe ist ‘`default`’.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`peer-limit-global`

Die globale Höchstgrenze für die Anzahl verbundener Netzwerkteilnehmer (Peers).

Die Vorgabe ist ‘`200`’.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`peer-limit-per-torrent`

Die Höchstgrenze pro Torrent für die Anzahl verbundener Netzwerkteilnehmer (Peers).

Die Vorgabe ist ‘`50`’.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`upload-slots-per-torrent`

An wie viele Netzwerkteilnehmer der Daemon höchstens für denselben Torrent gleichzeitig Daten hochlädt.

Die Vorgabe ist ‘`14`’.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`peer-id-ttl-hours`

Wie viele Stunden die jedem öffentlichen Torrent zugewiesene Peer-ID höchstens benutzt wird, bevor sie neu erzeugt wird.

Die Vorgabe ist '`6`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`blocklist-enabled?`

Wenn es `#t` ist, wird der Daemon in der Sperrliste vorkommende Netzwerkteilnehmer ignorieren. Die Sperrliste lädt er von der `blocklist-url` herunter.

Vorgegeben ist '`#f`'.

`Vielleicht-Zeichenkette` [`transmission-daemon-configuration`-Parameter]
`blocklist-url`

Die URL zu einer Filterliste zu sperrender Netzwerkteilnehmer (Peers) im P2P-Plaintext-Format oder im eMule-`.dat`-Format. Sie wird regelmäßig neu heruntergeladen und angewandt, wenn `blocklist-enabled?` auf `#t` gesetzt ist.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`download-queue-enabled?`

Wenn es `#t` ist, wird der Daemon höchstens `download-queue-size` *nicht* stillstehende Torrents gleichzeitig herunterladen.

Die Vorgabe ist '`#t`'.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`download-queue-size`

Die Größe der Warteschlange herunterzuladender Torrents. Sie ist die Höchstzahl *nicht* stillstehender Torrents, die gleichzeitig heruntergeladen werden, wenn `download-queue-enabled?` auf `#t` gesetzt ist.

Die Vorgabe ist '`5`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`seed-queue-enabled?`

Wenn es `#t` ist, wird der Daemon höchstens `seed-queue-size` *nicht* stillstehende Torrents gleichzeitig verteilen („seeden“).

Vorgegeben ist '`#f`'.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`seed-queue-size`

Die Größe der Warteschlange zu verteilender Torrents. Sie ist die Höchstzahl *nicht* stillstehender Torrents, die gleichzeitig verteilt werden, wenn `seed-queue-enabled?` auf `#t` gesetzt ist.

Die Vorgabe ist '`10`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`queue-stalled-enabled?`

Wenn es `#t` ist, wird der Daemon solche Torrents als stillstehend erachten, für die er in den letzten `queue-stalled-minutes` Minuten *keine* Daten geteilt hat, und diese

bei Beachtung der Beschränkungen in `download-queue-size` und `seed-queue-size` *nicht* berücksichtigen.

Die Vorgabe ist '`#t`'.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`queue-stalled-minutes`

Für wie viele Minuten ein Torrent höchstens inaktiv sein darf, bevor er als stillstehend gilt, sofern `queue-stalled-enabled?` auf `#t` steht.

Die Vorgabe ist '`30`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`ratio-limit-enabled?`

Wenn es `#t` ist, werden Torrents, die verteilt („geseedet") werden, automatisch pausiert, sobald das in `ratio-limit` festgelegte Verhältnis erreicht ist.

Vorgegeben ist '`#f`'.

`Nichtnegative-rationale-Zahl` [`transmission-daemon-configuration`-Parameter]
`ratio-limit`

Das Verhältnis, ab dem ein Torrent, der verteilt wird, pausiert wird, wenn `ratio-limit-enabled?` auf `#t` gesetzt ist.

Die Vorgabe ist '`2.0`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`idle-seeding-limit-enabled?`

Wenn es `#t` ist, werden Torrents, die verteilt werden, automatisch pausiert, sobald sie für `idle-seeding-limit` Minuten inaktiv waren.

Vorgegeben ist '`#f`'.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`idle-seeding-limit`

Wie viele Minuten ein Torrent, der verteilt wird, höchstens inaktiv sein darf, bevor er pausiert wird, wenn `idle-seeding-limit-enabled?` auf `#t` gesetzt ist.

Die Vorgabe ist '`30`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`dht-enabled?`

Das DHT-Protokoll (`http://bittorrent.org/beps/bep_0005.html`) zum Einsatz einer verteilten Hashtabelle (Distributed Hash Table, DHT) zur Unterstützung trackerloser Torrents aktivieren.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`lpd-enabled?`

Local Peer Discovery (`https://en.wikipedia.org/wiki/Local_Peer_Discovery`) (LPD) aktivieren, d.h. Netzwerkteilnehmer im lokalen Netz werden ermittelt, wodurch vielleicht weniger Daten über das öffentliche Internet gesendet werden müssen.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`pex-enabled?`

Peer Exchange (`https://en.wikipedia.org/wiki/Peer_exchange`) (PEX) aktivieren, wodurch der Daemon weniger von externen Trackern abhängt und vielleicht schneller ist.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`utp-enabled?`

Das Mikro-Transport-Protokoll (`http://bittorrent.org/beps/bep_0029.html`) (uTP) aktivieren, damit BitTorrent-Datenverkehr andere Netzwerknutzungen im lokalen Netzwerk weniger beeinträchtigt, die verfügbare Bandbreite aber möglichst voll ausgeschöpft wird.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`rpc-enabled?`

Wenn es `#t` ist, wird die Fernsteuerung des Daemons über die Schnittstelle entfernter Prozeduraufrufe (Remote Procedure Call, RPC) zugelassen. Dadurch kann man ihn über seine Weboberfläche, über den Befehlszeilen-Client `transmission-remote` oder über ähnliche Werkzeuge steuern.

Die Vorgabe ist '`#t`'.

`Zeichenkette` [`transmission-daemon-configuration`-Parameter]
`rpc-bind-address`

Auf welcher IP-Adresse auf RPC-Verbindungen gelauscht wird. Benutzen Sie „`0.0.0.0`“, wenn auf allen verfügbaren IP-Adressen gelauscht werden soll.

Die Vorgabe ist '`"0.0.0.0"`'.

`Portnummer rpc-port` [`transmission-daemon-configuration`-Parameter]

Auf welchem Port auf RPC-Verbindungen gelauscht werden soll.

Die Vorgabe ist '`9091`'.

`Zeichenkette rpc-url` [`transmission-daemon-configuration`-Parameter]

Das Präfix für die Pfade in den URLs des RPC-Endpunkts.

Die Vorgabe ist '`"/transmission/"`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`rpc-authentication-required?`

Wenn es `#t` ist, müssen sich Clients bei der RPC-Schnittstelle authentifizieren (siehe `rpc-username` und `rpc-password`). Anzumerken ist, dass dann die Liste erlaubter Rechnernamen ignoriert wird (siehe `rpc-host-whitelist-enabled?`.

Vorgegeben ist '`#f`'.

`Vielleicht-Zeichenkette` [`transmission-daemon-configuration`-Parameter]
`rpc-username`

Welchen Benutzernamen Clients verwenden müssen, um über die RPC-Schnittstelle zugreifen zu dürfen, wenn `rpc-authentication-required?` auf `#t` gesetzt ist.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Transmissionpassword-Hash` [`transmission-daemon-configuration`-Parameter]
`rpc-password`

Welches Passwort Clients brauchen, um über die RPC-Schnittstelle zugreifen zu dürfen, wenn `rpc-authentication-required?` auf `#t` gesetzt ist. Es muss als Passwort-Hash im von Transmission-Clients erkannten Format angegeben werden. Kopieren Sie es entweder aus einer bestehenden `settings.json`-Datei oder lassen Sie es mit der Prozedur `transmission-password-hash` erzeugen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`rpc-whitelist-enabled?`

Wenn es `#t` ist, werden RPC-Anfragen nur dann akzeptiert, wenn sie von einer in `rpc-whitelist` angegebenen Adresse kommen.

Die Vorgabe ist '`#t`'.

`Zeichenketten-Liste` [`transmission-daemon-configuration`-Parameter]
`rpc-whitelist`

Die Liste der IP- und IPv6-Adressen, von denen RPC-Anfragen angenommen werden, wenn `rpc-whitelist-enabled?` auf `#t` gesetzt ist. Hierbei kann '`*`' als Platzhalter verwendet werden.

Die Vorgabe ist '`("127.0.0.1" "::1")`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`rpc-host-whitelist-enabled?`

Wenn es `#t` ist, werden RPC-Anfragen nur dann angenommen, wenn sie an einen Rechnernamen aus der Liste `rpc-host-whitelist` adressiert sind. Allerdings werden Anfragen an „localhost“ oder „localhost.“ oder eine numerische Adresse immer angenommen, egal was hier eingestellt ist.

Anzumerken ist, dass diese Funktionalität ignoriert wird, wenn `rpc-authentication-required?` auf `#t` gesetzt ist.

Die Vorgabe ist '`#t`'.

`Zeichenketten-Liste` [`transmission-daemon-configuration`-Parameter]
`rpc-host-whitelist`

Die Liste der Rechnernamen, auf die der RPC-Server reagiert, wenn `rpc-host-whitelist-enabled?` auf `#t` gesetzt ist.

Die Vorgabe ist '`()`'.

`Meldungsstufe` [`transmission-daemon-configuration`-Parameter]
`message-level`

Ab welcher Schwerestufe der Daemon Meldungen ins Protokoll (in `/var/log/transmission.log`) aufnimmt, entweder `none` (gar nicht), `error` (nur Fehler), `info` oder `debug`.

Die Vorgabe ist '`info`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`start-added-torrents?`

Wenn es `#t` ist, werden Torrents gestartet, sobald sie hinzugefügt wurden, ansonsten sind sie nach dem Hinzufügen im pausierten Zustand.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`script-torrent-done-enabled?`

Wenn es `#t` ist, wird immer dann, wenn ein Torrent abgeschlossen wurde, das durch `script-torrent-done-filename` angegebene Skript aufgerufen.

Vorgegeben ist '`#f`'.

`Vielleicht-Dateiobjekt` [`transmission-daemon-configuration`-Parameter]
`script-torrent-done-filename`

Ein Dateiname oder ein dateiartiges Objekt eines Skripts, das nach Abschluss eines Torrents aufgerufen werden soll, wenn `script-torrent-done-enabled?` auf `#t` gesetzt ist.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`scrape-paused-torrents-enabled?`

Wenn es `#t` ist, wird der Daemon selbst dann auf Trackern nach einem Torrent suchen, wenn der Torrent pausiert ist.

Die Vorgabe ist '`#t`'.

`Nichtnegative-ganze-Zahl` [`transmission-daemon-configuration`-Parameter]
`cache-size-mb`

Wie viele Megabyte Speicher für den im Arbeitsspeicher liegenden Zwischenspeicher des Daemons reserviert werden sollen. Ein größerer Wert kann eine höhere Leistung bedeuten, indem weniger oft auf die Platte zugegriffen werden muss.

Die Vorgabe ist '`4`'.

`Boolescher-Ausdruck` [`transmission-daemon-configuration`-Parameter]
`prefetch-enabled?`

Wenn es `#t` ist, wird der Daemon versuchen, die Ein-/Ausgabeleistung zu verbessern, indem er dem Betriebssystem Hinweise gibt, auf welche Daten er wahrscheinlich als Nächstes zugreift, um den Anfragen anderer Netzwerkteilnehmer nachzukommen.

Die Vorgabe ist '`#t`'.

12.9.16 Systemüberwachungsdienste

Tailon-Dienst

Tailon (`https://tailon.readthedocs.io/`) ist eine Web-Anwendung, um Protokolldateien zu betrachten und zu durchsuchen.

Das folgende Beispiel zeigt, wie Sie den Dienst mit den Vorgabewerten konfigurieren. Nach Vorgabe kann auf Tailon auf Port 8080 zugegriffen werden (`http://localhost:8080`).

```
(service tailon-service-type)
```

Im folgenden Beispiel werden mehr Anpassungen an der Tailon-Konfiguration vorgenommen: `sed` gehört dort auch zur Liste der erlaubten Befehle dazu.

```
(service tailon-service-type
         (tailon-configuration
           (config-file
             (tailon-configuration-file
               (allowed-commands '("tail" "grep" "awk" "sed"))))))
```

`tailon-configuration` [Datentyp]

Der Datentyp, der die Konfiguration von Tailon repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`config-file` (Vorgabe: `(tailon-configuration-file)`)

Die Konfigurationsdatei, die für Tailon benutzt werden soll. Als Wert kann ein *tailon-configuration-file*-Verbundsobjekt oder ein beliebiger G-Ausdruck dienen (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

Um zum Beispiel stattdessen eine lokale Datei zu benutzen, kann von der Funktion `local-file` Gebrauch gemacht werden.

```
(service tailon-service-type
         (tailon-configuration
           (config-file (local-file "./my-tailon.conf"))))
```

`package` (Vorgabe: `tailon`)

Das tailon-Paket, das benutzt werden soll.

`tailon-configuration-file` [Datentyp]

Datentyp, der die Konfigurationsoptionen für Tailon repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`files` (Vorgabe: `(list "/var/log")`)

Die Liste der anzuzeigenden Dateien. In der Liste dürfen Zeichenketten stehen, die jeweils für eine einzelne Datei oder ein Verzeichnis stehen, oder auch Listen, deren erstes Element den Namen eines Unterbereichs angibt und deren übrige Elemente die Dateien oder Verzeichnisse in diesem Unterbereich benennen.

`bind` (Vorgabe: `"localhost:8080"`)

Adresse und Port, an die sich Tailon binden soll.

`relative-root` (Vorgabe: `#f`)

Welcher URL-Pfad für Tailon benutzt werden soll. Wenn Sie hierfür `#f` angeben, wird kein Pfad benutzt.

`allow-transfers?` (Vorgabe: `#t`)

Ob es möglich sein soll, die Protokolldateien über die Weboberfläche herunterzuladen.

`follow-names?` (Vorgabe: `#t`)

Ob noch nicht existierende Dateien „getailt“ werden können.

`tail-lines` (Vorgabe: `200`)

Wie viele Zeilen am Anfang aus jeder Datei gelesen werden.

`allowed-commands` (Vorgabe: `(list "tail" "grep" "awk")`)
: Welche Befehle ausgeführt werden dürfen. Nach Vorgabe wird `sed` *nicht* erlaubt.

`debug?` (Vorgabe: `#f`)
: Legen Sie `debug?` als `#t` fest, um Nachrichten zur Fehlersuche anzuzeigen.

`wrap-lines` (Vorgabe: `#t`)
: Ob lange Zeilen nach der Anfangseinstellung in der Weboberfläche umgebrochen werden sollen. Setzen Sie es auf `#t`, werden Zeilen in der Anfangseinstellung umgebrochen (die Vorgabe), bei `#f` werden sie anfänglich nicht umgebrochen.

`http-auth` (Vorgabe: `#f`)
: Welcher HTTP-Authentifizierungstyp benutzt werden soll. Setzen Sie dies auf `#f`, damit sich Benutzer nicht authentisieren müssen (die Vorgabe). Unterstützte Werte sind `"digest"` oder `"basic"`.

`users` (Vorgabe: `#f`)
: Wenn HTTP-Authentifizierung aktiviert ist (siehe `http-auth`), wird der Zugriff nur gewährt, nachdem die hier angegebenen Zugangsinformationen eingegeben wurden. Um Nutzer hinzuzufügen, geben Sie hier eine Liste von Paaren an, deren erstes Element jeweils der Benutzername und deren zweites Element das Passwort ist.

```
(tailon-configuration-file
  (http-auth "basic")
  (users     '(("benutzer1" . "passwort1")
               ("benutzer2" . "passwort2"))))
```

Darkstat-Dienst

Darkstat ist ein Netzwerkanalyseprogramm, das Pakete im Datenverkehr aufzeichnet, Statistiken zur Netzwerknutzung berechnet und über HTTP Berichte dazu bereitstellt.

`Scheme-Variable` *darkstat-service-type* [Variable]
: Dies ist der Diensttyp für den darkstat-Dienst (`https://unix4lyfe.org/darkstat/`). Sein Wert muss ein `darkstat-configuration`-Verbundsobjekt sein, wie in diesem Beispiel:

```
(service darkstat-service-type
         (darkstat-configuration
           (interface "eno1")))
```

`darkstat-configuration` [Datentyp]
: Datentyp, der die Konfiguration von `darkstat` repräsentiert.

`package` (Vorgabe: `darkstat`)
: Welches darkstat-Paket verwendet werden soll.

`interface`
: Datenverkehr an der angegebenen Netzwerkschnittstelle mitschneiden.

`port` (Vorgabe: `"667"`)
: Bindet die Weboberfläche an den angegebenen Port.

`bind-address` (Vorgabe: `"127.0.0.1"`)
: Bindet die Weboberfläche an die angegebene Adresse.

`base` (Vorgabe: `"/"`)
: Geben Sie den Pfad der Basis-URL an. Das kann nützlich sein, wenn auf `darkstat` über einen inversen Proxy („Reverse Proxy“) zugegriffen wird.

Prometheus-Node-Exporter-Dienst

Der Prometheus-„Node-Exporter“ stellt Statistiken über Hardware und Betriebssystem für das Prometheus-Systemüberwachungssystem bereit, die vom Linux-Kernel geliefert werden. Dieser Dienst sollte auf allen physischen Netzwerkknoten und virtuellen Maschinen installiert werden, für die eine Überwachung ihrer Statistiken gewünscht wird.

`Scheme-Variable` *prometheus-node-exporter-service-type* [Variable]
Dies ist der Diensttyp für den „prometheus-node-exporter“-Dienst (`https://github.com/prometheus/node_exporter/`). Sein Wert muss ein `prometheus-node-exporter-configuration`-Verbundsobjekt sein.

```
(service prometheus-node-exporter-service-type)
```

`prometheus-node-exporter-configuration` [Datentyp]
Repräsentiert die Konfiguration von `node_exporter`.

`package` (Vorgabe: `go-github-com-prometheus-node-exporter`)
: Das Paket für den prometheus-node-exporter, was benutzt werden soll.

`web-listen-address` (Vorgabe: `":9100"`)
: Bindet die Weboberfläche an die angegebene Adresse.

`textfile-directory` (Vorgabe: `"/var/lib/prometheus/node-exporter"`)
: In dieses Verzeichnis können maschinenspezifische Metriken exportiert werden. Hier hinein sollten Dateien mit Metriken im Textformat platziert werden, die die Dateiendung `.prom` haben.

`extra-options` (Vorgabe: `'()`)
: Zusätzliche Befehlszeilenoptionen, die an den prometheus-node-exporter übergeben werden sollen.

Zabbix-Server

Zabbix ist ein hochleistungsfähiges Überwachungssystem, mit dem Daten von vielen Quellen gesammelt in einer Web-Oberfläche zur Schau gestellt werden. Alarm- und Berichtsfunktionen sind eingebaut; auch gibt es *Vorlagen* für gängige Betriebssystemmetriken, unter anderem die Netzwerk- und Prozessorauslastung sowie den Plattenplatzverbrauch.

In diesem Diensttyp steckt das zentrale Überwachungssystem von Zabbix. Zudem brauchen Sie [zabbix-front-end], Seite 457, um Zabbix zu konfigurieren und Ergebnisse anzuzeigen, und vielleicht [zabbix-agent], Seite 455, auf den zu überwachenden Maschinen (auch andere Datenquellen werden unterstützt wie etwa [prometheus-node-exporter], Seite 454).

`Scheme-Variable` *zabbix-server-service-type* [Variable]
Der Diensttyp des Zabbix-Server-Dienstes. Sein Wert muss ein `zabbix-server-configuration`-Verbundsobjekt sein. Folgendes ist enthalten.

`zabbix-server-configuration` [Datentyp]

Verfügbare `zabbix-server-configuration`-Felder sind:

`zabbix-server` (Vorgabe: `zabbix-server`) (Typ: dateiartig)
: Das zabbix-server-Paket.

`user` (Vorgabe: `"zabbix"`) (Typ: Zeichenkette)
: Das Benutzerkonto, mit dem der Zabbix-Server ausgeführt wird.

`group` (Vorgabe: `"zabbix"`) (Typ: Gruppe)
: Die Gruppe, mit der der Zabbix-Server ausgeführt wird.

`db-host` (Vorgabe: `"127.0.0.1"`) (Typ: Zeichenkette)
: Rechnername der Datenbank.

`db-name` (Vorgabe: `"zabbix"`) (Typ: Zeichenkette)
: Datenbankname.

`db-user` (Vorgabe: `"zabbix"`) (Typ: Zeichenkette)
: Benutzerkonto der Datenbank.

`db-password` (Vorgabe: `""`) (Typ: Zeichenkette)
: Das Datenbankpasswort. Bitte benutzen Sie stattdessen `include-files` mit `DBPassword=SECRET` in einer angegebenen Datei.

`db-port` (Vorgabe: `5432`) (Typ: Zahl)
: Datenbank-Portnummer.

`log-type` (Vorgabe: `""`) (Typ: Zeichenkette)
: Gibt an, wohin Protokollnachrichten geschrieben werden.
  - `system` - Syslog.
  - `file` - Die im `log-file`-Parameter angegebene Datei.
  - `console` - Standardausgabe.

`log-file` (Vorgabe: `"/var/log/zabbix/server.log"`) (Typ: Zeichenkette)
: Protokolldateiname für den `file`-Parameter von `log-type`.

`pid-file` (Vorgabe: `"/var/run/zabbix/zabbix_server.pid"`) (Typ: Zeichenkette)
: Name der PID-Datei.

`ssl-ca-location` (Vorgabe: `"/etc/ssl/certs/ca-certificates.crt"`) (Typ: Zeichenkette)
: Der Ort mit den Dateien über die Zertifikatsautoritäten (Certificate Authoritys, CAs) zur Prüfung der SSL-Serverzertifikate.

`ssl-cert-location` (Vorgabe: `"/etc/ssl/certs"`) (Typ: Zeichenkette)
: Der Ort mit den SSL-Client-Zertifikaten.

`extra-options` (Vorgabe: `""`) (Typ: Zusatzoptionen)
: Zusätzliche Optionen werden an die Zabbix-Server-Konfigurationsdatei angehängt.

`include-files` (Vorgabe: `()`) (Typ: Einzubindende-Dateien)
: Sie können einzelne Dateien oder alle Dateien in einem Verzeichnis in die Konfigurationsdatei einbinden.

Zabbix-Agent

Der Zabbix-Agent sammelt Informationen über das laufende System zur Überwachung mit dem Zabbix-Server. Einige Überprüfungen sind von vornherein verfügbar; eigene können Sie über *Benutzerparameter* (`https://www.zabbix.com/documentation/current/en/manual/config/items/userparameters`) hinzufügen.

`Scheme-Variable` *zabbix-agent-service-type* [Variable]
Der Diensttyp des Zabbix-Agent-Dienstes. Sein Wert muss ein `zabbix-agent-configuration`-Verbundsobjekt sein. Folgendes ist enthalten.

`zabbix-agent-configuration` [Datentyp]
Verfügbare `zabbix-agent-configuration`-Felder sind:

`zabbix-agent` (Vorgabe: `zabbix-agentd`) (Typ: dateiartig)
Das zabbix-agent-Paket.

`user` (Vorgabe: `"zabbix"`) (Typ: Zeichenkette)
Das Benutzerkonto, mit dem der Zabbix-Agent ausgeführt wird.

`group` (Vorgabe: `"zabbix"`) (Typ: Gruppe)
Die Gruppe, mit der der Zabbix-Agent ausgeführt wird.

`hostname` (Vorgabe: `""`) (Typ: Zeichenkette)
Der eindeutige Rechnername in richtiger Groß-/Kleinschreibung, der für aktive Überprüfungen benötigt wird und dem im Server eingestellten Rechnernamen entsprechen muss.

`log-type` (Vorgabe: `""`) (Typ: Zeichenkette)
Gibt an, wohin Protokollnachrichten geschrieben werden.

- `system` - Syslog.
- `file` - Die Datei aus dem `log-file`-Parameter.
- `console` - Standardausgabe.

`log-file` (Vorgabe: `"/var/log/zabbix/agent.log"`) (Typ: Zeichenkette)
Protokolldateiname für den `file`-Parameter von `log-type`.

`pid-file` (Vorgabe: `"/var/run/zabbix/zabbix_agent.pid"`) (Typ: Zeichenkette)
Name der PID-Datei.

`server` (Vorgabe: `("127.0.0.1")`) (Typ: Liste)
Die Liste der IP-Adressen, optional in CIDR-Notation angegeben, oder die Rechnernamen von Zabbix-Servern und Zabbix-Proxys. Eingehende Verbindungen werden nur dann angenommen, wenn sie von hier angegebenen Rechnern stammen.

`server-active` (Vorgabe: `("127.0.0.1")`) (Typ: Liste)
Die Liste aus IP:Port-Paaren (oder Rechnername:Port-Paaren) von Zabbix-Servern und Zabbix-Proxys für aktive Überprüfungen. Wenn kein Port angegeben wurde, wird der Vorgabeport benutzt. Wenn dieser Parameter *nicht* angegeben wird, werden aktive Überprüfungen deaktiviert.

`extra-options` (Vorgabe: `""`) (Typ: Zusatzoptionen)
: Zusätzliche Optionen werden an die Zabbix-Server-Konfigurationsdatei angehängt.

`include-files` (Vorgabe: `()`) (Typ: Einzubindende-Dateien)
: Sie können einzelne Dateien oder alle Dateien in einem Verzeichnis in die Konfigurationsdatei einbinden.

Zabbix-Frontend

Mit dem Zabbix-Frontend wird eine Weboberfläche als Vordergrundsystem (*Frontend*) für Zabbix bereitgestellt. Es kann auch auf einer anderen Maschine als der Zabbix-Server laufen. Dieser Diensttyp beruht darauf, die Dienste für [PHP-FPM], Seite 480, und [NGINX], Seite 469, mit der nötigen Konfiguration zum Laden der Zabbix-Benutzeroberfläche zu erweitern.

`Scheme-Variable` *zabbix-front-end-service-type* [Variable]
: Dieser Diensttyp stellt das Zabbix-Web-Frontend als Vordergrundsystem zur Verfügung. Sein Wert muss ein `zabbix-front-end-configuration`-Verbundsobjekt sein wie hier gezeigt.

`zabbix-front-end-configuration` [Datentyp]
: Verfügbare `zabbix-front-end-configuration`-Felder sind:

  `zabbix-server` (Vorgabe: `zabbix-server`) (Typ: dateiartig)
  : Das zabbix-server-Paket, das benutzt werden soll.

  `nginx` (Vorgabe: `()`) (Typ: Liste)
  : Liste von [nginx-server-configuration], Seite 473-Blöcken für das Zabbix-Frontend. Wenn es leer gelassen wird, lässt die Vorgabeeinstellung auf Port 80 lauschen.

  `db-host` (Vorgabe: `"localhost"`) (Typ: Zeichenkette)
  : Rechnername der Datenbank.

  `db-port` (Vorgabe: `5432`) (Typ: Zahl)
  : Datenbank-Portnummer.

  `db-name` (Vorgabe: `"zabbix"`) (Typ: Zeichenkette)
  : Datenbankname.

  `db-user` (Vorgabe: `"zabbix"`) (Typ: Zeichenkette)
  : Benutzerkonto der Datenbank.

  `db-password` (Vorgabe: `""`) (Typ: Zeichenkette)
  : Das Datenbankpasswort. Bitte benutzen Sie stattdessen `db-secret-file`.

  `db-secret-file` (Vorgabe: `""`) (Typ: Zeichenkette)
  : Die Datei mit den Geheimnis-Informationen, die an die `zabbix.conf.php`-Datei angehängt wird. Diese Datei enthält Zugangsdaten für die Nutzung durch das Zabbix-Frontend. Es wird von Ihnen erwartet, dass Sie sie manuell erzeugen.

`zabbix-host` (Vorgabe: `"localhost"`) (Typ: Zeichenkette)
: Zabbix-Server-Rechnername.

`zabbix-port` (Vorgabe: `10051`) (Typ: Zahl)
: Zabbix-Server-Port.

12.9.17 Kerberos-Dienste

Das `(gnu services kerberos)`-Modul stellt Dienste zur Verfügung, die mit dem Authentifizierungsprotokoll *Kerberos* zu tun haben.

Krb5-Dienst

Programme, die eine Kerberos-Clientbibliothek benutzen, erwarten meist, dass sich eine Konfigurationsdatei in `/etc/krb5.conf` befindet. Dieser Dienst erzeugt eine solche Datei aus einer Definition, die in der Betriebssystemdeklaration angegebenen wurde. Durch ihn wird kein Daemon gestartet.

Keine „Schlüsseltabellen“-Dateien werden durch diesen Dienst zur Verfügung gestellt – Sie müssen sie ausdrücklich selbst anlegen. Dieser Dienst funktioniert bekanntermaßen mit der MIT-Clientbibliothek `mit-krb5`. Andere Implementierungen wurden nicht getestet.

`krb5-service-type` [Scheme-Variable]
: Ein Diensttyp für Kerberos-5-Clients.

Hier ist ein Beispiel, wie man ihn benutzt:

```
(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"))))))
```

Dieses Beispiel stellt eine Client-Konfiguration für Kerberos 5 zur Verfügung, mit der:

- Zwei Administrationsbereiche erkannt werden, nämlich: „EXAMPLE.COM“ und „ARGRX.EDU“, die beide verschiedene Administrationsserver und Schlüsselverteilungszentren haben,
- als Vorgabe der Administrationsbereich „EXAMPLE.COM“ verwendet wird, falls der Administrationsbereich von Clients nicht ausdrücklich angegeben wurde, und
- auch Dienste angenommen werden, die nur Verschlüsselungstypen unterstützen, die bekanntermaßen schwach sind.

Die Typen `krb5-realm` und `krb5-configuration` haben viele Felder. Hier werden nur die am häufigsten benutzten beschrieben. Eine vollständige Liste und jeweils detailliertere Erklärungen finden Sie in der Dokumentation von krb5.conf vom MIT.

`krb5-realm` [Datentyp]

`name` Dieses Feld enthält eine Zeichenkette, die den Namen des Administrationsbereichs bezeichnet. Üblich ist, den vollständigen DNS-Namen („Fully Qualified DNS Name“) Ihrer Organisation nur in Großbuchstaben zu benutzen.

`admin-server`
Dieses Feld enthält eine Zeichenkette, die den Rechner benennt, auf dem der Administrationsserver läuft.

`kdc` Dieses Feld enthält eine Zeichenkette, die das Schlüsselverteilungszentrum für den Administrationsbereich angibt.

`krb5-configuration` [Datentyp]

`allow-weak-crypto?` (Vorgabe: `#f`)
Wenn diese Option auf `#t` gesetzt ist, werden auch Dienste akzeptiert, die nur Verschlüsselungsalgorithmen anbieten, die bekanntermaßen schwach sind.

`default-realm` (Vorgabe: `#f`)
Dieses Feld sollte eine Zeichenkette enthalten, die den voreingestellten Kerberos-Administrationsbereich für den Client angibt. Sie sollten in diesem Feld den Namen Ihres Kerberos-Administrationsbereichs eintragen. Wenn der Wert `#f` ist, dann muss ein Administrationsbereich mit jedem Kerberos-Prinzipal zusammen angegeben werden, wenn Programme wie `kinit` aufgerufen werden.

`realms` Hierin sollte eine nichtleere Liste von je einem `krb5-realm`-Objekt pro Administrationsbereich stehen, auf den Clients zugreifen können. Normalerweise hat einer davon ein `name`-Feld, das mit dem `default-realm`-Feld übereinstimmt.

PAM-krb5-Dienst

Der `pam-krb5`-Dienst ermöglicht es, bei der Anmeldung und Passwortverwaltung Benutzer über Kerberos zu authentifizieren. Sie brauchen diesen Dienst, damit Anwendungen, die PAM benutzen können, Nutzer über Kerberos authentifizieren können.

`pam-krb5-service-type` [Scheme-Variable]
Ein Diensttyp für das PAM-Modul zu Kerberos 5.

`pam-krb5-configuration` [Datentyp]
Der Datentyp, der die Konfiguration des PAM-Moduls für Kerberos 5 repräsentiert. Dieser Typ hat die folgenden Parameter:

`pam-krb5` (Vorgabe: `pam-krb5`)
Das pam-krb5-Paket, das benutzt werden soll.

`minimum-uid` (Vorgabe: `1000`)
Der kleinste Benutzeridentifikator (UID), für den Authentifizierung über Kerberos versucht werden soll. Lokale Benutzerkonten mit niedrigeren Zahlwerten können sich nicht authentisieren und bekommen dazu keine Meldung angezeigt.

12.9.18 LDAP-Dienste

Das Modul `(gnu services authentication)` stellt den Diensttyp `nslcd-service-type` zur Verfügung, mit dem sich Benutzer gegenüber einem LDAP-Server authentisieren können. Sie möchten dabei wahrscheinlich nicht nur den Dienst konfigurieren, sondern auch `ldap` als einen Namensdienst („Name Service“) für den Name Service Switch hinzufügen. Siehe Abschnitt 12.12 [Name Service Switch], Seite 607, für Details.

Hier ist ein Beispiel für eine einfache Betriebssystemdeklaration mit einer der Vorgabe entsprechenden Konfiguration des `nslcd-service-type` und einer Konfiguration des Name Service Switch, die den `ldap`-Namensdienst zuletzt prüft:

```
(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)))))
```

Verfügbare `nslcd-configuration`-Felder sind:

`„package“ nss-pam-ldapd` [`nslcd-configuration`-Parameter]
: Das `nss-pam-ldapd`-Paket, was benutzt werden soll.

`Vielleicht-Zahl threads` [`nslcd-configuration`-Parameter]
: Die Anzahl zu startender Threads, die Anfragen bearbeiten und LDAP-Abfragen durchführen können. Jeder Thread öffnet eine separate Verbindung zum LDAP-Server. Die Vorgabe ist, 5 Threads zu starten.

  Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Zeichenkette uid` [`nslcd-configuration`-Parameter]
: Gibt den Benutzeridentifikator an, unter dem der Daemon ausgeführt werden soll.

  Die Vorgabe ist ‘`"nslcd"`’.

`Zeichenkette gid` [`nslcd-configuration`-Parameter]
: Gibt den Gruppenidentifikator an, unter dem der Daemon ausgeführt werden soll.

Die Vorgabe ist '`"nslcd"`'.

`Protokolleinstellung log` [`nslcd-configuration`-Parameter]
Diese Einstellung steuert über eine Liste aus SCHEMA und STUFE, wie protokolliert wird. Als SCHEMA-Argument darf entweder eines der Symbole '`none`' (keines) oder '`syslog`' angegeben werden oder ein absoluter Dateiname. Das Argument STUFE ist optional und legt die Protokollierungsstufe fest. Die Protokollierungsstufe kann als eines der folgenden Symbole angegeben werden: '`crit`' (kritisch), '`error`' (Fehler), '`warning`' (Warnung), '`notice`' (Benachrichtigung), '`info`' (Information) oder '`debug`' (Fehlersuche). Alle Mitteilungen mit der angegebenen Protokollierungsstufe oder einer höheren werden protokolliert.

Die Vorgabe ist '`("/var/log/nslcd" info)`'.

`Liste uri` [`nslcd-configuration`-Parameter]
Die Liste der LDAP-Server-URIs. Normalerweise wird nur der erste Server benutzt; nachfolgende Server dienen als Ersatz.

Die Vorgabe ist '`("ldap://localhost:389/")`'.

`Vielleicht-Zeichenkette ldap-version` [`nslcd-configuration`-Parameter]
Die zu benutzende Version des LDAP-Protokolls. Nach Vorgabe wird die höchste Version benutzt, die von der LDAP-Bibliothek unterstützt wird.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette binddn` [`nslcd-configuration`-Parameter]
Gibt den „Distinguished Name“ an, der an den Verzeichnisserver („Directory Server“) gebunden wird, um Einträge aufzulösen. Nach Vorgabe wird anonym gebunden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette bindpw` [`nslcd-configuration`-Parameter]
Gibt die Zugangsinformationen an, mit denen gebunden wird. Diese Einstellung ist nur dann wirksam, wenn sie mit binddn benutzt wird.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette rootpwmoddn` [`nslcd-configuration`-Parameter]
Gibt den „Distinguished Name“ an, der benutzt wird, wenn der Administratornutzer „root“ versucht, das Passwort eines Benutzers mit Hilfe des PAM-Moduls zu verändern.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette rootpwmodpw` [`nslcd-configuration`-Parameter]
Gibt die Zugangsinformationen an, die benutzt werden, wenn der Administratornutzer „root“ versucht, das Passwort eines Benutzers zu verändern. Diese Einstellung ist nur dann wirksam, wenn sie mit rootpwmoddn benutzt wird.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette sasl-mech` [`nslcd-configuration`-Parameter]
Gibt an, welcher SASL-Mechanismus benutzt werden soll, um Authentifizierung über SASL durchzuführen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette sasl-realm` [`nslcd-configuration`-Parameter]
Gibt den SASL-Administrationsbereich an, um Authentifizierungen über SASL durchzuführen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette sasl-authcid` [`nslcd-configuration`-Parameter]
Gibt die Authentisierungsidentität an, um Authentifizierungen über SASL durchzuführen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette sasl-authzid` [`nslcd-configuration`-Parameter]
Gibt die Autorisierungsidentität an, um Authentifizierungen über SASL durchzuführen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck sasl-canonicalize?` [`nslcd-configuration`-Parameter]
Legt fest, ob der kanonische Rechnername („Hostname“) des LDAP-Servers ermittelt werden soll. Wenn ja, wird die LDAP-Bibliothek eine inverse Auflösung („Reverse Lookup“) des Rechnernamens durchführen. Die Vorgabe ist, es der LDAP-Bibliothek zu überlassen, ob eine solche Überprüfung durchgeführt wird.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette krb5-ccname` [`nslcd-configuration`-Parameter]
Legt den Namen für den Zwischenspeicher der GSS-API-Kerberos-Zugangsdaten fest.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Zeichenkette base` [`nslcd-configuration`-Parameter]
Basis für die Verzeichnissuche.

Vorgegeben ist '`"dc=example,dc=com"`'.

`Suchbereichs-Einstellung scope` [`nslcd-configuration`-Parameter]
Legt den Suchbereich fest als subtree (Teilbaum), onelevel (eine Ebene), base (Basis) oder children (Kinder). Die Vorgabe für den Suchbereich ist subtree; base ist fast nie geeignet für Namensdienstauflösungen; children wird nicht auf allen Servern unterstützt.

Die Vorgabe ist '`(subtree)`'.

`Vielleicht-Deref-Einstellung deref` [`nslcd-configuration`-Parameter]
Legt die Richtlinie für das Dereferenzieren von Alias-Namen fest. Die vorgegebene Richtlinie ist, Alias-Namen niemals zu dereferenzieren.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck referrals` [`nslcd-configuration`-Parameter]
Gibt an, ob Verweise („Referrals“) automatisch verfolgt werden sollen. Das vorgegebene Verhalten ist, sie zu verfolgen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Liste-von-Abbildungseinträgen maps` [`nslcd-configuration`-Parameter]
Diese Option ermöglicht es, eigene Attribute bei der Auflösung anstelle der vorgegebenen RFC-2307-Attribute zu verwenden. Es ist eine Liste von Abbildungen („Maps“), von denen jede aus dem Namen der Abbildung, dem abzubildenden RFC-2307-Attribut und einem Anfrageausdruck besteht, mit dem es anhand des Verzeichnisses aufgelöst wird.

Die Vorgabe ist ‘`()`’.

`Liste-von-Filtereinträgen filters` [`nslcd-configuration`-Parameter]
Eine Liste von Filtern, von denen jeder aus dem Namen einer Abbildung, auf die sich der Filter auswirkt, und einem LDAP-Suchfilter-Ausdruck besteht.

Die Vorgabe ist ‘`()`’.

`Vielleicht-Zahl bind-timelimit` [`nslcd-configuration`-Parameter]
Gibt die Zeitbeschränkung in Sekunden an, wie lange eine Verbindung zum Verzeichnisserver dauern darf. Die Vorgabe ist 10 Sekunden.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zahl timelimit` [`nslcd-configuration`-Parameter]
Gibt die Zeitbeschränkung (in Sekunden) an, wie lange auf eine Antwort vom LDAP-Server gewartet wird. Ein Wert von null, was die Vorgabe ist, bewirkt, dass beliebig lange gewartet wird, bis Suchen abgeschlossen sind.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zahl idle-timelimit` [`nslcd-configuration`-Parameter]
Gibt an, wie lange bei Inaktivität gewartet wird (in Sekunden), bis die Verbindung zum LDAP-Server geschlossen wird. Die Vorgabe ist, dass es zu keiner Zeitüberschreitung bei Verbindungen kommen kann.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zahl reconnect-sleeptime` [`nslcd-configuration`-Parameter]
Gibt die Anzahl an Sekunden an, wie lange „schlafend“ gewartet wird, wenn zu *keinem* LDAP-Server eine Verbindung hergestellt werden kann. Die Vorgabe ist, zwischen dem ersten Fehlversuch und dem ersten neuen Versuch 1 Sekunde zu warten.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zahl reconnect-retrytime` [`nslcd-configuration`-Parameter]
Gibt an, nach wie viel Zeit der LDAP-Server als dauerhaft nicht verfügbar angesehen wird. Sobald dieser Fall eintritt, wird eine Verbindungsaufnahme nur noch einmal pro weiterem Ablauf dieser Zeitperiode versucht. Der Vorgabewert beträgt 10 Sekunden.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-SSL-Einstellung ssl` [`nslcd-configuration`-Parameter]
Gibt an, ob SSL/TLS benutzt werden soll oder nicht (die Vorgabe ist, es *nicht* zu benutzen). Wenn 'start-tls angegeben wird, dann wird StartTLS statt schlichtem LDAP über SSL benutzt.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-„tls-reqcert“-Einstellung` [`nslcd-configuration`-Parameter]
`tls-reqcert`

Gibt an, welche Überprüfungen auf einem vom Server empfangenen Zertifikat durchgeführt werden sollen. Die Bedeutung der Werte wird auf der Handbuchseite zu ldap.conf(5) beschrieben.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette tls-cacertdir` [`nslcd-configuration`-Parameter]
Gibt das Verzeichnis an, das X.509-Zertifikate zur Authentifikation von Kommunikationspartnern enthält. Dieser Parameter wird ignoriert, wenn Sie GnuTLS benutzen lassen.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette tls-cacertfile` [`nslcd-configuration`-Parameter]
Gibt den Dateipfad zu dem X.509-Zertifikat zur Authentifikation von Kommunikationspartnern an.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette tls-randfile` [`nslcd-configuration`-Parameter]
Gibt den Pfad zu einer Entropiequelle an. Dieser Parameter wird ignoriert, wenn Sie GnuTLS benutzen lassen.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette tls-ciphers` [`nslcd-configuration`-Parameter]
Gibt als eine Zeichenkette an, welche Ciphers für TLS benutzt werden sollen.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette tls-cert` [`nslcd-configuration`-Parameter]
Gibt den Pfad zu der Datei an, die das lokale Zertifikat zur TLS-Authentisierung als Client enthält.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette tls-key` [`nslcd-configuration`-Parameter]
Gibt den Pfad zu der Datei an, die den privaten Schlüssel zur TLS-Authentisierung als Client enthält.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zahl pagesize` [`nslcd-configuration`-Parameter]
Geben Sie hier eine Zahl größer als 0 an, um beim LDAP-Server seitenweise Antworten anzufordern, entsprechend RFC2696. Die Vorgabe (0) fordert alle Ergebnisse auf einmal an.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-„ignore-users“-Einstellung` [`nslcd-configuration`-Parameter]
`nss-initgroups-ignoreusers`

Diese Einstellung verhindert, dass für die angegebenen Benutzer die Gruppenmitgliedschaft über LDAP aufgelöst wird. Alternativ kann der Wert 'all-local verwendet werden. Für diesen Wert erzeugt nslcd eine vollständige Liste aller Nicht-LDAP-Benutzer, wenn es startet.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zahl nss-min-uid` [`nslcd-configuration`-Parameter]
Diese Einstellung lässt sicherstellen, dass LDAP-Benutzer, deren numerischer Benutzeridentifikator kleiner als der angegebene Wert ist, ignoriert werden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zahl nss-uid-offset` [`nslcd-configuration`-Parameter]
Diese Einstellung gibt einen Versatz an, der auf den numerischen Benutzeridentifikator jedes LDAP-Nutzers aufaddiert wird. Damit können Konflikte zwischen den Benutzeridentifikatoren lokaler Benutzerkonten und LDAP vermieden werden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zahl nss-gid-offset` [`nslcd-configuration`-Parameter]
Diese Einstellung gibt einen Versatz an, der auf den numerischen Gruppenidentifikator jedes LDAP-Nutzers aufaddiert wird. Damit können Konflikte zwischen den Gruppenidentifikatoren lokaler Gruppen und LDAP vermieden werden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck` [`nslcd-configuration`-Parameter]
`nss-nested-groups`
Wenn diese Einstellung aktiviert ist, können die Attribute einer Gruppe auch wieder Verweise auf eine andere Gruppe sein. Attribute darin verschachtelter („nested“) Gruppen werden für die Gruppe auf höherer Ebene ebenfalls zurückgeliefert und Elterngruppen werden zurückgeliefert, wenn nach den Gruppen eines bestimmten Nutzers gesucht wird. Die Vorgabe ist, keine zusätzlichen Suchen nach verschachtelten Gruppen durchzuführen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck` [`nslcd-configuration`-Parameter]
`nss-getgrent-skipmembers`
Wenn diese Einstellung aktiviert ist, wird die Liste der Gruppenmitglieder beim Auflösen von Gruppen nicht angefragt. Zu welchen Gruppen ein Benutzer gehört, kann weiterhin angefragt werden, damit dem Benutzer bei der Anmeldung wahrscheinlich dennoch die richtigen Gruppen zugeordnet werden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck` [`nslcd-configuration`-Parameter]
`nss-disable-enumeration`
Wenn diese Einstellung aktiviert ist, scheitern Funktionen, die alle Benutzer-/Gruppeneinträge aus dem Verzeichnis zu laden versuchen. Dadurch kann die Auslastung von LDAP-Servern wesentlich reduziert werden, wenn es eine große Anzahl von Benutzern und/oder Gruppen gibt. Diese Einstellung wird für die meisten Konfigurationen *nicht* empfohlen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Zeichenkette validnames` [`nslcd-configuration`-Parameter]
Mit dieser Einstellung kann festgelegt werden, wie Benutzer- und Gruppennamen vom System geprüft werden. Das angegebene Muster wird zur Prüfung aller Benutzer- und Gruppennamen benutzt, die über LDAP angefragt und zurückgeliefert werden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck` [`nslcd-configuration`-Parameter]
`ignorecase`

Hiermit wird festgelegt, ob bei Suchen nach passenden Einträgen *nicht* auf Groß- und Kleinschreibung geachtet wird. Wenn Sie dies aktivieren, könnte es zu Sicherheitslücken kommen, mit denen Autorisierungen umgangen („Authorization Bypass“) oder der nscd-Zwischenspeicher vergiftet werden kann („Cache Poisoning“), was gezielte Überlastungen ermöglichen würde („Denial of Service“).

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck` [`nslcd-configuration`-Parameter]
`pam-authc-ppolicy`

Mit dieser Einstellung wird festgelegt, ob Passwortrichtliniensteuerung vom LDAP-Server angefragt und behandelt wird, wenn Nutzer authentifiziert werden.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette` [`nslcd-configuration`-Parameter]
`pam-authc-search`

Nach Vorgabe führt nslcd eine LDAP-Suche nach jeder BIND-Operation (zur Authentisierung) durch, um sicherzustellen, dass die BIND-Operation erfolgreich durchgeführt wurde. Die vorgegebene Suche ist eine einfache Überprüfung, ob der DN eines Benutzers existiert. Hier kann ein Suchfilter angegeben werden, der stattdessen benutzt werden soll. Er sollte mindestens einen Eintrag liefern.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette` [`nslcd-configuration`-Parameter]
`pam-authz-search`

Diese Einstellung ermöglicht flexible Feineinstellungen an der durchzuführenden Autorisierungsprüfung. Der angegebene Suchfilter wird ausgeführt, woraufhin Zugriff gewährt wird, wenn mindestens ein Eintrag passt, andernfall wird der Zugriff verweigert.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Zeichenkette` [`nslcd-configuration`-Parameter]
`pam-password-prohibit-message`

Wenn diese Einstellung festgelegt wurde, werden Passwortänderungen über pam_ldap abgelehnt und dem Anwender wird stattdessen die festgelegte Nachricht gezeigt. Die Nachricht kann benutzt werden, um den Anwender auf alternative Methoden aufmerksam zu machen, wie er sein Passwort ändern kann.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Liste pam-services` [`nslcd-configuration`-Parameter]

Die Liste der PAM-Dienstnamen, für die eine LDAP-Authentisierung als ausreichend gilt.

Die Vorgabe ist ‘`()`’.

12.9.19 Web-Dienste

Das Modul `(gnu services web)` stellt den Apache-HTTP-Server, den nginx-Webserver und auch einen fastcgi-Wrapperdienst bereit.

Apache-HTTP-Server

`httpd-service-type` [Scheme-Variable]

Diensttyp für den Apache-HTTP-Server (`https://httpd.apache.org/`) *httpd*. Der Wert dieses Diensttyps ist ein `httpd-configuration`-Verbund.

Es folgt ein einfaches Beispiel der Konfiguration.

```
(service httpd-service-type
         (httpd-configuration
           (config
             (httpd-config-file
               (server-name "www.example.com")
               (document-root "/srv/http/www.example.com")))))
```

Andere Dienste können den `httpd-service-type` auch erweitern, um etwas zur Konfiguration hinzuzufügen.

```
(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")))))
```

Nun folgt eine Beschreibung der Verbundstypen `httpd-configuration`, `httpd-module`, `httpd-config-file` und `httpd-virtualhost`.

`httpd-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des httpd-Dienstes.

`package` (Vorgabe: `httpd`)
: Das zu benutzende httpd-Paket.

`pid-file` (Vorgabe: `"/var/run/httpd"`)
: Die vom Shepherd-Dienst benutzte PID-Datei.

`config` (Vorgabe: `(httpd-config-file)`)
: Die vom httpd-Dienst zu benutzende Konfigurationsdatei. Vorgegeben ist ein `httpd-config-file`-Verbundsobjekt, aber als Wert kann auch ein anderer G-Ausdruck benutzt werden, der eine Datei erzeugt, zum Beispiel ein `plain-file`. Es kann auch eine Datei außerhalb des Stores mit einer Zeichenkette angegeben werden.

`httpd-module` [Datentyp]

Dieser Datentyp steht für ein Modul des httpd-Dienstes.

`name` Der Name des Moduls.

`file` Die Datei, in der das Modul steht. Sie kann relativ zum benutzten httpd-Paket oder als absoluter Pfad einer Datei oder als ein G-Ausdruck für eine Datei im Store angegeben werden, zum Beispiel `(file-append mod-wsgi "/modules/mod_wsgi.so")`.

`%default-httpd-modules` [Scheme-Variable]
Eine vorgegebene Liste von `httpd-module`-Objekten.

`httpd-config-file` [Datentyp]
Dieser Datentyp repräsentiert eine Konfigurationsdatei für den httpd-Dienst.

`modules` (Vorgabe: `%default-httpd-modules`)
Welche Module geladen werden sollen. Zusätzliche Module können hier eingetragen werden oder durch eine zusätzliche Konfigurationsangabe geladen werden.

Um zum Beispiel Anfragen nach PHP-Dateien zu behandeln, können Sie das Modul `mod_proxy_fcgi` von Apache zusammen mit `php-fpm-service-type` benutzen:

```
(service httpd-service-type
         (httpd-configuration
          (config
           (httpd-config-file
            (modules (cons*
                      (httpd-module
                       (name "proxy_module")
                       (file "modules/mod_proxy.so"))
                      (httpd-module
                       (name "proxy_fcgi_module")
                       (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` (Vorgabe: `httpd`)
Die `ServerRoot` in der Konfigurationsdatei, vorgegeben ist das httpd-Paket. Direktiven wie `Include` und `LoadModule` werden relativ zur ServerRoot interpretiert.

`server-name` (Vorgabe: `#f`)
Der `ServerName` in der Konfigurationsdatei, mit dem das Anfrageschema (Request Scheme), der Rechnername (Hostname) und Port angegeben wird, mit denen sich der Server identifiziert.

Es muss nicht als Teil der Server-Konfiguration festgelegt werden, sondern kann auch in virtuellen Rechnern (Virtual Hosts) festgelegt werden. Vorgegeben ist `#f`, wodurch kein `ServerName` festgelegt wird.

`document-root` (Vorgabe: `"/srv/http"`)
: Das `DocumentRoot`-Verzeichnis, in dem sich die Dateien befinden, die man vom Server abrufen kann.

`listen` (Vorgabe: `'("80")`)
: Die Liste der Werte für die `Listen`-Direktive in der Konfigurationsdatei. Als Wert sollte eine Liste von Zeichenketten angegeben werden, die jeweils die Portnummer, auf der gelauscht wird, und optional auch die zu benutzende IP-Adresse und das Protokoll angeben.

`pid-file` (Vorgabe: `"/var/run/httpd"`)
: Hiermit wird die PID-Datei als `PidFile`-Direktive angegeben. Der Wert sollte mit der `pid-file`-Datei in der `httpd-configuration` übereinstimmen, damit der Shepherd-Dienst richtig konfiguriert ist.

`error-log` (Vorgabe: `"/var/log/httpd/error_log"`)
: Der Ort, an den der Server mit der `ErrorLog`-Direktive Fehlerprotokolle schreibt.

`user` (Vorgabe: `"httpd"`)
: Der Benutzer, als der der Server durch die `User`-Direktive Anfragen beantwortet.

`group` (Vorgabe: `"httpd"`)
: Die Gruppe, mit der der Server durch die `Group`-Direktive Anfragen beantwortet.

`extra-config` (Vorgabe: `(list "TypesConfig etc/httpd/mime.types")`)
: Eine flache Liste von Zeichenketten und G-Ausdrücken, die am Ende der Konfigurationsdatei hinzugefügt werden.

  Alle Werte, mit denen dieser Dienst erweitert wird, werden an die Liste angehängt.

`httpd-virtualhost` [Datentyp]

Dieser Datentyp repräsentiert einen Konfigurationsblock für einen virtuellen Rechner (Virtual Host) des httpd-Dienstes.

Sie sollten zur zusätzlichen Konfiguration extra-config des httpd-Dienstes hinzugefügt werden.

```
(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")))))
```

`addresses-and-ports`
: Adressen und Ports für die `VirtualHost`-Direktive.

`contents` Der Inhalt der `VirtualHost`-Direktive. Er sollte als Liste von Zeichenketten und G-Ausdrücken angegeben werden.

NGINX

`nginx-service-type` [Scheme-Variable]

Diensttyp für den NGinx-Webserver (`https://nginx.org/`). Der Wert des Dienstes ist ein `<nginx-configuration>`-Verbundsobjekt.

Es folgt ein einfaches Beispiel der Konfiguration.

```
(service nginx-service-type
         (nginx-configuration
           (server-blocks
             (list (nginx-server-configuration
                     (server-name '("www.example.com"))
                     (root "/srv/http/www.example.com"))))))
```

Außer durch direktes Hinzufügen von Server-Blöcken zur Dienstkonfiguration kann der Dienst auch durch andere Dienste erweitert werden, um Server-Blöcke hinzuzufügen, wie man im folgenden Beispiel sieht:

```
(simple-service 'my-extra-server nginx-service-type
                (list (nginx-server-configuration
                        (root "/srv/http/extra-website")
                        (try-files (list "$uri" "$uri/index.html")))))
```

Beim Starten hat `nginx` seine Konfigurationsdatei noch nicht gelesen und benutzt eine vorgegebene Datei, um Fehlermeldungen zu protokollieren. Wenn er seine Konfigurationsdatei nicht laden kann, landen Fehlermeldungen also dort. Nachdem die Konfigurationsdatei geladen ist, werden Fehlerprotokolle nach Voreinstellung in die Datei geschrieben, die in der Konfiguration angegeben ist. In unserem Fall können Sie Fehlermeldungen beim Starten in `/var/run/nginx/logs/error.log` finden und nachdem die Konfiguration eingelesen wurde, finden Sie sie in `/var/log/nginx/error.log`. Letzterer Ort kann mit der Konfigurationsoption *log-directory* geändert werden.

`nginx-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration von NGinx. Ein Teil der Konfiguration kann hierüber und über die anderen zu Ihrer Verfügung stehenden Verbundstypen geschehen, alternativ können Sie eine Konfigurationsdatei mitgeben.

`nginx` (Vorgabe: `nginx`)
: Das zu benutzende nginx-Paket.

`shepherd-requirement` (Vorgabe: `'()`)
: Eine Liste von Symbolen, die angeben, von welchen anderen Shepherd-Diensten der nginx-Dienst abhängen soll.

  Das können Sie dann gebrauchen, wenn Sie wollen, dass `nginx` erst dann gestartet wird, wenn Web-Server im Hintergrund oder Protokollierungsdienste wie Anonip bereits gestartet wurden.

`log-directory` (Vorgabe: `"/var/log/nginx"`)
: In welches Verzeichnis NGinx Protokolldateien schreiben wird.

`run-directory` (Vorgabe: `"/var/run/nginx"`)
: In welchem Verzeichnis NGinx eine PID-Datei anlegen und temporäre Dateien ablegen wird.

`server-blocks` (Vorgabe: `'()`)
: Eine Liste von *Server-Blöcken*, die in der erzeugten Konfigurationsdatei stehen sollen. Die Elemente davon sollten den Typ `<nginx-server-configuration>` haben.

  Im folgenden Beispiel wäre NGinx so eingerichtet, dass Anfragen an `www.example.com` mit Dateien aus dem Verzeichnis `/srv/http/www.example.com` beantwortet werden, ohne HTTPS zu benutzen.

```
(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` (Vorgabe: `'()`)
: Eine Liste von *Upstream-Blöcken*, die in der erzeugten Konfigurationsdatei stehen sollen. Ihre Elemente sollten den Typ `<nginx-upstream-configuration>` haben.

  Upstreams als `upstream-blocks` zu konfigurieren, kann hilfreich sein, wenn es mit `locations` in `<nginx-server-configuration>` verbunden wird. Das folgende Beispiel erzeugt eine Server-Konfiguration mit einer Location-Konfiguration, bei der Anfragen als Proxy entsprechend einer Upstream-Konfiguration weitergeleitet werden, wodurch zwei Server diese beantworten können.

```
(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 "/path1")
                  (body '("proxy_pass http://server-proxy;"))))))))
    (upstream-blocks
      (list (nginx-upstream-configuration
              (name "server-proxy")
              (servers (list "server1.example.com"
                             "server2.example.com")))))))
```

`file` (Vorgabe: `#f`)
: Wenn eine Konfigurationsdatei als *file* angegeben wird, dann wird diese benutzt und *keine* Konfigurationsdatei anhand der angegebenen `log-directory`, `run-directory`, `server-blocks` und `upstream-blocks` er-

zeugt. Trotzdem sollten diese Argumente bei einer richtigen Konfiguration mit denen in der Datei *file* übereinstimmen, damit die Verzeichnisse bei Aktivierung des Dienstes erzeugt werden.

Das kann nützlich sein, wenn Sie schon eine bestehende Konfigurationsdatei haben oder das, was Sie brauchen, nicht mit anderen Teilen eines nginx-configuration-Verbundsobjekts umgesetzt werden kann.

`server-names-hash-bucket-size` (Vorgabe: `#f`)
: Größe der Behälter (englisch „Buckets“) für die Hashtabelle der Servernamen; vorgegeben ist `#f`, wodurch die Größe der Cache-Lines des Prozessors verwendet wird.

`server-names-hash-bucket-max-size` (Vorgabe: `#f`)
: Maximale Behältergröße für die Hashtabelle der Servernamen.

`modules` (Vorgabe: `'()`)
: Die Liste zu ladender dynamisch gebundener Module für nginx. Die dynamischen Module sollten als Liste von Dateinamen ladbarer Module angegeben werden. Zum Beispiel:

```
(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` (Vorgabe: `'()`)
: Die Liste zu ladender nginx-Lua-Pakete. Hier sollte eine Liste von Paketnamen ladbarer Lua-Module angegeben werden. Zum Beispiel:

```
(lua-package-path (list lua-resty-core
                        lua-resty-lrucache
                        lua-resty-signal
                        lua-tablepool
                        lua-resty-shell))
```

`lua-package-cpath` (Vorgabe: `'()`)
: Die Liste zu ladender nginx-Lua-C-Pakete. Hier sollte eine Liste von Paketnamen ladbarer Lua-C-Module angegeben werden. Zum Beispiel:

```
(lua-package-cpath (list lua-resty-signal))
```

`global-directives` (Vorgabe: `'((events . ()))`)
: Assoziative Liste von globalen Direktiven für die oberste Ebene der nginx-Konfiguration. Als Werte können wiederum assoziative Listen angegeben werden.

```
(global-directives
 `((worker_processes . 16)
   (pcre_jit . on)
   (events . ((worker_connections . 1024)))))
```

`extra-content` (Vorgabe: `""`)
: Zusätzlicher Inhalt des `http`-Blocks. Er sollte eine Zeichenkette oder ein zeichenkettenwertiger G-Ausdruck.

`nginx-server-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration eines nginx-Serverblocks repräsentiert. Dieser Typ hat die folgenden Parameter:

  `listen` (Vorgabe: `'("80" "443 ssl")`)
  : Jede `listen`-Direktive legt Adresse und Port für eine IP fest oder gibt einen Unix-Socket an, auf dem der Server Anfragen beantwortet. Es können entweder sowohl Adresse als auch Port oder nur die Adresse oder nur der Port angegeben werden. Als Adresse kann auch ein Rechnername („Hostname“) angegeben werden, zum Beispiel:

    ```
    '("127.0.0.1:8000" "127.0.0.1" "8000" "*:8000" "localhost:8000")
    ```

  `server-name` (Vorgabe: `(list 'default)`)
  : Eine Liste von Servernamen, die dieser Server repräsentiert. `'default` repräsentiert den voreingestellten Server, der für Verbindungen verwendet wird, die zu keinem anderen Server passen.

  `root` (Vorgabe: `"/srv/http"`)
  : Wurzelverzeichnis des Webauftritts, der über nginx abgerufen werden kann.

  `locations` (Vorgabe: `'()`)
  : Eine Liste von *nginx-location-configuration*- oder *nginx-named-location-configuration*-Verbundsobjekten, die innerhalb des Serverblocks benutzt werden.

  `index` (Vorgabe: `(list "index.html")`)
  : Index-Dateien, mit denen Anfragen nach einem Verzeichnis beantwortet werden. Wenn *keine* davon gefunden wird, antwortet Nginx mit der Liste der Dateien im Verzeichnis.

  `try-files` (Vorgabe: `'()`)
  : Eine Liste der Dateien, bei denen in der angegebenen Reihenfolge geprüft wird, ob sie existieren. `nginx` beantwortet die Anfrage mit der ersten Datei, die es findet.

  `ssl-certificate` (Vorgabe: `#f`)
  : Wo das Zertifikat für sichere Verbindungen gespeichert ist. Sie sollten es auf `#f` setzen, wenn Sie kein Zertifikat haben oder kein HTTPS benutzen möchten.

  `ssl-certificate-key` (Vorgabe: `#f`)
  : Wo der private Schlüssel für sichere Verbindungen gespeichert ist. Sie sollten ihn auf `#f` setzen, wenn Sie keinen Schlüssel haben oder kein HTTPS benutzen möchten.

  `server-tokens?` (Vorgabe: `#f`)
  : Ob der Server Informationen über seine Konfiguration bei Antworten beilegen soll.

`raw-content` (Vorgabe: `'()`)
: Eine Liste von Zeilen, die unverändert in den Serverblock eingefügt werden.

`nginx-upstream-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration eines nginx-`upstream`-Blocks repräsentiert. Dieser Typ hat folgende Parameter:

  `name` Der Name dieser Servergruppe.

  `servers` Gibt die Adressen der Server in der Gruppe an. Die Adresse kann als IP-Adresse (z.B. '`127.0.0.1`'), Domänenname (z.B. '`backend1.example.com`') oder als Pfad eines Unix-Sockets mit dem vorangestellten Präfix '`unix:`' angegeben werden. Wenn Adressen eine IP-Adresse oder einen Domänennamen benutzen, ist der voreingestellte Port 80, aber ein abweichender Port kann auch explizit angegeben werden.

  `extra-content`
  Eine Liste von Zeilen, die in den `upstream`-Block eingefügt werden.

`nginx-location-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration eines nginx-`location`-Blocks angibt. Der Typ hat die folgenden Parameter:

  `uri` Die URI, die auf diesen Block passt.

  `body` Der Rumpf des location-Blocks, der als eine Liste von Zeichenketten angegeben werden muss. Er kann viele Konfigurationsdirektiven enthalten, zum Beispiel können Anfragen an eine Upstream-Servergruppe weitergeleitet werden, die mit einem `nginx-upstream-configuration`-Block angegeben wurde, indem diese Direktive im Rumpf angegeben wird: '`(list "proxy_pass http://upstream-name;")`'.

`nginx-named-location-configuration` [Datentyp]
: Der Datentyp repräsentiert die Konfiguration eines mit Namen versehenen nginx-location-Blocks („Named Location Block“). Ein mit Namen versehener location-Block wird zur Umleitung von Anfragen benutzt und nicht für die normale Anfrageverarbeitung. Dieser Typ hat die folgenden Parameter:

  `name` Der Name, mit dem dieser location-Block identifiziert wird.

  `body` Siehe [nginx-location-configuration body], Seite 474, weil der Rumpf („Body“) eines mit Namen versehenen location-Blocks wie ein `nginx-location-configuration body` benutzt werden kann. Eine Einschränkung ist, dass der Rumpf eines mit Namen versehenen location-Blocks keine location-Blöcke enthalten kann.

Varnish Cache

Varnish ist ein schneller zwischenspeichernder Server, der zwischen Web-Anwendungen und deren Endbenutzern sitzt. Er leitet Anfragen von Clients weiter und lagert die URLs, auf die zugegriffen wird, in einen Zwischenspeicher ein, damit bei mehreren Anfragen auf dieselbe Ressource nur eine Anfrage an die Hintergrundanwendung gestellt wird.

`varnish-service-type` [Scheme-Variable]
Diensttyp für den Varnish-Daemon.

`varnish-configuration` [Datentyp]
Der Datentyp, der die Konfiguration des `varnish`-Dienstes repräsentiert. Dieser Typ hat die folgenden Parameter:

`package` (Vorgabe: `varnish`)
Das Varnish-Paket, was benutzt werden soll.

`name` (Vorgabe: `"default"`)
Ein Name für diese Varnish-Instanz. Varnish wird ein Verzeichnis in `/var/varnish/` mit diesem Namen erzeugen und dort temporäre Dateien speichern. Wenn der Name mit einem Schrägstrich beginnt, wird er als absoluter Verzeichnispfad interpretiert.

Übergeben Sie die Befehlszeilenoption `-n` an andere Varnish-Programme, um sich mit der Instanz diesen Namens zu verbinden, z.B. `varnishncsa -n default`.

`backend` (Vorgabe: `"localhost:8080"`)
Welcher Hintergrunddienst benutzt werden soll. Diese Option wird ignoriert, wenn `vcl` gesetzt ist.

`vcl` (Vorgabe: #f)
Das *VCL*-Programm (in der Varnish Configuration Language), das ausgeführt werden soll. Ist dies auf `#f` gesetzt, fungiert Varnish als Proxy für den Hintergrunddienst `backend` mit der voreingestellten Konfiguration. Andernfalls muss dies ein dateiartiges Objekt mit gültiger VCL-Syntax sein.

Um zum Beispiel mit VCL einen Spiegelserver für www.gnu.org (`https://www.gnu.org`) einzurichten, können Sie so etwas benutzen:

```
(define %gnu-mirror
  (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 %gnu-mirror)))
                  %base-services)))
```

Die Konfiguration einer bereits laufenden Varnish-Instanz kann mit dem Programm `varnishadm` eingesehen und verändert werden.

Ziehen Sie die Varnish User Guide (`https://varnish-cache.org/docs/`) und das Varnish Book (`https://book.varnish-software.com/4.0/`) zu Rate, wenn Sie eine umfassende Dokumentation zu Varnish und seiner Konfigurationssprache suchen.

`listen` (Vorgabe: `'("localhost:80")`)
: Liste der Adressen, auf denen Varnish lauschen soll.

`storage` (Vorgabe: `'("malloc,128m")`)
: Liste der Speicher-Hintergrunddienste („Storage Backends“), die von der VCL aus benutzt werden können.

`parameters` (Vorgabe: `'()`)
: Liste der Laufzeitparameter von der Form `'(("Parameter" . "Wert"))`.

`extra-options` (Vorgabe: `'()`)
: Zusätzliche Argumente, die an den `varnishd`-Prozess übergeben werden.

Patchwork

Patchwork ist ein System, um eingereichten Patches zu folgen. Es kann an eine Mailing-Liste geschickte Patches sammeln und auf einer Web-Oberfläche anzeigen.

`patchwork-service-type` [Scheme-Variable]
: Diensttyp für Patchwork.

Es folgt ein Minimalbeispiel für einen Patchwork-Dienst, der auf der Domain `patchwork.example.com` läuft.

```
(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 . ("Patches")))))))))
```

Der Patchwork-Dienst wird über drei Verbundsobjekte konfiguriert. Die `<patchwork-configuration>` hat mit der Konfiguration von Patchwork innerhalb des HTTPD-Dienstes zu tun.

Das `settings-module`-Feld innerhalb des `<patchwork-configuration>`-Verbundsobjekts kann mit einem `<patchwork-settings-module>`-Verbundsobjekt ausgefüllt werden, das ein im Guix-Store angelegtes Einstellungsmodul beschreibt.

Für das `database-configuration`-Feld innerhalb des `<patchwork-settings-module>` muss eine `<patchwork-database-configuration>` benutzt werden.

`patchwork-configuration` [Datentyp]

Der Datentyp, der die Konfiguration des Patchwork-Dienstes repräsentiert. Dieser Typ hat die folgenden Parameter:

`patchwork` (Vorgabe: `patchwork`)
: Welches Patchwork-Paket benutzt werden soll.

`domain`
: Welche Domain für Patchwork benutzt werden soll. Sie findet Verwendung in Patchworks virtuellen Rechner („Virtual Host“) für den HTTPD-Dienst.

`settings-module`
: Das durch Patchwork benutzte Einstellungsmodul. Als eine Django-Anwendung wird Patchwork mit einem Python-Modul konfiguriert, das die Einstellungen speichert. Es kann entweder eine Instanz des `<patchwork-settings-module>`-Verbundstyps sein, ein beliebiges anderes Verbundsobjekt sein, das die Einstellungen im Store repräsentiert, oder ein Verzeichnis außerhalb des Stores.

`static-path` (Vorgabe: `"/static/"`)
: Der Pfad, auf dem der HTTPD-Dienst die statischen Dateien anbieten soll.

`getmail-retriever-config`
: Das durch Patchwork benutzte getmail-retriever-configuration-Verbundsobjekt. Getmail wird mit diesem Wert konfiguriert. Die Mitteilungen werden mit Patchwork als Empfänger zugestellt.

`patchwork-settings-module` [Datentyp]

Der Datentyp, der das Einstellungsmodul für Patchwork repräsentiert. Manche dieser Einstellungen haben direkt mit Patchwork zu tun, andere beziehen sich auf Django, dem Web-Framework auf dem Patchwork aufsetzt, oder dessen Django-Rest-Framework-Bibliothek. Dieser Typ verfügt über die folgenden Parameter:

`database-configuration` (Vorgabe: `(patchwork-database-configuration)`)
: Die für Patchwork benutzten Datenbankverbindungseinstellungen. Siehe den `<patchwork-database-configuration>`-Verbundstyp für weitere Informationen.

`secret-key-file` (Vorgabe: `"/etc/patchwork/django-secret-key"`)
: Patchwork benutzt als eine Django-Webanwendung einen geheimen Schlüssel, um Werte kryptographisch zu signieren. Diese Datei sollte einen einzigartigen, unvorhersehbaren Wert enthalten.

  Wenn diese Datei nicht existiert, wird sie erzeugt und ein zufälliger Wert wird durch den Shepherd-Dienst für patchwork-setup hineingeschrieben.

  Diese Einstellung bezieht sich auf Django.

`allowed-hosts`
: Eine Liste zulässiger Netzwerkschnittstellen (Hosts), auf denen dieser Patchwork-Dienst antwortet. Sie sollte wenigstens die im `<patchwork-configuration>`-Verbundsobjekt genannte Domain enthalten.

  Dies ist eine Django-Einstellung.

`default-from-email`
: Die E-Mail-Adresse, von der aus Patchwork nach Voreinstellung E-Mails verschicken soll.

  Dies ist eine Patchwork-Einstellung.

`static-url` (Vorgabe: `#f`)
: Die URL, über die statische Dokumente angeboten werden. Es kann eine teilweise URL oder eine vollständige URL angegeben werden, aber sie muss auf `/` enden.

  Wenn der Vorgabewert benutzt wird, wird der `static-path`-Wert aus dem `<patchwork-configuration>`-Verbundsobjekt benutzt.

  Dies ist eine Django-Einstellung.

`admins` (Vorgabe: `'()`)
: Die E-Mail-Adressen, an die Details zu auftretenden Fehlern versendet werden. Jeder Wert sollte eine zweielementige Liste mit dem Namen und der E-Mail-Adresse sein.

  Dies ist eine Django-Einstellung.

`debug?` (Vorgabe: `#f`)
: Ob Patchwork im Fehlersuchmodus („Debug Mode“) ausgeführt werden soll. Wenn dies auf `#t` steht, werden detaillierte Fehlermeldungen angezeigt.

  Dies ist eine Django-Einstellung.

`enable-rest-api?` (Vorgabe: `#t`)
: Ob Patchworks REST-Programmschnittstelle („REST-API“) aktiviert werden soll.

  Dies ist eine Patchwork-Einstellung.

`enable-xmlrpc?` (Vorgabe: `#t`)
: Ob die XML-Programmschnittstelle für entfernte Prozeduraufrufe („XML-RPC-API“) aktiviert werden soll.

  Dies ist eine Patchwork-Einstellung.

`force-https-links?` (Vorgabe: `#t`)
: Ob auf den Webseiten von Patchwork die Verweise auf andere Seiten HTTPS verwenden sollen.

  Dies ist eine Patchwork-Einstellung.

`extra-settings` (Vorgabe: `""`)
: Zusätzlicher Code, der am Ende des Patchwork-Einstellungsmoduls platziert werden soll.

`patchwork-database-configuration` [Datentyp]
: Der Datentyp, der die Datenbankkonfiguration für Patchwork repräsentiert.

  `engine` (Vorgabe: `"django.db.backends.postgresql_psycopg2"`)
  : Welcher Datenbanktreiber („Engine“) benutzt werden soll.

`name` (Vorgabe: `"patchwork"`)
: Der Name der zu benutzenden Datenbank.

`user` (Vorgabe: `"httpd"`)
: Der Benutzer, als der sich Patchwork mit der Datenbank verbindet.

`password` (Vorgabe: `""`)
: Das Passwort, das zum Herstellen einer Verbindung zur Datenbank verwendet werden soll.

`host` (Vorgabe: `""`)
: Der Rechner, mit dem die Datenbankverbindung hergestellt wird.

`port` (Vorgabe: `""`)
: Der Port, auf dem sich Patchwork mit der Datenbank verbindet.

Mumi

Mumi (`https://git.elephly.net/gitweb.cgi?p=software/mumi.git`) ist eine Weboberfläche für Debbugs, einem System, um Fehlerberichte zu verwalten. Nach Vorgabe zeigt es die bei GNU angebotene Debbugs-Instanz (`https://bugs.gnu.org`). Mumi ist ein Web-Server, er lädt aber auch E-Mails von Debbugs herunter und indiziert diese.

`mumi-service-type` [Scheme-Variable]
: Dies ist der Diensttyp für Mumi.

`mumi-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration des Mumi-Dienstes repräsentiert. Dieser Typ hat die folgenden Felder:

  `mumi` (Vorgabe: `mumi`)
  : Das zu verwendende Mumi-Paket.

  `mailer?` (Vorgabe: `#true`)
  : Ob die Komponente zum Mailversand aktiviert sein soll oder nicht.

  `mumi-configuration-sender`
  : Die E-Mail-Adresse, die bei Kommentaren als Absender angegeben wird.

  `mumi-configuration-smtp`
  : Eine URI, um die SMTP-Einstellungen für Mailutils einzurichten. Hierfür könnte etwas wie `sendmail:///path/to/bin/msmtp` angegeben werden, oder eine beliebige andere URI, die von Mailutils unterstützt wird. Siehe Abschnitt “SMTP Mailboxes” in *GNU Mailutils*.

FastCGI

FastCGI ist eine Schnittstelle zwischen den Anwendungen im Vordergrund („Front-End“) und Hintergrund („Back-End“) eines Webdienstes. Die Rolle, die es ausübt, ist nicht mehr ganz aktuell, weil neue Webdienste im Allgemeinen einfach über HTTP zwischen Vorder- und Hintergrund kommunizieren sollten. Allerdings gibt es eine Menge von Hintergrunddiensten wie PHP oder den optimierten Git-Repository-Zugang über HTTP, welche FastCGI benutzen, also wird es auch in Guix unterstützt.

Um FastCGI zu benutzen, konfigurieren Sie den Webserver im Vordergrund (z.B. nginx) so, dass er eine Teilmenge der Anfragen an die fastcgi-Hintergrundanwendung weiterleitet, dass auf einem lokalen TCP- oder Unix-Socket lauscht. Ein dazwischenliegendes `fcgiwrap`-Programm sitzt zwischen dem eigentlichen Hintergrundprozess und dem Webserver. Vom Vordergrund wird angezeigt, welches Hintergrundprogramm ausgeführt werden soll. Diese Informationen werden an den `fcgiwrap`-Prozess übermittelt.

`fcgiwrap-service-type` [Scheme-Variable]
Ein Diensttyp für den `fcgiwrap`-FastCGI-Proxy.

`fcgiwrap-configuration` [Datentyp]
Der Datentyp, der die Konfiguration des `fcgiwrap`-Dienstes repräsentiert. Dieser Typ hat die folgenden Parameter:

`package` (Vorgabe: `fcgiwrap`)
Welches fcgiwrap-Paket benutzt werden soll.

`socket` (Vorgabe: `tcp:127.0.0.1:9000`)
Der Socket, auf dem der `fcgiwrap`-Prozess lauschen soll, als eine Zeichenkette. Gültige Werte für *socket* wären unter anderem `unix:`*`/pfad/zum/unix/socket`*, `tcp:`*`vier.teile.gepunkt.et`*`:`*`Port`* und `tcp6:[`*`IPv6-Adresse`*`]:Port`.

`user` (Vorgabe: `fcgiwrap`)
`group` (Vorgabe: `fcgiwrap`)
Die Benutzerkonten- und Gruppennamen als Zeichenketten, unter denen der `fcgiwrap`-Prozess ausgeführt werden soll. Der `fastcgi`-Dienst wird sicherstellen, dass, wenn der Nutzer den Benutzer- oder Gruppennamen `fcgiwrap` verlangt, der entsprechende Benutzer und/oder Gruppe auch auf dem System existiert.

Es ist möglich, einen FastCGI-gestützten Webdienst so zu konfigurieren, dass er HTTP-Authentisierungsinformationen vom Vordergrundserver an das Hintergrundsystem weiterreicht und es `fcgiwrap` möglich macht, den Hintergrundprozess als ein entsprechender lokaler Nutzer auszuführen. Um dem Hintergrundsystem diese Funktionalität anzubieten, lassen Sie `fcgiwrap` als der Administratornutzer `root` mit selbiger Gruppe ausführen. Beachten Sie, dass die Funktionalität auch auf dem Vordergrundsystem erst eingerichtet werden muss.

PHP-FPM

PHP-FPM (FastCGI Process Manager) ist eine alternative PHP-FastCGI-Implementierung, die über einige zusätzliche Funktionalitäten verfügt, die für Webauftritte jeder Größe nützlich sind.

Zu diesen Funktionalitäten gehören:

- Prozesserzeugung nach Bedarf
- Grundlegende Statistiken (ähnlich wie Apaches mod_status)
- Fortschrittliche Prozessverwaltung mit sanftem Stoppen und Starten

- Die Möglichkeit, Arbeiter-Threads mit verschiedenen UIDs, GIDs, Chroot- oder Umgebungseinstellungen zu starten und mit verschiedener php.ini (als Ersatz für safe_mode)
- Protokollierung der Standard- und Standardfehlerausgabe
- Neustart im Notfall einer ungewollten Zerstörung des Befehlscode-Zwischenspeichers
- Unterstützung für beschleunigtes Hochladen
- Unterstützung für „langsames Protokollieren“ („slowlog“)
- Verbesserungen gegenüber FastCGI, wie z.B. fastcgi_finish_request() – eine besondere Funktion, um eine Anfrage fertig abzuarbeiten und alle Daten zu Ende zu verarbeiten, während etwas Zeitintensives abläuft (Videokonvertierung, Statistikverarbeitung usw.)

... und vieles mehr.

`php-fpm-service-type` [Scheme-Variable]
Ein Diensttyp für `php-fpm`.

`php-fpm-configuration` [Datentyp]
Datentyp für die Konfiguration des php-fpm-Dienstes.

`php` (Vorgabe: `php`)
Das zu benutzende PHP-Paket.

`socket` (Vorgabe: `(string-append "/var/run/php" (version-major (package-version php)) "-fpm.sock")`)
Die Adresse, auf der FastCGI-Anfragen angenommen werden. Gültige Syntax hierfür ist:

`"ip.ad.res.se:Port"`
Lässt auf einem TCP-Socket auf der angegebenen Adresse auf dem angegebenen Port lauschen.

`"port"` Lässt auf einem TCP-Socket auf allen Adressen auf dem angegebenen Port lauschen.

`"/pfad/zum/unix/socket"`
Lässt auf einem Unix-Socket lauschen.

`user` (Vorgabe: `php-fpm`)
Der Benutzer, dem die PHP-Arbeiterprozesse gehören werden.

`group` (Vorgabe: `php-fpm`)
Die Gruppe für die Arbeiterprozesse.

`socket-user` (Vorgabe: `php-fpm`)
Der Benutzer, der mit dem php-fpm-Socket kommunizieren kann.

`socket-group` (Vorgabe: `nginx`)
Die Gruppe, die mit dem php-fpm-Socket kommunizieren kann.

`pid-file` (Vorgabe: `(string-append "/var/run/php" (version-major (package-version php)) "-fpm.pid")`)
Der Prozessidentifikator des php-fpm-Prozesses wird in diese Datei geschrieben, sobald der Dienst gestartet wurde.

`log-file` (Vorgabe: `(string-append "/var/log/php" (version-major (package-version php)) "-fpm.log")`)
: Wohin das Protokoll für den php-fpm-Hauptprozess geschrieben wird.

`process-manager` (Vorgabe: `(php-fpm-dynamic-process-manager-configuration)`)
: Detaillierte Einstellungen für die php-fpm-Prozessverwaltung. Sie müssen eines der Folgenden sein:

```
<php-fpm-dynamic-process-manager-configuration>
<php-fpm-static-process-manager-configuration>
<php-fpm-on-demand-process-manager-configuration>
```

`display-errors` (Vorgabe: `#f`)
: Legt fest, ob PHP-Fehler und Warnungen an Clients geschickt und in ihren Browsern angezeigt werden. Dies ist nützlich für lokale PHP-Entwicklung, aber ein Sicherheitsrisiko für öffentliche Webauftritte, weil Fehlermeldungen Passwörter und Passwortdaten offenlegen können.

`timezone` (Vorgabe: `#f`)
: Legt den Parameter `php_admin_value[date.timezone]` fest.

`workers-logfile` (Vorgabe: `(string-append "/var/log/php" (version-major (package-version php)) "-fpm.www.log")`)
: In dieser Datei werden `stderr`-Ausgaben von PHP-Arbeiterprozessen protokolliert. Das Feld kann auf `#f` gesetzt werden, damit nicht protokolliert wird.

`file` (Vorgabe: `#f`)
: Optional kann hier ein vorrangig benutzter Ersatz für die gesamte Konfigurationsdatei angegeben werden. Sie können dafür die `mixed-text-file`-Funktion oder einen absoluten Dateipfad verwenden.

`php-ini-file` (Vorgabe: `#f`)
: Hier können optional die PHP-Voreinstellungen geändert werden. Geben Sie dazu eine beliebige Art von „dateiartigem“ Objekt an (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175). Sie können dafür die `mixed-text-file`-Funktion oder einen absoluten Dateipfad verwenden.

  Bei der Entwicklung mit dem lokalen Rechner bietet es sich an, die Zeit- und Speicherbegrenzung erzeugter PHP-Prozesse anzuheben. Benutzen Sie dazu folgndes Code-Schnipsel in Ihrer Betriebssystemkonfiguration:

```
(define %local-php-ini
  (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 %local-php-ini)))
```

```
                                %base-services)))
```

Umfassende Dokumentation dazu, welche Direktiven Sie in der `php.ini` verwenden dürfen, sollten Sie unter den php.ini-Kerndirektiven (`https://www.php.net/manual/en/ini.core.php`) nachschlagen.

`php-fpm-dynamic-process-manager-configuration` [Datentyp]
Datentyp für die *dynamische* Prozessverwaltung durch php-fpm. Bei der *dynamischen* Prozessverwaltung bleiben Arbeiterprozesse nach Abschluss ihrer Aufgabe weiterhin erhalten, solange die konfigurierten Beschränkungen eingehalten werden.

`max-children` (Vorgabe: `5`)
Die maximale Anzahl an Arbeiterprozessen.

`start-servers` (Vorgabe: `2`)
Wie viele Arbeiterprozesse gleich zu Beginn gestartet werden sollen.

`min-spare-servers` (Vorgabe: `1`)
Wie viele untätige Arbeiterprozesse mindestens weiterhin vorgehalten bleiben sollen.

`max-spare-servers` (Vorgabe: `3`)
Wie viele untätige Arbeiterprozesse höchstens weiterhin vorgehalten bleiben sollen.

`php-fpm-static-process-manager-configuration` [Datentyp]
Datentyp für die *statische* Prozessverwaltung durch php-fpm. Bei der *statischen* Prozessverwaltung wird eine unveränderliche Anzahl an Arbeiterprozessen erzeugt.

`max-children` (Vorgabe: `5`)
Die maximale Anzahl an Arbeiterprozessen.

`php-fpm-on-demand-process-manager-configuration` [Datentyp]
Datentyp für die Prozessverwaltung *nach Bedarf* durch php-fpm. Bei der Prozessverwaltung *nach Bedarf* werden Arbeiterprozesse erst erzeugt, wenn Anfragen vorliegen.

`max-children` (Vorgabe: `5`)
Die maximale Anzahl an Arbeiterprozessen.

`process-idle-timeout` (Vorgabe: `10`)
Die Zeit in Sekunden, nach der ein Prozess ohne Anfragen abgewürgt wird.

`nginx-php-location` [*#:nginx-package nginx*] [*socket* (*string-append "/var/run/php" (version-major (package-version php)) "-fpm.sock")*]* [Scheme-Prozedur]
Eine Hilfsfunktion, mit der in kurzer Zeit PHP zu einer `nginx-server-configuration` hinzugefügt werden kann.

Eine einfache Art, die Dienste für nginx mit PHP einzurichten, kann so aussehen:

```
(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))
```

Der Cat Avatar Generator („Katzenavatargenerator“) ist ein einfacher Dienst, um die Nutzung von php-fpm in `Nginx` zu demonstrieren. Mit ihm können Katzenavatarbilder aus einem Startwert („Seed“) heraus erzeugt werden, zum Beispiel aus dem Hash der E-Mail-Adresse eines Benutzers.

`cat-avatar-generator-service` [*#:cache-dir* "*/var/cache/cat-avatar-generator*"] [*#:package* cat-avatar-generator] [#:configuration (nginx-server-configuration)] [Scheme-Prozedur]

Liefert ein nginx-server-configuration-Objekt, das von der in `configuration` angegebenen Konfiguration erbt. Es erweitert die Nginx-Konfiguration, indem es einen Server-Block hinzufügt, der die in `package` angegebene Version vom cat-avatar-generator anbietet. Bei der Ausführung wird dem cat-avatar-generator Zugriff auf sein in `cache-dir` angegebenes Zwischenspeicherverzeichnis gewährt.

Eine einfache Konfiguration des cat-avatar-generator kann so aussehen:

```
(services (cons* (cat-avatar-generator-service
                  #:configuration
                  (nginx-server-configuration
                    (server-name '("example.com"))))
                 ...
                 %base-services))
```

Hpcguix-web

Das Programm hpcguix-web (`https://github.com/UMCUGenetics/hpcguix-web/`) ist eine anpassbare Weboberfläche, um Guix-Pakete zu suchen. Am Anfang war es für Nutzer von Hochleistungs-Rechenclustern gedacht („High-Performance Computing“, kurz HPC).

`hpcguix-web-service-type` [Scheme-Variable]

Der Diensttyp für `hpcguix-web`.

`hpcguix-web-configuration` [Datentyp]

Datentyp für die Konfiguration des hpcguix-web-Dienstes.

`specs` Ein G-Ausdruck (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), der die Konfiguration des hpcguix-web-Dienstes festlegt. In dieser Spezifikation („Spec“) sollten vor allem diese Sachen angegeben werden:

`title-prefix` (Vorgabe: `"hpcguix | "`)
Das Präfix der Webseitentitel.

`guix-command` (Vorgabe: `"guix"`)
: Der `guix`-Befehl.

`package-filter-proc` (Vorgabe: `(const #t)`)
: Eine Prozedur, die festlegt, wie anzuzeigende Pakete gefiltert werden.

`package-page-extension-proc` (Vorgabe: `(const '())`)
: Erweiterungspaket für `hpcguix-web`.

`menu` (Vorgabe: `'()`)
: Zusätzlicher Eintrag auf der Menüseite.

`channels` (Vorgabe: `%default-channels`)
: Liste der Kanäle, aus denen die Paketliste erstellt wird (siehe Kapitel 7 [Kanäle], Seite 77).

`package-list-expiration` (Vorgabe: `(* 12 3600)`)
: Nach wie viel Zeit in Sekunden die Paketliste aus den neuesten Instanzen der angegebenen Kanäle neu erzeugt wird.

Siehe das Repository von hpcguix-web für ein vollständiges Beispiel (`https://github.com/UMCUGenetics/hpcguix-web/blob/master/hpcweb-configuration.scm`).

`package` (Vorgabe: `hpcguix-web`)
: Das hpcguix-web-Paket, was benutzt werden soll.

`address` (Vorgabe: `"127.0.0.1"`)
: Die IP-Adresse, auf die gelauscht werden soll.

`port` (Vorgabe: `5000`)
: Die Portnummer, auf der gelauscht werden soll.

Eine typische Deklaration eines hpcguix-web-Dienstes sieht so aus:

```
(service hpcguix-web-service-type
         (hpcguix-web-configuration
          (specs
           #~(define site-config
               (hpcweb-configuration
                (title-prefix "Guix-HPC - ")
                (menu '(("/about" "ABOUT"))))))))
```

Anmerkung: Der hpcguix-web-Dienst aktualisiert die Liste der Pakete, die er veröffentlicht, periodisch, indem er die Kanäle über einen Git-„Pull“ lädt. Dazu muss er auf X.509-Zertifikate zugreifen, damit Git-Server authentifiziert werden können, wenn mit diesen über HTTPS kommuniziert wird, wofür der Dienst davon ausgeht, dass sich jene Zertifikate in `/etc/ssl/certs` befinden.

Stellen Sie also sicher, dass `nss-certs` oder ein anderes Zertifikatspaket im `packages`-Feld ihrer Konfiguration steht. Siehe Abschnitt 12.11 [X.509-Zertifikate], Seite 606, für weitere Informationen zu X.509-Zertifikaten.

gmnisrv

Das Programm gmnisrv (`https://git.sr.ht/~sircmpwn/gmnisrv`) ist ein einfacher Server für das Gemini-Protokoll (`https://gemini.circumlunar.space/`).

`gmnisrv-service-type` [Scheme-Variable]

Dies ist der Diensttyp des gmnisrv-Dienstes, dessen Wert ein `gmnisrv-configuration`-Objekt wie in diesem Beispiel sein sollte:

```
(service gmnisrv-service-type
         (gmnisrv-configuration
           (config-file (local-file "./my-gmnisrv.ini"))))
```

`gmnisrv-configuration` [Datentyp]

Datentyp, der die Konfiguration von gmnisrv repräsentiert.

`package` (Vorgabe: *gmnisrv*)
: Das Paketobjekt des gmnisrv-Servers.

`config-file` (Vorgabe: `%default-gmnisrv-config-file`)
: Ein dateiartiges Objekt mit der zu nutzenden gmnisrv-Konfigurationsdatei. Die Vorgabekonfiguration lauscht auf Port 1965 und bietet die Dateien in `/srv/gemini` an. Zertifikate werden in `/var/lib/gemini/certs` gespeichert. Führen Sie für mehr Informationen `man gmnisrv` und `man gmnisrv.ini` aus.

Agate

Das Programm Agate (`gemini://qwertqwefsday.eu/agate.gmi`) (GitHub-Seite über HTTPS (`https://github.com/mbrubeck/agate`)) ist ein einfacher Server für das Gemini-Protokoll (`https://gemini.circumlunar.space/`), der in Rust geschrieben ist.

`agate-service-type` [Scheme-Variable]

Dies ist der Diensttyp des agate-Dienstes, dessen Wert ein `agate-configuration`-Objekt wie in diesem Beispiel sein sollte:

```
(service agate-service-type
         (agate-configuration
           (content "/srv/gemini")
           (cert "/srv/cert.pem")
           (key "/srv/key.rsa")))
```

Das obige Beispiel steht für die minimalen Abänderungen, um Agate lauffähig zu machen. Den Pfad zum Zertifikat und zum Schlüssel anzugeben, ist auf jeden Fall nötig, denn das Gemini-Protokoll setzt standardmäßig TLS voraus.

Um ein Zertifikat und einen Schlüssel zu bekommen, können Sie zum Beispiel OpenSSL benutzen. Das ginge mit einem Befehl ähnlich zu diesem Beispiel hier:

```
openssl req -x509 -newkey rsa:4096 -keyout key.rsa -out cert.pem \
    -days 3650 -nodes -subj "/CN=beispiel.com"
```

Natürlich müssen Sie statt *beispiel.com* Ihren eigenen Domainnamen angeben und dann die Konfiguration von Agate auf den Pfad des damit erzeugten Schlüssels und Zertifikats verweisen lassen.

`agate-configuration` [Datentyp]

Der Datentyp, der die Konfiguration von Agate repräsentiert.

`package` (Vorgabe: `agate`)
: Das Paketobjekt des Agate-Servers.

`content` (Vorgabe: `"/srv/gemini"`)
: Aus welchem Verzeichnis Agate Dateien anbieten soll.

`cert` (Vorgabe: `#f`)
: Der Pfad zur PEM-Datei mit dem TLS-Zertifikat, das verschlüsselte Verbindungen ermöglicht. Hier muss ein Wert angegeben werden.

`key` (Vorgabe: `#f`)
: Dateipfad zur Datei mit dem privaten PKCS8-Schlüssel, der für verschlüsselte Verbindungen benutzt wird. Hier muss ein Wert angegeben werden.

`addr` (Vorgabe: `'("0.0.0.0:1965" "[::]:1965")`)
: Liste der Adressen, auf denen gelauscht werden soll.

`hostname` (Vorgabe: `#f`)
: Der Domainname dieses Gemini-Servers. Optional.

`lang` (Vorgabe: `#f`)
: RFC-4646-Sprachcode(s) für Dokumente des Typs text/gemini. Optional.

`silent?` (Vorgabe: `#f`)
: Setzen Sie dies auf `#t`, um Protokollausgaben auszuschalten.

`serve-secret?` (Vorgabe: `#f`)
: Setzen Sie dies auf `#t`, um geheime Dateien (mit einem Punkt beginnende Dateien und Verzeichnisse) anzubieten.

`log-ip?` (Vorgabe: `#t`)
: Ob in die Protokolle auch IP-Adressen geschrieben werden sollen.

`user` (Vorgabe: `"agate"`)
: Besitzer des `agate`-Prozesses.

`group` (Vorgabe: `"agate"`)
: Gruppe des Besitzers des `agate`-Prozesses.

`log-file` (Vorgabe: `"/var/log/agate.log"`)
: Die Datei, in die die Protokollierung von Agate ausgegeben werden soll.

12.9.20 Zertifikatsdienste

Das Modul `(gnu services certbot)` stellt einen Dienst zur Verfügung, um automatisch ein gültiges TLS-Zertifikat von der Zertifikatsautorität Let's Encrypt zu beziehen. Mit diesen Zertifikaten können Informationen sicher über HTTPS oder andere TLS-basierte Protokolle übertragen werden, im Wissen, dass der Client die Authentizität des Servers überprüfen wird können.

Let's Encrypt (`https://letsencrypt.org/`) macht das `certbot`-Werkzeug verfügbar, mit dem der Zertifizierungsvorgang automatisiert werden kann. Das Werkzeug erzeugt

zunächst auf sichere Weise einen Schlüssel auf dem Server und stellt dann eine Anfrage an die Let's-Encrypt-Zertifikatsautorität („Certificate Authority“, kurz CA), den Schlüssel zu signieren. Die Zertifikatsautorität prüft mit einem Challenge-Response-Protokoll, dass die Anfrage auch wirklich vom fraglichen Rechner (auch „Host“ genannt) kommt, wozu der Server über HTTP seine Antwort geben muss. Wenn dieses Protokoll erfolgreich befolgt wurde, signiert die Zertifikatsautorität den Schlüssel, woraus sich ein Zertifikat ergibt. Dieses Zertifikat ist eine begrenzte Zeit lang gültig, daher muss der Server für eine andauernde Bereitstellung von TLS-Leistungen immer wieder neu der Zertifikatsautorität eine Bitte um die Erneuerung der Signatur zukommen lassen.

Mit dem certbot-Dienst wird dieser Prozess automatisiert. Er sorgt dafür, dass am Anfang Schlüssel erzeugt werden und eine erste Zertifizierungsanfrage an den Dienst von Let's Encrypt gestellt wird. Weiterhin ist das Challenge-/Response-Verfahren per Web-Server integriert. Das Zertifikat wird auf die Platte geschrieben und automatisch periodisch erneuert und bei der Erneuerung anfallende Aufgaben werden erledigt (z.B. das Neuladen von Diensten und das Kopieren von Schlüsseln mit entsprechenden Berechtigungen).

Certbot wird zweimal täglich zu einer zufälligen Minute der Stunde ausgeführt. Es tut so lange nichts, bis eine Erneuerung Ihrer Zertifikate fällig wird oder sie gesperrt wurden, durch regelmäßige Ausführung bekommen Sie aber die Chance, dass Ihr Server am Netz bleibt, wenn Let's Encrypt eine Sperrung aus irgendeinem Grund anordnet.

Durch die Nutzung dieses Dienstes stimmen Sie dem „ACME Subscriber Agreement“ zu, das Sie hier finden:: `https://acme-v01.api.letsencrypt.org/directory`.

`certbot-service-type` [Scheme-Variable]

Ein Diensttyp für den `certbot`-Client für Let's Encrypt. Sein Wert muss ein `certbot-configuration`-Verbundsobjekt wie in diesem Beispiel sein:

```
(define %nginx-deploy-hook
  (program-file
   "nginx-deploy-hook"
   #~(let ((pid (call-with-input-file "/var/run/nginx/pid" read)))
       (kill pid SIGHUP))))

(service certbot-service-type
         (certbot-configuration
          (email "foo@example.net")
          (certificates
           (list
            (certificate-configuration
             (domains '("example.net" "www.example.net"))
             (deploy-hook %nginx-deploy-hook))
            (certificate-configuration
             (domains '("bar.example.net")))))))
```

Siehe unten für Details zur `certbot-configuration`.

`certbot-configuration` [Datentyp]

Datentyp, der die Konfiguration des `certbot`-Dienstes repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`package` (Vorgabe: `certbot`)
: Das certbot-Paket, das benutzt werden soll.

`webroot` (Vorgabe: `/var/www`)
: Das Verzeichnis, aus dem heraus die Dateien für den Challenge-/Response-Prozess von Let's Encrypt angeboten werden sollen.

`certificates` (Vorgabe: `()`)
: Eine Liste der `certificates-configuration`-Objekte, für die Zertifikate und Anfragesignaturen erzeugt werden. Für jedes Zertifikat gibt es einen `name`-Eintrag und mehrere `domains`.

`email` (Vorgabe: `#f`)
: Optional die E-Mail-Adresse, die als Kontakt für die Registrierung und das Wiedererlangen dienen soll. Es wird empfohlen, sie anzugeben, weil Sie darüber wichtige Mitteilungen über Ihr Konto und ausgestellte Zertifikate mitbekommen können.

`server` (Vorgabe: `#f`)
: Optional eine andere URL des ACME-Servers. Wenn sie festgelegt wird, ersetzt sie die Voreinstellung von Certbot, nämlich die URL des Let's-Encrypt-Servers.

`rsa-key-size` (Vorgabe: `2048`)
: Wie groß der RSA-Schlüssel sein soll.

`default-location` (Vorgabe: *siehe unten*)
: Die vorgegebene `nginx-location-configuration`. Weil `certbot` „Challenges“ und „Responses“ anbieten muss, muss durch ihn ein Web-Server ausgeführt werden können. Das tut er, indem er den `nginx`-Webdienst mit einer `nginx-server-configuration` erweitert, die auf den *domains* auf Port 80 lauscht und eine `nginx-location-configuration` für den URI-Pfad-Teilraum `/.well-known/` umfasst, der von Let's Encrypt benutzt wird. Siehe Abschnitt 12.9.19 [Web-Dienste], Seite 466, für mehr Informationen über diese nginx-Konfigurationsdatentypen.

  Anfragen an andere URL-Pfade werden mit der `default-location` abgeglichen. Wenn sie angegeben wurde, wird sie zu jeder `nginx-server-configuration` hinzugefügt.

  Nach Vorgabe stellt die `default-location` eine Weiterleitung von `http://`*`domain`*`/...` nach `https://`*`domain`*`/...` her. Sie müssen dann nur noch festlegen, was Sie auf Ihrem Webauftritt über `https` anbieten wollen.

  Übergeben Sie `#f`, um keine `default-location` vorzugeben.

`certificate-configuration` [Datentyp]
Der Datentyp, der die Konfiguration eines Zertifikats repräsentiert. Dieser Typ hat die folgenden Parameter:

`name` (Vorgabe: *siehe unten*)
: Dieser Name wird vom Certbot intern zum Aufräumen und in Dateipfaden benutzt; er hat keinen Einfluss auf den Inhalt des erzeugten Zertifi-

kats. Um Zertifikatsnamen einzusehen, führen Sie `certbot certificates` aus.

Die Vorgabe ist die erste angegebene Domain.

`domains` (Vorgabe: `()`)
: Die erste angegebene Domain wird als Name des Zertifikatseigentümers („Subject CN“) benutzt und alle Domains werden als alternative Namen („Subject Alternative Names“) auf dem Zertifikat stehen.

`challenge` (Vorgabe: `#f`)
: Welche Art von Challenge durch den Certbot ausgeführt wird. Wenn `#f` angegeben wird, wird die HTTP-Challenge voreingestellt. Wenn ein Wert angegeben wird, wird das Plugin benutzt, das auch bei manuellen Ausführungen benutzt wird (siehe `authentication-hook`, `cleanup-hook` und die Dokumentation unter `https://certbot.eff.org/docs/using.html#hooks`), und Let's Encrypt wird dazu berechtigt, die öffentliche IP-Adresse der anfordernden Maschine in seinem Protokoll zu speichern.

`csr` (Vorgabe: `#f`)
: Der Dateiname der Zertifizierungsanfrage („Certificate Signing Request“, CSR) im DER- oder PEM-Format. Wenn `#f` angegeben wird, wird kein solches Argument an den Certbot übermittelt. Wenn ein Wert angegeben wird, wird der Certbot ihn anstelle einer selbsterstellten CSR verwenden. Die in `domains` genannten Domänen müssen dabei konsistent sein mit denen, die in der CSR-Datei aufgeführt werden.

`authentication-hook` (Vorgabe: `#f`)
: Welcher Befehl in einer Shell zum Antworten auf eine Zertifikats-„Challenge“ einmalig ausgeführt wird. Für diesen Befehl wird die Shell-Variable `$CERTBOT_DOMAIN` die Domain enthalten, für die sich der Certbot authentisiert, `$CERTBOT_VALIDATION` enthält die Validierungs-Zeichenkette und `$CERTBOT_TOKEN` enthält den Dateinamen der bei einer HTTP-01-Challenge angefragten Ressource.

`cleanup-hook` (Vorgabe: `#f`)
: Welcher Befehl in einer Shell für jede Zertifikat-„Challenge“ einmalig ausgeführt wird, die vom `auth-hook` beantwortet wurde. Für diesen Befehl bleiben die Shell-Variablen weiterhin verfügbar, die im `auth-hook`-Skript zur Verfügung standen, und außerdem wird `$CERTBOT_AUTH_OUTPUT` die Standardausgabe des `auth-hook`-Skripts enthalten.

`deploy-hook` (Vorgabe: `#f`)
: Welcher Befehl in einer Shell für jedes erfolgreich ausgestellte Zertifikat einmalig ausgeführt wird. Bei diesem Befehl wird die Shell-Variable `$RENEWED_LINEAGE` auf das Unterverzeichnis für die aktuelle Konfiguration zeigen (zum Beispiel ‘`"/etc/letsencrypt/live/example.com"`’), in dem sich die neuen Zertifikate und Schlüssel befinden. Die Shell-Variable `$RENEWED_DOMAINS` wird eine leerzeichengetrennte Liste der erneuerten Zertifikatsdomänen enthalten (zum Beispiel ‘`"example.com www.example.com"`’.

Für jede `certificate-configuration` wird das Zertifikat in `/etc/letsencrypt/live/`*name*`/fullchain.pe` und der Schlüssel in `/etc/letsencrypt/live/`*name*`/privkey.pem` gespeichert.

12.9.21 DNS-Dienste

Das Modul `(gnu services dns)` stellt Dienste zur Verfügung, die mit dem *Domain Name System* (DNS) zu tun haben. Es bietet einen Server-Dienst an, mit dem ein *autoritativer* DNS-Server für mehrere Zonen betrieben werden kann, jeweils als untergeordneter „Slave“ oder als „Master“. Dieser Dienst benutzt Knot DNS (`https://www.knot-dns.cz/`). Außerdem wird ein zwischenspeichernder und weiterleitender DNS-Server für das LAN bereitgestellt, der dnsmasq (`http://www.thekelleys.org.uk/dnsmasq/doc.html`) benutzt.

Knot-Dienst

Eine Beispielkonfiguration eines autoritativen Servers für zwei Zonen, eine „Master“, eine „Slave“, wäre:

```
(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` [Scheme-Variable]

Dies ist der Diensttyp für den Knot-DNS-Server.

Knot DNS ist ein autoritativer DNS-Server, das heißt, er kann mehrere Zonen bedienen, also mehrere Domainnamen, die Sie von einem Registrar kaufen würden. Dieser Server ist kein „Resolver“, er dient also nur zur Auflösung von Namen, für die er autoritativ ist. Dieser Server kann so konfiguriert werden, dass er Zonen als „Master“-Server oder als „Slave“-Server bereitstellt, je nachdem, wie er für die jeweilige Zone eingestellt ist. Server für „Slave“-Zonen erhalten ihre Daten von „Master“-Servern und stellen mit ihnen einen autoritativen Server zur Verfügung. Für einen „Resolver“ macht es keinen Unterschied, ob er Namen auflöst, indem er einen „Master“ oder einen „Slave“ danach fragt.

Die folgenden Datentypen werden benutzt, um den Knot-DNS-Server zu konfigurieren:

`knot-key-configuration` [Datentyp]

Datentyp, der einen Schlüssel repräsentiert. Dieser Typ hat die folgenden Parameter:

`id` (Vorgabe: `""`)
: Ein Identifikator, mit dem sich andere Konfigurationsfelder auf diesen Schlüssel beziehen können. IDs müssen eindeutig sein und dürfen *nicht* leer sein.

`algorithm` (Vorgabe: `#f`)
: Der Algorithmus, der benutzt werden soll. Wählen Sie zwischen `#f`, `'hmac-md5`, `'hmac-sha1`, `'hmac-sha224`, `'hmac-sha256`, `'hmac-sha384` und `'hmac-sha512`.

`secret` (Vorgabe: `""`)
: Was dabei der geheime Schlüssel sein soll.

`knot-acl-configuration` [Datentyp]

Datentyp, der die Konfiguration einer Zugriffssteuerungsliste („Access Control List“, ACL) repräsentiert. Dieser Typ hat die folgenden Parameter:

`id` (Vorgabe: `""`)
: Ein Identifikator, mit dem sich andere Konfigurationsfelder auf diesen Schlüssel beziehen können. IDs müssen eindeutig sein und dürfen *nicht* leer sein.

`address` (Vorgabe: `'()`)
: Eine geordnete Liste aus IP-Adresse, Netzwerk-Subnetzen oder Netzwerkbereichen, die jeweils als Zeichenketten angegeben werden. Die Anfrage muss zu einem davon passen. Ein leerer Wert bedeutet, dass die Adresse nicht passen muss.

`key` (Vorgabe: `'()`)
: Eine geordnete Liste von Referenzen auf Schlüssel, die jeweils als Zeichenketten angegeben werden. Die Zeichenkette muss zu einem Schlüsselidentifikator passen, der in einem der `knot-key-configuration`-Objekte definiert wurde. Wenn kein Schlüssel angegeben

wird, bedeutet das, dass kein Schlüssel zu dieser Zugriffssteuerungsliste passen muss.

`action` (Vorgabe: `'()`)
: Eine geordete Liste der Aktionen, die von dieser Zugriffssteuerungsliste zugelassen oder gesperrt werden. Mögliche Werte sind Listen aus null oder mehr Elementen, die jeweils `'transfer`, `'notify` oder `'update` sind.

`deny?` (Vorgabe: `#f`)
: Wenn dies auf wahr steht, werden mit der Zugriffssteuerungsliste Einschränkungen festgelegt, d.h. aufgelistet Aktionen werden gesperrt. Steht es auf falsch, werden aufgelistete Aktionen zugelassen.

`zone-entry` [Datentyp]

Datentyp, der einen Eintrag in einer Zonendatei repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`name` (Vorgabe: `"@"`)
: Der Name des Eintrags. `"@"` bezieht sich auf den Ursprung der Zone. Namen sind relativ zum Ursprung der Zone. Zum Beispiel bezieht sich in einer Zone `example.org` der Eintrag `"ns.example.org"` tatsächlich auf `ns.example.org.example.org`. Namen, die auf einen Punkt enden, sind absolut. Das bedeutet, dass sich `"ns.example.org."` auf `ns.example.org` bezieht.

`ttl` (Vorgabe: `""`)
: Wie lange dieser Eintrag zwischengespeichert werden darf, d.h. seine „Time-To-Live“ (TTL). Ist sie nicht festgelegt, wird die voreingestellte TTL benutzt.

`class` (Vorgabe: `"IN"`)
: Welche Klasse der Eintrag hat. Derzeit unterstützt Knot nur `"IN"` und teilweise `"CH"`.

`type` (Vorgabe: `"A"`)
: Der Typ des Eintrags. Zu den üblichen Typen gehören A (für eine IPv4-Adresse), AAAA (für eine IPv6-Adresse), NS (der Namens-Server) und MX („Mail eXchange“ für E-Mails). Viele andere Typen sind auch definiert.

`data` (Vorgabe: `""`)
: Die Daten, die im Eintrag stehen, zum Beispiel eine IP-Adresse bei einem A-Eintrag oder ein Domain-Name bei einem NS-Eintrag. Bedenken Sie, dass Domain-Namen relativ zum Ursprung angegeben werden, außer wenn sie auf einen Punkt enden.

`zone-file` [Datentyp]

Datentyp, der den Inhalt einer Zonendatei repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`entries` (Vorgabe: `'()`)
: Die Liste der Einträge. Für den SOA-Eintrag wird automatisch gesorgt, also müssen Sie ihn nicht zur Liste der Einträge hinzufügen. In der Lis-

te sollte vermutlich ein Eintrag für Ihren primären autoritativen DNS-Server stehen. Abgesehen vom direkten Aufzählen der Einträge können Sie `define-zone-entries` verwenden, um ein Objekt zu definieren, worin eine Liste von Einträgen leichter angegeben werden kann, und was sie dann im `entries`-Feld des `zone-file` angeben können.

`origin` (Vorgabe: `""`)
: Der Name Ihrer Zone. Dieser Parameter darf nicht leer sein.

`ns` (Vorgabe: `"ns"`)
: Die Domain Ihres primären autoritativen DNS-Servers. Der Name wird relativ zum Ursprung angegeben, außer wenn er auf einen Punkt endet. Dieser primäre DNS-Server *muss* verpflichtend einem NS-Eintrag in der Zone entsprechen, dem eine IP-Adresse in der Liste der Einträge zugeordnet werden muss.

`mail` (Vorgabe: `"hostmaster"`)
: Eine E-Mail-Adresse, unter der man Sie als für diese Zone Verantwortlichen („Besitzer“/„Owner“) kontaktieren kann. Sie wird zu `<mail>@<origin>` umgeschrieben.

`serial` (Vorgabe: `1`)
: Die Seriennummer der Zone. Da sie von sowohl „Slaves“ als auch „Resolvern“ benutzt wird, um bei Änderungen auf dem Laufenden zu bleiben, ist es notwendig, dass sie *niemals* kleiner gemacht wird. Erhöhen Sie sie, wann immer Sie eine Änderung an Ihrer Zone durchführen.

`refresh` (Vorgabe: `(* 2 24 3600)`)
: Die Häufigkeit, wie oft Slaves eine Zonenübertragung („Zone Transfer“) durchführen. Als Wert wird eine Anzahl von Sekunden angegeben. Sie kann über eine Multiplikation oder mit `(string->duration)` angegeben werden.

`retry` (Vorgabe: `(* 15 60)`)
: Nach welcher Zeitperiode ein Slave versuchen wird, Kontakt mit seinem Master aufzunehmen, wenn er ihn beim ersten Mal nicht erreichen kann.

`expiry` (Vorgabe: `(* 14 24 3600)`)
: Die Voreinstellung, welche TTL für Einträge verwendet werden soll. Bestehende Einträge werden für höchstens diese Zeitspanne als korrekt angesehen. Nach Ablauf dieser Zeitspanne werden „Resolver“ ihren Zwischenspeicher als ungültig markieren und erneut prüfen, ob der Eintrag noch existiert.

`nx` (Vorgabe: `3600`)
: Die voreingestellte TTL der *nicht* existierenden Einträge. Sie stellt normalerweise eine kurze Verzögerung dar, weil Sie möchten, dass neue Domains für jeden schnell erkannt werden.

`knot-remote-configuration` [Datentyp]
Datentyp, der die Konfiguration eines entfernten Servers („Remote“) repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`id` (Vorgabe: `""`)
: Ein Identifikator, mit dem man sich in anderen Konfigurationsfeldern auf diesen entfernten Server („Remote“) beziehen kann. IDs müssen eindeutig sein und dürfen *nicht* leer sein.

`address` (Vorgabe: `'()`)
: Eine geordnete Liste der Empfänger-IP-Adressen. Adressen werden der Reihe nach durchprobiert. Optional kann eine Portnummer nach dem Trennzeichen @ angegeben werden, zum Beispiel als `(list "1.2.3.4" "2.3.4.5@53")`. Die Vorgabe ist 53.

`via` (Vorgabe: `'()`)
: Eine geordnete Liste der Quell-IP-Adressen. Eine leere Liste wird Knot eine sinnvolle Quell-IP-Adresse auswählen lassen. Optional kann eine Portnummer nach dem Trennzeichen @ angegeben werden. Die Vorgabe wird zufällig ausgewählt.

`key` (Vorgabe: `#f`)
: Ein Verweis auf einen Schlüssel („Key“), also eine Zeichenkette, die den Identifikator eines Schlüssels enthält, der in einem `knot-key-configuration`-Feld festgelegt wurde.

`knot-keystore-configuration` [Datentyp]

Datentyp, der einen Schlüsselspeicher („Keystore“) repräsentiert, um DNSSEC-Schlüssel zu fassen. Dieser Typ verfügt über die folgenden Parameter:

`id` (Vorgabe: `""`)
: Der Identifikator des Schlüsselspeichers. Er darf nicht leer gelassen werden.

`backend` (Vorgabe: `'pem`)
: Die Art von Hintergrundspeicher, in dem Schlüssel eingetragen werden. Sie kann `'pem` oder `'pkcs11` sein.

`config` (Vorgabe: `"/var/lib/knot/keys/keys"`)
: Die Zeichenkette mit der Konfiguration des Hintergrundspeichers. Ein Beispiel für die PKCS#11 ist `"pkcs11:token=knot;pin-value=1234 /gnu/store/.../lib/pkcs11/libsofthsm2.so"`. Für pem als Hintergrundspeicher repräsentiert die Zeichenkette einen Pfad im Dateisystem.

`knot-policy-configuration` [Datentyp]

Datentyp, der die DNSSEC-Richtlinie repräsentiert. Knot DNS kann Ihre Zonen automatisch signieren. Der Dienst kann Ihre Schlüssel automatisch erzeugen und verwalten oder Schlüssel benutzen, die Sie selbst erzeugen.

DNSSEC wird in der Regel mit zwei Schlüsseln implementiert: Ein Schlüssel, mit dem Schlüssel signiert werden („Key Signing Key“, KSK), signiert den zweiten Schlüssel, einen Schlüssel, der Zonen signiert („Zone Signing Key“, ZSK), mit dem die Zone signiert wird. Damit er als vertrauenswürdig angesehen wird, muss der KSK in der Elternzone stehen (meistens ist das eine Top-Level-Domain). Wenn Ihr Registrar DNSSEC unterstützt, müssen Sie ihm den Hash Ihres KSK übermitteln, damit er

einen DS-Eintrag für Ihre Zone hinzufügen kann. Das passiert nicht automatisch und muss jedes Mal wiederholt werden, wenn Sie Ihren KSK ändern.

Die Richtlinie legt auch fest, wie lange Ihre Schlüssel gültig bleiben. Normalerweise kann der ZSK leicht geändert werden und benutzt kryptografisch schwächere Funktionen (also niedrigere Parameter), damit Einträge schnell signiert werden können, wodurch man sie oft verändern kann. Der KSK setzt jedoch eine manuelle Interaktion mit dem Registrar voraus, also werden sie weniger oft geändert und verwenden stärkere Parameter, weil mit ihnen nur ein einziger Eintrag signiert wird.

Dieser Typ verfügt über die folgenden Parameter:

`id` (Vorgabe: `""`)
: Der Identifikator der Richtlinie. Er darf nicht leer sein.

`keystore` (Vorgabe: `"default"`)
: Eine Referenz auf einen Schlüsselspeicher („Keystore“), also eine Zeichenkette, die den Identifikator eines Schlüsselspeichers enthält, der in einem `knot-keystore-configuration`-Feld gespeichert ist. Der Identifikator `"default"` sorgt dafür, dass der vorgegebene Schlüsselspeicher verwendet wird (eine KASP-Datenbank, die durch diesen Dienst eingerichtet wurde).

`manual?` (Vorgabe: `#f`)
: Ob Schlüssel manuell verwaltet werden sollen; andernfalls werden sie automatisch verwaltet.

`single-type-signing?` (Vorgabe: `#f`)
: Wenn es auf `#t` steht, werden dieselben Schlüssel als KSK und ZSK verwendet („Single-Type Signing Scheme“).

`algorithm` (Vorgabe: `"ecdsap256sha256"`)
: Ein Algorithmus für zum Signieren verwendete Schlüssel und ausgestellte Signaturen.

`ksk-size` (Vorgabe: `256`)
: Die Länge des KSK. Beachten Sie, dass dieser Wert für den vorgegebenen Algorithmus korrekt ist, aber für andere Algorithmen *nicht* sicher wäre.

`zsk-size` (Vorgabe: `256`)
: Die Länge des ZSK. Beachten Sie, dass dieser Wert für den vorgegebenen Algorithmus korrekt ist, aber für andere Algorithmen *nicht* sicher wäre.

`dnskey-ttl` (Vorgabe: `'default`)
: Der TTL-Wert für DNSKEY-Einträge, die die Wurzel der Zone betreffen. Der besondere Wert `'default` bedeutet, dass dieselbe TTL wie für den SOA-Eintrag der Zone verwendet wird.

`zsk-lifetime` (Vorgabe: `(* 30 24 3600)`)
: Die Zeitspanne zwischen der Veröffentlichung eines ZSK und dem Anfang des nächsten Schlüsselübergangs („Key Rollover“).

`propagation-delay` (Vorgabe: `(* 24 3600)`)
: Eine zusätzliche Verlängerung, die bei jedem Schritt im Schlüsselübergang („Key Rollover“) gewartet wird. Dieser Wert sollte hoch genug sein, damit in dieser Zeit Daten vom Master-Server alle Slaves erreichen.

`rrsig-lifetime` (Vorgabe: `(* 14 24 3600)`)
: Wie lange neu ausgestellte Signaturen gültig bleiben.

`rrsig-refresh` (Vorgabe: `(* 7 24 3600)`)
: Wie lange im Voraus vor einem Auslaufen der Signatur diese Signatur erneuert werden soll.

`nsec3?` (Vorgabe: `#f`)
: Ist es auf `#t` gesetzt, wird NSEC3 statt NSEC benutzt.

`nsec3-iterations` (Vorgabe: `5`)
: Wie oft zusätzlich gehasht werden soll.

`nsec3-salt-length` (Vorgabe: `8`)
: Wie lange das kryptografische „Salt“ sein soll, als Anzahl von Oktetten. Es wird vor dem Hashen an den Namen des ursprünglichen Besitzers angehängt.

`nsec3-salt-lifetime` (Vorgabe: `(* 30 24 3600)`)
: Wie lange das neu ausgestellte Salt-Feld gültig bleiben soll.

`knot-zone-configuration` [Datentyp]
: Datentyp, der eine durch Knot verfügbar gemachte Zone repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

  `domain` (Vorgabe: `""`)
  : Die Domain, die durch diese Konfiguration zur Verfügung gestellt wird. Sie darf nicht leer sein.

  `file` (Vorgabe: `""`)
  : Die Datei, in der diese Zone abgespeichert wird. Dieser Parameter wird bei Master-Zonen ignoriert. Bleibt er leer, wird die vom Domain-Namen abhängige Voreinstellung benutzt.

  `zone` (Vorgabe: `(zone-file)`)
  : Der Inhalt der Zonendatei. Dieser Parameter wird bei Slave-Zonen ignoriert. Er muss ein Verbundsobjekt vom Typ `zone-file` enthalten.

  `master` (Vorgabe: `'()`)
  : Eine Liste von als „Master“ geltenden entfernten Servern. Ist sie leer, ist diese Zone ein Master, sonst ein Slave. Es handelt sich um eine Liste von Identifikatoren entfernter Server („Remotes“).

  `ddns-master` (Vorgabe: `#f`)
  : Der vorrangige „Master“. Ist dies leer, wird hierfür der erste Master aus der Liste der Master benutzt.

  `notify` (Vorgabe: `'()`)
  : Eine Liste der Identifikatoren von entfernten Slave-Servern („Remotes“).

`acl` (Vorgabe: `'()`)
: Eine Liste von Identifikatoren von Zugriffssteuerungslisten.

`semantic-checks?` (Vorgabe: `#f`)
: Wenn dies festgelegt ist, werden für die Zone mehr semantische Überprüfungen durchgeführt.

`zonefile-sync` (Vorgabe: `0`)
: Wie lange nach einer Änderung der im Arbeitsspeicher zwischengespeicherten Daten gewartet wird, bis die Daten auf die Platte geschrieben werden. Bei 0 werden sie sofort synchronisiert.

`zonefile-load` (Vorgabe: `#f`)
: Wie die in der Zonendatei gespeicherten Daten benutzt werden, wenn die Zone geladen wird. Mögliche Werte sind:

  - `#f` sorgt dafür, dass nach der Voreinstellung von Knot verfahren wird,
  - `'none` bewirkt, dass die Zonendatei überhaupt nicht benutzt wird,
  - `'difference` lässt den Unterschied zwischen den bereits vorliegenden Daten und dem gespeicherten Inhalt der Zone berechnen, welcher dann zum vorliegenden Zoneninhalt hinzugenommen wird,
  - `'difference-no-serial` für dasselbe wie bei `'difference`, aber die SOA-Seriennummer in der Zonendatei wird ignoriert und der Server kümmert sich automatisch darum.
  - `'whole` lässt den ganzen Inhalt der Zone aus der Zonendatei auslesen.

`journal-content` (Vorgabe: `#f`)
: Wie in den Aufzeichnungen die Zone und Änderungen daran gespeichert werden sollen. Mögliche Werte sind `'none`, um keine Aufzeichnungen zu führen, `'changes`, um Änderungen zu speichern, und `'all`, wodurch der gesamte Inhalt gespeichert wird. Für `#f` wird dieser Wert nicht gesetzt, so dass der in Knot voreingestellte Wert benutzt wird.

`max-journal-usage` (Vorgabe: `#f`)
: Die maximale Größe, die Aufzeichnungen für die Wiederherstellbarkeit („Journal“) auf der Platte einnehmen können. Für `#f` wird dieser Wert nicht gesetzt, so dass der in Knot voreingestellte Wert benutzt wird.

`max-journal-depth` (Vorgabe: `#f`)
: Wie viele Aufzeichnungen höchstens im Änderungsverlauf gespeichert werden. Für `#f` wird dieser Wert nicht gesetzt, so dass der in Knot voreingestellte Wert benutzt wird.

`max-zone-size` (Vorgabe: `#f`)
: Die maximale Größe der Zonendatei. Diese Beschränkung wird auf eingehende Übertragungen und Aktualisierungen angewandt. Für `#f` wird dieser Wert nicht gesetzt, so dass der in Knot voreingestellte Wert benutzt wird.

`dnssec-policy` (Vorgabe: `#f`)
: Ein Verweis auf ein `knot-policy-configuration`-Verbundsobjekt oder der besondere Name `"default"`, um die Voreinstellung von Knot zu ver-

wenden. Wenn dies als der Wert `#f` angegeben wurde, findet in dieser Zone kein Signieren mit DNSSEC statt.

`serial-policy` (Vorgabe: `'increment`)
: Eine Richtlinie; entweder `'increment` (Seriennummer hochzählen) oder `'unixtime` (Unix-Zeitstempel verwenden).

`knot-configuration` [Datentyp]

Datentyp, der die Konfiguration von Knot repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`knot` (Vorgabe: `knot`)
: Das Knot-Paket.

`run-directory` (Vorgabe: `"/var/run/knot"`)
: Das Laufzeit-Verzeichnis („run“-Verzeichnis). In diesem Verzeichnis werden die PID-Datei mit dem Prozessidentifikator und Sockets gespeichert.

`includes` (Vorgabe: `'()`)
: Eine flache Liste von Zeichenketten oder dateiartigen Objekten, die oben in der Konfigurationsdatei eingebunden werden müssen.

  Hiermit können Geheimnisse abseits von Guix' Zuständigkeitsbereich gespeichert werden. Zum Beispiel können Sie geheime Schlüssel so in einer externen Datei speichern, die nicht von Guix verwaltet und daher auch nicht von jedem in `/gnu/store` ausgelesen werden kann – Sie können etwa Ihre geheime Schlüsselkonfiguration in `/etc/knot/secrets.conf` speichern und diese Datei dann zu Ihrer `includes`-Liste hinzufügen.

  Sie können mit dem Schlüsselverwaltungsprogramm `keymgr` aus dem Knot-Paket einen geheimen TSIG-Schlüssel erzeugen lassen (für `nsupdate` und Zonentransfers). Beachten Sie, dass das Paket *nicht* automatisch durch den Dienst installiert wird. Das folgende Beispiel zeigt, wie man einen neuen TSIG-Schlüssel erzeugen lässt:

  ```
  keymgr -t meingeheimnis > /etc/knot/secrets.conf
  chmod 600 /etc/knot/secrets.conf
  ```

  Außerdem sollten Sie bedenken, dass der erzeugte Schlüssel den Namen *meingeheimnis* bekommt, dieser Name also auch im *key*-Feld des `knot-acl-configuration`-Verbundsobjekts und an anderen Stellen verwendet werden muss, wo auf den Schlüssel verwiesen wird.

  Sie können die `includes` auch benutzen, um von der Guix-Schnittstelle nicht unterstützte Einstellungen festzulegen.

`listen-v4` (Vorgabe: `"0.0.0.0"`)
: Eine IP-Adresse, auf der gelauscht werden soll.

`listen-v6` (Vorgabe: `"::"`)
: Eine IP-Adresse, auf der gelauscht werden soll.

`listen-port` (Vorgabe: `53`)
: Ein Port, auf dem gelauscht werden soll.

`keys` (Vorgabe: `'()`)
: Die Liste der `knot-key-configuration`-Objekte, die von dieser Konfiguration benutzt werden sollen.

`acls` (Vorgabe: `'()`)
: Die Liste der `knot-acl-configuration`-Objekte, die von dieser Konfiguration benutzt werden sollen.

`remotes` (Vorgabe: `'()`)
: Die Liste der `knot-remote-configuration`-Objekte, die von dieser Konfiguration benutzt werden sollen.

`zones` (Vorgabe: `'()`)
: Die Liste der `knot-zone-configuration`-Objekte, die von dieser Konfiguration benutzt werden sollen.

Knot-Resolver-Dienst

`knot-resolver-service-type` [Scheme-Variable]

Dies ist der Diensttyp des Knot-Resolver-Dienstes, dessen Wert ein `knot-resolver-configuration`-Objekt wie in diesem Beispiel sein sollte:

```
(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
"))))
```

Weitere Informationen finden Sie in seinem Handbuch (`https://knot-resolver.readthedocs.org/en/stable/daemon.html#configuration`).

`knot-resolver-configuration` [Datentyp]

Der Datentyp, der die Konfiguration von knot-resolver repräsentiert.

`package` (Vorgabe: *knot-resolver*)
: Das Paketobjekt des Knot-DNS-Resolvers.

`kresd-config-file` (Vorgabe: %kresd.conf)
: Dateiartiges Objekt der zu nutzenden kresd-Konfigurationsdatei. Nach Vorgabe lauscht der Knot-Resolver auf `127.0.0.1` und `::1`.

`garbage-collection-interval` (Vorgabe: 1000)
: Wie viele Millisekunden `kres-cache-gc` zwischen Bereinigungen seines Zwischenspeichers wartet.

Dnsmasq-Dienst

`dnsmasq-service-type` [Scheme-Variable]

Dies ist der Diensttyp des dnsmasq-Dienstes, dessen Wert ein `dnsmasq-configuration`-Objekt wie in diesem Beispiel sein sollte:

```
(service dnsmasq-service-type
```

```
(dnsmasq-configuration
  (no-resolv? #t)
  (servers '("192.168.1.1"))))
```

`dnsmasq-configuration` [Datentyp]

Repräsentiert die dnsmasq-Konfiguration.

`package` (Vorgabe: *dnsmasq*)
: Paketobjekt des dnsmasq-Servers.

`no-hosts?` (Vorgabe: `#f`)
: Ist es auf wahr gesetzt, werden keine Rechnernamen („Hostnames“) aus /etc/hosts ausgelesen.

`port` (Vorgabe: `53`)
: Der Port, auf dem gelauscht werden soll. Wird dies auf null gesetzt, werden keinerlei DNS-Anfragen beantwortet und es bleiben nur DHCP- und/oder TFTP-Funktionen.

`local-service?` (Vorgabe: `#t`)
: DNS-Anfragen nur von Rechnern akzeptieren, deren Adresse auf einem lokalen Subnetz liegt, d.h. einem Subnetz, für dem auf dem Server eine Schnittstelle existiert.

`listen-addresses` (Vorgabe: `'()`)
: Lässt auf den angegebenen IP-Adressen lauschen.

`resolv-file` (Vorgabe: `"/etc/resolv.conf"`)
: Aus welcher Datei die IP-Adresse der zu verwendenden Namensserver gelesen werden sollen.

`no-resolv?` (Vorgabe: `#f`)
: Wenn es auf wahr steht, wird das *resolv-file* nicht gelesen.

`forward-private-reverse-lookup?` (Vorgabe: `#t`)
: Wenn dies auf falsch gesetzt ist, werden inverse Namensauflösungen in privaten IP-Adressbereichen wie „Domain existiert nicht“ beantwortet, anstatt sie an vorgelagerte Server weiterzuleiten.

`query-servers-in-order?` (Vorgabe: `#f`)
: Wenn dies auf wahr gesetzt ist, wird dnsmasq die Server in genau der Reihenfolge anfragen, in der sie in *servers* aufgeführt sind.

`servers` (Vorgabe: `'()`)
: Geben Sie die IP-Adresse von anzufragenden Servern direkt an.

`addresses` (Vorgabe: `'()`)
: Geben Sie in jedem Eintrag eine IP-Adresse an, die für jeden Rechner mit einer der angegebenen Domains zurückgeliefert werden soll. Anfragen nach den Domains werden niemals weitergegeben, sondern werden immer mit der festgelegten IP-Adresse beantwortet.

  Dies ist nützlich, um für Rechnernamen lokale Umleitungen einzurichten, wie in diesem Beispiel:

```
(service dnsmasq-service-type
```

```
(dnsmasq-configuration
  (addresses
    '(; Weiterleitung auf lokalen Webserver.
      "/example.org/127.0.0.1"
      ; Weiterleitung einer Subdomain auf eine bestimmte IP.
      "/subdomain.example.org/192.168.1.42"))))
```

Beachten Sie, dass die Regeln in `/etc/hosts` Vorrang haben.

`cache-size` (Vorgabe: `150`)
: Bestimmt die Größe des Zwischenspeichers von dnsmasq. Wird die Zwischenspeichergröße auf null festgelegt, wird kein Zwischenspeicher benutzt.

`negative-cache?` (Vorgabe: `#t`)
: Ist dies auf falsch gesetzt, werden Negativergebnisse nicht zwischengespeichert.

`cpe-id` (Vorgabe: `#f`)
: Wenn es festgelegt ist, wird dies als Endgerätebezeichnung („Customer-Premises Equipment Identifier“) in DNS-Anfragen übermittelt, die an vorgelagerte Server weitergeleitet werden.

`tftp-enable?` (Vorgabe: `#f`)
: Ob der eingebaute TFTP-Server aktiviert werden soll.

`tftp-no-fail?` (Vorgabe: `#f`)
: Wenn dies wahr ist und der TFTP-Server nicht gestartet werden kann, gilt dnsmasq trotzdem *nicht* als fehlgeschlagen.

`tftp-single-port?` (Vorgabe: `#f`)
: Ob nur ein einzelner Port für TFTP benutzt werden soll.

`tftp-secure?` (Vorgabe: `#f`)
: Wenn dies wahr ist, kann nur auf solche Dateien zugegriffen werden, die dem Benutzerkonto gehören, das den dnsmasq-Prozess ausführt.

  Wird dnsmasq durch den Administratornutzer root ausgeführt, gelten andere Regeln: `tftp-secure?` wirkt sich nicht aus, aber es kann nur auf Dateien zugegriffen werden, auf die jeder Benutzer zugreifen darf (das „world-readable bit“ ist gesetzt).

`tftp-max` (Vorgabe: `#f`)
: Wenn dies gesetzt ist, gibt es die Maximalzahl gleichzeitig zugelassener Verbindungen an.

`tftp-mtu` (Vorgabe: `#f`)
: Wenn es gesetzt ist, gibt es die MTU für TFTP-Pakete an, also die Maximalgröße für Netzwerkschnittstellen.

`tftp-no-blocksize?` (Vorgabe: `#f`)
: Steht es auf wahr, wird der TFTP-Server die Blockgröße *nicht* mit dem Client aushandeln.

`tftp-lowercase?` (Vorgabe: `#f`)
: Ob alle Dateinamen in TFTP-Anfragen in Kleinbuchstaben umgesetzt werden sollen.

`tftp-port-range` (Vorgabe: `#f`)
: Wenn es gesetzt ist, wird jeder dynamische Port (einer pro Client) aus dem angegebenen Bereich (`"<von>,<bis>"`) genommen.

`tftp-root` (Vorgabe: `/var/empty,lo`)
: Relativ zu welchem Verzeichnis die Dateien für die Übertragung per TFTP gesucht werden. Wenn dies gesetzt ist, werden TFTP-Pfade, die '`..`' enthalten, abgelehnt, damit Clients keinen Zugriff auf Dateien außerhalb des angegebenen Wurzelverzeichnisses haben. Absolute Pfade (solche, die mit '`/`' beginnen) sind erlaubt, aber sie müssen in der TFTP-Root liegen. Wenn das optionale Argument `interface` angegeben wird, wird das Verzeichnis nur für TFTP-Anfragen über diese Schnittstelle benutzt.

`tftp-unique-root` (Vorgabe: `#f`)
: Wenn es gesetzt ist, wird entsprechend die IP- oder Hardware-Adresse des TFTP-Clients als eine Pfadkomponente ans Ende der TFTP-Root angehängt. Das ist nur gültig, wenn eine TFTP-Root gesetzt ist und das Verzeichnis existiert. Die Voreinstellung ist, die IP-Adresse anzuhängen (wie üblich als punktgetrenntes Quadrupel).

  Ist zum Beispiel `tftp-root` '`/tftp`' und fragt Client '`1.2.3.4`' die Datei `meinedatei` an, dann wird effektiv als Pfad `/tftp/1.2.3.4/meinedatei` abgerufen, wenn `/tftp/1.2.3.4` existiert, oder `/tftp/meinedatei` andernfalls. Wird '`=mac`' angegeben, wird stattdessen die MAC-Adresse angehängt (in Kleinbuchstaben und durch Striche getrennt, aufgefüllt mit der Ziffer null), z.B. '`01-02-03-04-aa-bb`'. Beachten Sie, dass MAC-Adressen nur aufgelöst werden können, wenn sich der Client im lokalen Netzwerk befindet oder von dnsmasq eine Adresse per DHCP zugewiesen bekommen hat.

ddclient-Dienst

Der im Folgenden beschriebene ddclient-Dienst führt den ddclient-Daemon aus, der dafür sorgt, dass DNS-Einträge für Dienstanbieter wie Dyn (`https://dyn.com/dns/`) automatisch aktualisiert werden.

Das folgende Beispiel zeigt, wie man den Dienst mit seiner Vorgabekonfiguration instanziiert:

```
(service ddclient-service-type)
```

Beachten Sie, dass der ddclient auf Zugangsdaten zugreifen muss, die in einer *Geheimnisdatei* („Secret File“) stehen; nach Vorgabe wird sie in `/etc/ddclient/secrets` gesucht (siehe `secret-file` unten). Es wird erwartet, dass Sie diese Datei manuell erstellen, ohne Guix dafür zu benutzen (theoretisch *könnten* Sie die Datei zu einem Teil Ihrer Dienstkonfiguration machen, indem Sie z.B. `plain-file` benutzen, aber dann könnte jeder über `/gnu/store` ihren Inhalt einsehen). Siehe die Beispiele im Verzeichnis `share/ddclient` des `ddclient`-Pakets.

Verfügbare `ddclient-configuration`-Felder sind:

`„package“ ddclient` [`ddclient-configuration`-Parameter]
Das ddclient-Paket.

`Ganze-Zahl daemon` [`ddclient-configuration`-Parameter]
Nach wie viel Zeit ddclient erneut versuchen wird, IP und Domain-Namen zu überprüfen.

Die Vorgabe ist ‘`300`’.

`Boolescher-Ausdruck syslog` [`ddclient-configuration`-Parameter]
Ob die Ausgabe an Syslog gehen soll.

Die Vorgabe ist ‘`#t`’.

`Zeichenkette mail` [`ddclient-configuration`-Parameter]
An welchen Benutzer Mitteilungen gemailt werden sollen.

Die Vorgabe ist ‘`"root"`’.

`Zeichenkette mail-failure` [`ddclient-configuration`-Parameter]
Den Nutzer per Mail bei fehlgeschlagenen Aktualisierungen benachrichtigen.

Die Vorgabe ist ‘`"root"`’.

`Zeichenkette pid` [`ddclient-configuration`-Parameter]
PID-Datei für den ddclient.

Die Vorgabe ist ‘`"/var/run/ddclient/ddclient.pid"`’.

`Boolescher-Ausdruck ssl` [`ddclient-configuration`-Parameter]
SSL-Unterstützung aktivieren.

Die Vorgabe ist ‘`#t`’.

`Zeichenkette user` [`ddclient-configuration`-Parameter]
Gibt den Namen oder Identifikator des Benutzerkontos an, unter dem das ddclient-Programm laufen soll.

Die Vorgabe ist ‘`"ddclient"`’.

`Zeichenkette group` [`ddclient-configuration`-Parameter]
Die Gruppe des Benutzers, mit dem das ddclient-Programm läuft.

Die Vorgabe ist ‘`"ddclient"`’.

`Zeichenkette secret-file` [`ddclient-configuration`-Parameter]
Die Geheimnisdatei („Secret File“), die an die erzeugte `ddclient.conf`-Datei angehängt wird. Diese Datei enthält die Zugangsdaten, die ddclient benutzen soll. Es wird erwartet, dass Sie sie manuell erzeugen.

Die Vorgabe ist ‘`"/etc/ddclient/secrets.conf"`’.

`Liste extra-options` [`ddclient-configuration`-Parameter]
Zusätzliche Einstellungsoptionen, die an die `ddclient.conf`-Datei angehängt werden.

Die Vorgabe ist ‘`()`’.

12.9.22 VNC-Dienste

Im Modul `(gnu services vnc)` werden Dienste angeboten, die mit *Virtual Network Computing* (VNC) zu tun haben, einer Technik, um grafische Xorg-Anwendungen, die auf einer entfernten Maschine laufen, lokal zu benutzen. Zusammen mit einer grafischen Anzeigenverwaltung, die das *X Display Manager Control Protocol* unterstützt, wie etwa GDM (siehe [gdm], Seite 342) oder LightDM (siehe [lightdm], Seite 347), kann man somit eine ganze entfernte Desktop-Arbeitsumgebung auf den eigenen Rechner bringen, um eine Mehrbenutzerumgebung herzustellen.

Xvnc

Xvnc ist ein VNC-Server, der einen eigenen X-Fensterserver startet. So lässt er sich auf Servern ohne Bildschirm („headless server“) betreiben. Die Xvnc-Implementierungen, die durch den `tigervnc-server` und `turbovnc` bereitgestellt werden, sind auf Schnelligkeit und Effizienz ausgerichtet.

`Scheme-Variable` *xvnc-service-type* [Variable]

Ein Dienst des Diensttyps `xvnc-server-type` kann eingerichtet werden mit einem `xvnc-configuration`-Verbundsobjekt, wie es nun beschrieben wird. Über die folgende Konfiguration würde eine zweite, virtuelle Anzeige auf einer entfernten Maschine verfügbar gemacht:

```
(service xvnc-service-type
         (xvnc-configuration (display-number 10)))
```

Zu Demonstrationszwecken könnte man dann den Befehl `xclock` auf der entfernten Maschine auf Anzeige Nummer 10 starten und ihn über den Befehl `vncviewer` lokal anzeigen:

```
# xclock auf der entfernten Maschine starten.
ssh -L5910:localhost:5910 -- guix shell xclock -- env DISPLAY=:10 xclock
# Über VNC darauf zugreifen.
guix shell tigervnc-client -- vncviewer localhost:5910
```

Die folgende Konfiguration setzt XDMCP und Inetd gemeinsam ein, damit mehrere Benutzer gleichzeitig das entfernte System nutzen können und sich grafisch über die GDM-Anzeigenverwaltung anmelden können:

```
(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)))))))
```

Eine entfernte Nutzerin könnte sich damit verbinden, indem sie den Befehl `vncviewer` oder einen kompatiblen VNC-Client einsetzt und eine Sitzung auf einer Arbeitsumgebung ihrer Wahl beginnt:

```
vncviewer name-des-entfernten-rechners:5905
```

Warnung: Sofern sich Ihre Maschine *nicht* in einer kontrollierten Umgebung befindet, ist es aus Sicherheitsgründen ratsam, in der Konfiguration das Feld `localhost?` des `xvnc-configuration`-Verbundsobjekts beim vorgegebenen Wert `#t` zu belassen und den entfernten Rechner nur über sichere Mittel wie eine SSH-Portweiterleitung zugänglich zu machen. Der XDMCP-Port, UDP 177, sollte nach außen hin über eine Firewall gesperrt sein, denn VNC ist *kein* sicheres Protokoll und Anmeldedaten könnten im Klartext verschickt werden.

`xvnc-configuration` [Datentyp]

Verfügbare `xvnc-configuration`-Felder sind:

`xvnc` (Vorgabe: `tigervnc-server`) (Typ: dateiartig)
: Das Paket, das die Binärdatei Xvnc zur Verfügung stellt.

`display-number` (Vorgabe: `0`) (Typ: Zahl)
: Die Nummer der Anzeige, um die sich Xvnc kümmert. Sie sollten eine Zahl wählen, die der Xorg-Server *nicht* bereits in Beschlag nimmt.

`geometry` (Vorgabe: `"1024x768"`) (Typ: Zeichenkette)
: Die Ausmaße der Anzeige, die angelegt wird.

`depth` (Vorgabe: `24`) (Typ: Farbtiefe)
: Die Pixeltiefe in Bits der anzulegenden Anzeige. Akzeptiert werden die Werte 16, 24 oder 32.

`port` (Typ: Vielleicht-Port)
: Auf welchem Port auf Verbindungen durch VNC-Betrachter gelauscht wird. Wenn keiner angegeben wird, ist 5900 plus der Anzeigennummer vorgegeben.

`ipv4?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Für eingehende und ausgehende Verbindungen IPv4 benutzen.

`ipv6?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Für eingehende und ausgehende Verbindungen IPv6 benutzen.

`password-file` (Typ: Vielleicht-Zeichenkette)
: Welche Passwortdatei benutzt werden soll, wenn gewünscht. Schauen Sie sich vncpasswd(1) an, um zu erfahren, wie Sie so eine Datei erzeugen lassen.

`xdmcp?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Stellt an den XDMCP-Server eine Sitzungsanfrage. Dadurch können sich Benutzer an der Oberfläche der Anmeldeverwaltung für eine Desktop-Sitzung anmelden. In einem Szenario mit mehreren Nutzern wird man außerdem die Option `inetd?` aktivieren wollen, damit jede Verbindung zum VNC-Server getrennt behandelt wird, statt sie zusammenzulegen.

`inetd?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Einen Dienst im Inetd-Stil benutzen. Dadurch wird der Xvnc-Server bei Bedarf gestartet.

`frame-rate` (Vorgabe: `60`) (Typ: Zahl)
: Wie viele Aktualisierungen höchstens pro Sekunde an jeden Client geschickt werden.

`security-types` (Vorgabe: `("None")`) (Typ: Sicherheitstypen)
: Welche Sicherheitsschemata für eingehende Verbindungen zugelassen sind. Vorgegeben ist "None" (keine), was sicher ist, weil Xvnc so eingestellt ist, dass sich der Benutzer mit der Anzeigenverwaltung anmeldet und nur lokale Verbindungen stattfinden. Als Wert akzeptiert wird eine Teilmenge hiervon: ("None" "VncAuth" "Plain" "TLSNone" "TLSVnc" "TLSPlain" "X509None" "X509Vnc")

`localhost?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Nur Verbindungen von derselben Maschine zulassen. Nach Vorgabe ist dies auf wahr gesetzt (`#true`), weil das sicherer ist, denn man sollte ohnehin nur Ports für sichere Verbindungsmethoden wie SSH freigeben.

`log-level` (Vorgabe: `30`) (Typ: Protokollstufe)
: Die Protokollstufe als Zahl zwischen 0 und 100. Dabei bedeutet 100 die ausführlichste Ausgabe. Protokollnachrichten werden an Syslog ausgegeben.

`extra-options` (Vorgabe: `()`) (Typ: Zeichenketten)
: Hiermit können zusätzliche Xvnc-Optionen angegeben werden, die über das `<xvnc-configuration>`-Verbundsobjekt anderweitig unzugänglich sind.

12.9.23 VPN-Dienste

Das Modul `(gnu services vpn)` stellt Dienste zur Verfügung, die mit *Virtual Private Networks* (VPNs) zu tun haben.

Bitmask

`bitmask-service-type` [Scheme-Variable]
: Ein Diensttyp für den VPN-Client Bitmask (`https://bitmask.net`). Damit wird der Client systemweit verfügbar gemacht und dessen Polkit-Richtlinie geladen. Beachten Sie, dass der Client nur mit einem laufenden polkit-agent funktioniert, der entweder von Ihrer „Desktop“-Arbeitsumgebung oder von Ihnen manuell gestartet wird.

OpenVPN

Hiermit wird ein *Client*-Dienst angeboten, mit dem sich Ihre Maschine mit einem VPN verbinden kann, sowie ein *Server*-Dienst, mit dem Sie auf Ihrer Maschine ein VPN betreiben können.

`openvpn-client-service` [*#:config* (*openvpn-client-configuration*)] [Scheme-Prozedur]
: Liefert einen Dienst, der den VPN-Daemon `openvpn` als Client ausführt.

`openvpn-server-service` [*#:config* (*openvpn-server-configuration*)] [Scheme-Prozedur]

Liefert einen Dienst, der den VPN-Daemon `openvpn` als Server ausführt.

Beide können zeitgleich laufen gelassen werden.

`openvpn-client-configuration` [Datentyp]

Verfügbare `openvpn-client-configuration`-Felder sind:

`openvpn` (Vorgabe: `openvpn`) (Typ: dateiartig)
Das OpenVPN-Paket.

`pid-file` (Vorgabe: `"/var/run/openvpn/openvpn.pid"`) (Typ: Zeichenkette)
Die Datei für den Prozessidentifikator („PID“) von OpenVPN.

`proto` (Vorgabe: `udp`) (Typ: Protokoll)
Das Protokoll (UDP oder TCP), das benutzt werden soll, um einen Kommunikationskanal zwischen Clients und Servern herzustellen.

`dev` (Vorgabe: `tun`) (Typ: Gerätetyp)
Der Gerätetyp, mit dem die VPN-Verbindung repräsentiert werden soll.

`ca` (Vorgabe: `"/etc/openvpn/ca.crt"`) (Typ: Vielleicht-Zeichenkette)
Die Zertifikatsautorität, mit der Verbindungen geprüft werden.

`cert` (Vorgabe: `"/etc/openvpn/client.crt"`) (Typ: Vielleicht-Zeichenkette)
Das Zertifikat der Maschine, auf der der Daemon läuft. Es sollte von der in `ca` angegebenen Zertifikatsautorität signiert sein.

`key` (Vorgabe: `"/etc/openvpn/client.key"`) (Typ: Vielleicht-Zeichenkette)
Der Schlüssel der Maschine, auf der der Daemon läuft. Er muss der Schlüssel zum in `cert` angegebenen Zertifikat sein.

`comp-lzo?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
Ob der Kompressionsalgorithmus lzo benutzt werden soll.

`persist-key?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
Die Schlüsseldateien nach Auftreten von SIGUSR1 oder –ping-restart *nicht* erneut einlesen.

`persist-tun?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
Nach dem Auftreten von SIGUSR1 oder –ping-restart TUN/TAP-Geräte *nicht* schließen und wieder öffnen und auch keine Start-/Stop-Skripte ausführen.

`fast-io?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
(Experimentell) Schreibzugriffe durch Datenverkehr bei TUN/TAP/UDP optimieren, indem ein Aufruf von poll/epoll/select vor der Schreiboperation eingespart wird.

`verbosity` (Vorgabe: `3`) (Typ: Zahl)
Ausführlichkeitsstufe.

`tls-auth` (Vorgabe: `#f`) (Typ: TLS-Clientauthentifizierung)
Eine weitere HMAC-Authentifizierung zusätzlich zum TLS-Steuerungskanal einsetzen, um das System vor gezielten Überlastungsangriffen („Denial of Service“) zu schützen.

`auth-user-pass` (Typ: Vielleicht-Zeichenkette)
: Beim Server eine Authentisierung über Benutzername/Passwort durchführen. Die Option nimmt eine Datei, welche Benutzername und Passwort auf zwei Zeilen enthält. Benutzen Sie dafür *kein* dateiartiges Objekt, weil es in den Store eingelagert würde, wo es jeder Benutzer einsehen könnte.

`verify-key-usage?` (Vorgabe: `#t`) (Typ: Schlüsselnutzung)
: Ob sichergestellt werden soll, dass das Server-Zertifikat auch über eine Erweiterung („Extension“) verfügt, dass es für die Nutzung als Server gültig ist.

`bind?` (Vorgabe: `#f`) (Typ: Binden)
: Ob an immer dieselbe, feste lokale Portnummer gebunden wird.

`resolv-retry?` (Vorgabe: `#t`) (Typ: resolv-retry)
: Ob, wenn die Server-Adresse nicht aufgelöst werden konnte, die Auflösung erneut versucht wird.

`remote` (Vorgabe: `()`) (Typ: Liste-von-„openvpn-remote“)
: Eine Liste entfernter Server, mit denen eine Verbindung hergestellt werden soll.

  `openvpn-remote-configuration` [Datentyp]
  : Verfügbare `openvpn-remote-configuration`-Felder sind:

    `name` (Vorgabe: `"my-server"`) (Typ: Zeichenkette)
    : Der Servername.

    `port` (Vorgabe: `1194`) (Typ: Zahl)
    : Die Portnummer, auf der der Server lauscht.

`openvpn-server-configuration` [Datentyp]
: Verfügbare `openvpn-server-configuration`-Felder sind:

  `openvpn` (Vorgabe: `openvpn`) (Typ: dateiartig)
  : Das OpenVPN-Paket.

  `pid-file` (Vorgabe: `"/var/run/openvpn/openvpn.pid"`) (Typ: Zeichenkette)
  : Die Datei für den Prozessidentifikator („PID“) von OpenVPN.

  `proto` (Vorgabe: `udp`) (Typ: Protokoll)
  : Das Protokoll (UDP oder TCP), das benutzt werden soll, um einen Kommunikationskanal zwischen Clients und Servern herzustellen.

  `dev` (Vorgabe: `tun`) (Typ: Gerätetyp)
  : Der Gerätetyp, mit dem die VPN-Verbindung repräsentiert werden soll.

  `ca` (Vorgabe: `"/etc/openvpn/ca.crt"`) (Typ: Vielleicht-Zeichenkette)
  : Die Zertifikatsautorität, mit der Verbindungen geprüft werden.

  `cert` (Vorgabe: `"/etc/openvpn/client.crt"`) (Typ: Vielleicht-Zeichenkette)
  : Das Zertifikat der Maschine, auf der der Daemon läuft. Es sollte von der in `ca` angegebenen Zertifikatsautorität signiert sein.

`key` (Vorgabe: `"/etc/openvpn/client.key"`) (Typ: Vielleicht-Zeichenkette)
: Der Schlüssel der Maschine, auf der der Daemon läuft. Er muss der Schlüssel zum in `cert` angegebenen Zertifikat sein.

`comp-lzo?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Ob der Kompressionsalgorithmus lzo benutzt werden soll.

`persist-key?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Die Schlüsseldateien nach Auftreten von SIGUSR1 oder –ping-restart *nicht* erneut einlesen.

`persist-tun?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Nach dem Auftreten von SIGUSR1 oder –ping-restart TUN/TAP-Geräte *nicht* schließen und wieder öffnen und auch keine Start-/Stop-Skripte ausführen.

`fast-io?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: (Experimentell) Schreibzugriffe durch Datenverkehr bei TUN/TAP/UDP optimieren, indem ein Aufruf von poll/epoll/select vor der Schreiboperation eingespart wird.

`verbosity` (Vorgabe: `3`) (Typ: Zahl)
: Ausführlichkeitsstufe.

`tls-auth` (Vorgabe: `#f`) (Typ: TLS-Serverauthentifizierung)
: Eine weitere HMAC-Authentifizierung zusätzlich zum TLS-Steuerungskanal einsetzen, um das System vor gezielten Überlastungsangriffen („Denial of Service“) zu schützen.

`port` (Vorgabe: `1194`) (Typ: Zahl)
: Gibt die Portnummer an, auf der der Server lauscht.

`server` (Vorgabe: `"10.8.0.0 255.255.255.0"`) (Typ: IP-Maske)
: Eine IP-Adresse gefolgt von deren Maske, die das Subnetz im virtuellen Netzwerk angibt.

`server-ipv6` (Vorgabe: `#f`) (Typ: CIDR6)
: Eine CIDR-Notation, mit der das IPv6-Subnetz im virtuellen Netzwerk angegeben wird.

`dh` (Vorgabe: `"/etc/openvpn/dh2048.pem"`) (Typ: Zeichenkette)
: Die Datei mit den Diffie-Hellman-Parametern.

`ifconfig-pool-persist` (Vorgabe: `"/etc/openvpn/ipp.txt"`) (Typ: Zeichenkette)
: Die Datei, in der Client-IPs eingetragen werden.

`redirect-gateway?` (Vorgabe: `#f`) (Typ: Zugang)
: Wenn dies auf wahr steht, fungiert der Server als Zugang („Gateway“) für seine Clients.

`client-to-client?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Wenn dies auf wahr steht, ist es zugelassen, dass Clients untereinander innerhalb des VPNs kommunizieren.

`keepalive` (Vorgabe: `(10 120)`) (Typ: keepalive)
: Lässt ping-artige Nachrichtigen in beide Richtungen über die Leitung übertragen, damit beide Seiten bemerken, wenn der Kommunikationspartner offline geht. Für `keepalive` muss ein Paar angegeben werden. Das erste Element ist die Periode, nach der das Pingsignal wieder gesendet werden soll, und das zweite ist eine Zeitbegrenzung, in der eines ankommen muss, damit die andere Seite nicht als offline gilt.

`max-clients` (Vorgabe: `100`) (Typ: Zahl)
: Wie viele Clients es höchstens geben kann.

`status` (Vorgabe: `"/var/run/openvpn/status"`) (Typ: Zeichenkette)
: Die Datei für einen Zustandsbericht („Status File“). Diese Datei enthält einen kurzen Bericht über die momentane Verbindung. Sie wird jede Minute gekürzt und neu geschrieben.

`client-config-dir` (Vorgabe: `()`) (Typ: Liste-von-„openvpn-ccd“)
: Die Liste der Konfigurationen für einige Clients.

strongSwan

Derzeit unterstützt der strongSwan-Dienst nur die alte Art der Konfiguration über die Dateien `ipsec.conf` und `ipsec.secrets`.

`strongswan-service-type` [Scheme-Variable]
: Ein Diensttyp, um mit strongSwan VPN (Virtual Private Networking) über IPsec einzurichten. Als Wert muss ein `strongswan-configuration`-Verbundsobjekt angegeben werden, wie im folgenden Beispiel:

```
(service strongswan-service-type
         (strongswan-configuration
          (ipsec-conf "/etc/ipsec.conf")
          (ipsec-secrets "/etc/ipsec.secrets")))
```

`strongswan-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration des strongSwan-Dienstes repräsentiert.

`strongswan`
: Das strongSwan-Paket, was für diesen Dienst benutzt werden soll.

`ipsec-conf` (Vorgabe: `#f`)
: Der Dateiname Ihrer `ipsec.conf`. Wenn es wahr ist, muss hierfür und für `ipsec-secrets` jeweils eine Zeichenkette angegeben werden.

`ipsec-secrets` (Vorgabe: `#f`)
: Der Dateiname Ihrer `ipsec.secrets`. Wenn es wahr ist, muss hierfür und für `ipsec-conf` jeweils eine Zeichenkette angegeben werden.

Wireguard

`wireguard-service-type` [Scheme-Variable]
: Ein Diensttyp für eine Schnittstelle über einen Wireguard-Tunnel. Sein Wert muss ein `wireguard-configuration`-Verbundsobjekt wie in diesem Beispiel sein:

```
(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` [Datentyp]

Der Datentyp, der die Konfiguration des Wireguard-Dienstes repräsentiert.

`wireguard`
: Das Wireguard-Paket, was für diesen Dienst benutzt werden soll.

`interface` (Vorgabe: `"wg0"`)
: Der Name der VPN-Schnittstelle.

`addresses` (Vorgabe: `'("10.0.0.1/32")`)
: Welche IP-Adressen obiger Schnittstelle zugeordnet werden.

`port` (Vorgabe: `51820`)
: Auf welchem Port auf eingehende Verbindungen gelauscht werden soll.

`dns` (Vorgabe: `#f`)
: Die DNS-Server, die den VPN-Clients per DHCP mitgeteilt werden.

`private-key` (Vorgabe: `"/etc/wireguard/private.key"`)
: Die Datei mit dem privaten Schlüssel für die Schnittstelle. Wenn die angegebene Datei nicht existiert, wird sie automatisch erzeugt.

`peers` (Vorgabe: `'()`)
: Die zugelassenen Netzwerkteilnehmer auf dieser Schnittstelle. Dies ist eine Liste von *wireguard-peer*-Verbundsobjekten.

`pre-up` (Vorgabe: `'()`)
: Welche Skriptbefehle vor dem Einrichten der Schnittstelle ausgeführt werden soll.

`post-up` (Vorgabe: `'()`)
: Welche Skriptbefehle nach dem Einrichten der Schnittstelle ausgeführt werden soll.

`pre-down` (Vorgabe: `'()`)
: Welche Skriptbefehle vor dem Abwickeln der Schnittstelle ausgeführt werden soll.

`post-down` (Vorgabe: `'()`)
: Welche Skriptbefehle nach dem Abwickeln der Schnittstelle ausgeführt werden soll.

`table` (Vorgabe: `"auto"`)
: Zu welcher Routing-Tabelle neue Routen hinzugefügt werden, angegeben als Zeichenkette. Es gibt zwei besondere Werte: Mit `"off"` schalten Sie

die Erzeugung neuer Routen ganz ab, mit `"auto"` (was vorgegeben ist) werden Routen zur Standard-Routingtabelle hinzugefügt und Standardrouten besonders behandelt.

`wireguard-peer` [Datentyp]
Der Datentyp, der einen an die angegebene Schnittstelle angeschlossenen Netzwerkteilnehmer („Peer“) repräsentiert.

`name` Die Bezeichnung für den Netzwerkteilnehmer.

`endpoint` (Vorgabe: `#f`)
Optional der Endpunkt des Netzwerkteilnehmers, etwa `"demo.wireguard.com:51820"`.

`public-key`
Der öffentliche Schlüssel des Netzwerkteilnehmers, dargestellt als eine base64-kodierte Zeichenkette.

`allowed-ips`
Eine Liste der IP-Adressen, von denen für diesen Netzwerkteilnehmer eingehende Kommunikation zugelassen wird und an die ausgehende Kommunikation für diesen Netzwerkteilnehmer gerichtet wird.

`keep-alive` (Vorgabe: `#f`)
Optional ein Zeitintervall in Sekunden. Einmal pro Zeitintervall wird ein Paket an den Serverendpunkt geschickt. Das hilft beim Empfangen eingehender Verbindungen von diesem Netzwerkteilnehmer, wenn Sie nur über eine Netzwerkadressübersetzung („NAT“) oder hinter einer Firewall erreichbar sind.

12.9.24 Network File System

Das Modul `(gnu services nfs)` stellt die folgenden Dienste zur Verfügung, die meistens dazu benutzt werden, Verzeichnisbäume als Netzwerkdateisysteme, englisch *Network File Systems* (NFS), einzubinden oder an andere zu exportieren.

Obwohl man auch die einzelnen Komponenten eines Network-File-System-Dienstes separat einrichten kann, raten wir dazu, einen NFS-Server mittels `nfs-service-type` zu konfigurieren.

NFS-Dienst

Der NFS-Dienst sorgt dafür, dass alle NFS-Komponentendienste, die Konfiguration des NFS-Kernels und Dateisysteme eingerichtet werden, und er installiert an den von NFS erwarteten Orten Konfigurationsdateien.

`nfs-service-type` [Scheme-Variable]
Ein Diensttyp für einen vollständigen NFS-Server.

`nfs-configuration` [Datentyp]
Dieser Datentyp repräsentiert die Konfiguration des NFS-Dienstes und all seiner Subsysteme.

Er hat folgende Parameter:

`nfs-utils` (Vorgabe: `nfs-utils`)
: Das nfs-utils-Paket, was benutzt werden soll.

`nfs-versions` (Vorgabe: `'("4.2" "4.1" "4.0")`)
: Wenn eine Liste von Zeichenketten angegeben wird, beschränkt sich der `rpc.nfsd`-Daemon auf die Unterstützung der angegebenen Versionen des NFS-Protokolls.

`exports` (Vorgabe: `'()`)
: Diese Liste von Verzeichnissen soll der NFS-Server exportieren. Jeder Eintrag ist eine Liste, die zwei Elemente enthält: den Namen eines Verzeichnisses und eine Zeichenkette mit allen Optionen. Ein Beispiel, wo das Verzeichnis `/export` an alle NFS-Clients nur mit Lesezugriff freigegeben wird, sieht so aus:

```
(nfs-configuration
 (exports
  '(("/export"
     "*(ro,insecure,no_subtree_check,crossmnt,fsid=0)"))))
```

`rpcmountd-port` (Vorgabe: `#f`)
: Welchen Netzwerk-Port der `rpc.mountd`-Daemon benutzen soll.

`rpcstatd-port` (Vorgabe: `#f`)
: Welchen Netzwerk-Port der `rpc.statd`-Daemon benutzen soll.

`rpcbind` (Vorgabe: `rpcbind`)
: Das rpcbind-Paket, das benutzt werden soll.

`idmap-domain` (Vorgabe: `"localdomain"`)
: Der lokale NFSv4-Domainname.

`nfsd-port` (Vorgabe: `2049`)
: Welchen Netzwerk-Port der `nfsd`-Daemon benutzen soll.

`nfsd-threads` (Vorgabe: `8`)
: Wie viele Threads der `rpc.statd`-Daemon benutzen soll.

`nfsd-tcp?` (Vorgabe: `#t`)
: Ob der `nfsd`-Daemon auf einem TCP-Socket lauschen soll.

`nfsd-udp?` (Vorgabe: `#f`)
: Ob der `nfsd`-Daemon auf einem UDP-Socket lauschen soll.

`pipefs-directory` (Vorgabe: `"/var/lib/nfs/rpc_pipefs"`)
: Das Verzeichnis, unter dem das Pipefs-Dateisystem eingebunden wurde.

`debug` (Vorgabe: `'()"`)
: Eine Liste der Subsysteme, für die Informationen zur Fehlersuche bereitgestellt werden sollen, als Liste von Symbolen. Jedes der folgenden Symbole ist gültig: `nfsd`, `nfs`, `rpc`, `idmap`, `statd` oder `mountd`.

Wenn Sie keinen vollständigen NFS-Dienst benötigen oder ihn selbst einrichten wollen, können Sie stattdessen die im Folgenden dokumentierten Komponentendienste benutzen.

RPC-Bind-Dienst

Mit dem RPC-Bind-Dienst können Programmnummern auf universelle Adressen abgebildet werden. Viele Dienste, die mit dem NFS zu tun haben, benutzen diese Funktion, daher wird sie automatisch gestartet, sobald ein davon abhängiger Dienst startet.

`rpcbind-service-type` [Scheme-Variable]
Ein Diensttyp für den RPC-Portplaner-Daemon („Portmapper“).

`rpcbind-configuration` [Datentyp]
Datentyp, der die Konfiguration des RPC-Bind-Dienstes repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`rpcbind` (Vorgabe: `rpcbind`)
Das rpcbind-Paket, das benutzt werden soll.

`warm-start?` (Vorgabe: `#t`)
Wenn dieser Parameter `#t` ist, wird der Daemon beim Starten eine Zustandsdatei („State File“) lesen, aus der er die von der vorherigen Instanz gespeicherten Zustandsinformationen wiederherstellt.

Pipefs-Pseudodateisystem

Mit dem Pipefs-Dateisystem können NFS-bezogene Daten zwischen dem Kernel und Programmen auf der Anwendungsebene (dem „User Space“) übertragen werden.

`pipefs-service-type` [Scheme-Variable]
Ein Diensttyp für das Pseudodateisystem „Pipefs“.

`pipefs-configuration` [Datentyp]
Datentyp, der die Konfiguration des Pipefs-Pseudodateisystemdienstes repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`mount-point` (Vorgabe: `"/var/lib/nfs/rpc_pipefs"`)
Das Verzeichnis, unter dem das Dateisystem eingebunden werden soll.

GSS-Daemon-Dienst

Der Daemon des *Global Security System* (GSS) ermöglicht starke Informationssicherheit für RPC-basierte Protokolle. Vor dem Austausch von Anfragen über entfernte Prozeduraufrufe („Remote Procedure Calls“, kurz RPC) muss ein RPC-Client einen Sicherheitskontext („Security Context“) herstellen. Typischerweise wird dazu der `kinit`-Befehl von Kerberos benutzt, oder er wird automatisch bei der Anmeldung über PAM-Dienste hergestellt (siehe Abschnitt 12.9.17 [Kerberos-Dienste], Seite 458).

`gss-service-type` [Scheme-Variable]
Ein Diensttyp für den Daemon des Global Security System (GSS).

`gss-configuration` [Datentyp]
Datentyp, der die Konfiguration des GSS-Daemon-Dienstes repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

`nfs-utils` (Vorgabe: `nfs-utils`)
Das Paket, in dem der Befehl `rpc.gssd` gesucht werden soll.

`pipefs-directory` (Vorgabe: `"/var/lib/nfs/rpc_pipefs"`)
: Das Verzeichnis, unter dem das Pipefs-Dateisystem eingebunden wurde.

IDMAP-Daemon-Dienst

Der idmap-Daemon-Dienst ermöglicht eine Abbildung zwischen Benutzeridentifikatoren und Benutzernamen. Er wird in der Regel dafür benötigt, auf über NFSv4 eingebundene Dateisysteme zuzugreifen.

`idmap-service-type` [Scheme-Variable]
: Ein Diensttyp für den Identity-Mapper-Daemon (IDMAP).

`idmap-configuration` [Datentyp]
: Datentyp, der die Konfiguration des IDMAP-Daemon-Dienstes repräsentiert. Dieser Typ verfügt über die folgenden Parameter:

  `nfs-utils` (Vorgabe: `nfs-utils`)
  : Das Paket, in dem der Befehl `rpc.idmapd` gesucht werden soll.

  `pipefs-directory` (Vorgabe: `"/var/lib/nfs/rpc_pipefs"`)
  : Das Verzeichnis, unter dem das Pipefs-Dateisystem eingebunden wurde.

  `domain` (Vorgabe: `#f`)
  : Der lokale NFSv4-Domain-Name. Für ihn muss eine Zeichenkette oder `#f` angegeben werden. Wenn `#f` angegeben wird, benutzt der Daemon den vollständigen Domain-Namen („Fully Qualified Domain Name“) des Rechners.

  `verbosity` (Vorgabe: `0`)
  : Die Ausführlichkeitsstufe des Daemons.

12.9.25 Samba-Dienste

Das Modul `(gnu services samba)` stellt Dienstdefinitionen für Samba und zugehörige Hilfsdienste zur Verfügung. Zurzeit werden die folgenden Dienste unterstützt.

Samba

Mit Samba (`https://www.samba.org`) wird das Teilen von Verzeichnissen und Druckern im Netzwerk über das Protokoll SMB/CIFS ermöglicht, was in Windows häufig verwendet wird. Mit Samba kann auch die Rolle eines Active-Directory-Domain-Controllers (AD DC) für andere Rechner in einem heterogenen Netzwerk mit vielen Arten von Computersystemen übernommen werden.

`Scheme-Variable` *samba-service-type* [Variable]
: Der Diensttyp, um die Samba-Dienste `samba`, `nmbd`, `smbd` und `winbindd` zu aktivieren. Nach der Vorgabeeinstellung wird noch keiner der Samba-Daemons ausgeführt. Sie müssen einzeln angeben, welche Sie aktiviert haben möchten.

  Hier sehen Sie ein grundlegendes Beispiel einer schlichten, anonymen (ohne Authentifizierung) Samba-Dateifreigabe, wodurch Zugriff auf das Verzeichnis `/public` gewährt wird.

Tipp: Es wird vorausgesetzt, dass alle Benutzerkonten auf das Verzeichnis `/public` und enthaltene Dateien Lese- und Schreibzugriff haben, also möchten Sie vielleicht '`chmod -R 777 /public`' darauf ausführen.

Vorsicht: Sie sollten so eine Samba-Konfiguration nur in kontrollierten Umgebungen einsetzen und darüber keine privaten Dateien teilen, denn jeder in Ihrem Netzwerk würde darauf zugreifen können.

```
(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` *Verbundsobjekt für die* [Datentyp]
Konfiguration der Samba-Programme.

`package` (Vorgabe: `samba`)
: Das Samba-Paket, das benutzt werden soll.

`config-file` (Vorgabe: `#f`)
: Welche Konfigurationsdatei benutzt werden soll. Die Syntax können Sie erfahren, wenn Sie '`man smb.conf`' ausführen.

`enable-samba?` (Vorgabe: `#f`)
: Den `samba`-Daemon aktivieren.

`enable-smbd?` (Vorgabe: `#f`)
: Den `smbd`-Daemon aktivieren.

`enable-nmbd?` (Vorgabe: `#f`)
: Den `nmbd`-Daemon aktivieren.

`enable-winbindd?` (Vorgabe: `#f`)
: Den `winbindd`-Daemon aktivieren.

Web Service Discovery Daemon

Mit dem WSDD (Web-Service-Discovery-Daemon) wird das Protokoll Web Services Dynamic Discovery (`http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html`) zum Finden anderer Rechner über Multicast DNS aktiviert, ähnlich zu dem, was Avahi macht. Es kann als Alternative einspringen für Rechner mit SMB, auf denen SMBv1 aus Sicherheitsgründen abgeschaltet wurde.

`wsdd-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den WSD-Host-Daemon. Als Wert wird für diesen Diensttyp ein Verbundsobjekt vom Typ `wsdd-configuration` benutzt. Die Details zum `wsdd-configuration`-Verbundstyp folgen nun.

`wsdd-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des wsdd-Dienstes.

`package` (Vorgabe: `wsdd`)
: Das zu benutzende wsdd-Paket.

`ipv4only?` (Vorgabe: `#f`)
: Lässt nur auf IPv4-Adressen lauschen.

`ipv6only` (Vorgabe: `#f`)
: Lässt nur auf IPv6-Adressen lauschen. Beachten Sie: Es ist *nicht* möglich, beide Optionen zu aktivieren, denn sonst würde `wsdd` auf gar keiner IP-Version lauschen.

`chroot` (Vorgabe: `#f`)
: Mit Chroot in ein eigenes Verzeichnis wechseln und damit den Zugriff auf ungewollte Verzeichnisse verhindern. Dies bietet zusätzliche Sicherheit, wenn es eine Schwachstelle in `wsdd` geben sollte.

`hop-limit` (Vorgabe: `1`)
: Beschränkung, wie viele Zwischenschritte („Hops“) Multicast-Pakete nehmen dürfen. Voreingestellt ist *1*, um zu verhindern, dass Pakete das lokale Netzwerk verlassen.

`interface` (Vorgabe: `'()`)
: Nur auf den hier angegebenen Netzwerkschnittstellen lauschen. Die Vorgabe ist, dass wsdd auf allen Schnittstellen erreichbar ist, außer der Loopback-Schnittstelle, die niemals benutzt wird.

`uuid-device` (Vorgabe: `#f`)
: Im WSD-Protokoll muss ein Gerät eine UUID haben. Sie können hier eine UUID für den Dienst selbst vergeben.

`domain` (Vorgabe: `#f`)
: Wenn gemeldet werden soll, dass dieser Rechner zu einer Active Directory gehört.

`host-name` (Vorgabe: `#f`)
: Legen Sie den Rechnernamen selbst fest, statt dass `wsdd` den Rechnernamen des Rechners verwendet. Normalerweise wird aus einem möglichen vollständigen Domain-Namen („Fully Qualified Domain Name“) des Rechners nur der Teil mit dem Rechnernamen benutzt.

`preserve-case?` (Vorgabe: `#f`)
: Vorgegeben ist, dass `wsdd` die Rechnernamen in der Arbeitsgruppe („Workgroup“) in Großbuchstaben wiedergibt. Das Gegenteil ist der Fall, wenn der Rechnername aus einer Domain kommt. Wenn Sie diesen Parameter auf wahr setzen, bleibt die Groß-/Kleinschreibung erhalten.

`workgroup` (Vorgabe: "*WORKGROUP*")
: Hier können Sie den Namen der Arbeitsgruppe ändern. `wsdd` wird diesen Rechner als Mitglied dieser Arbeitsgruppe melden.

12.9.26 Kontinuierliche Integration

Cuirass (`https://guix.gnu.org/cuirass/`) ist ein Werkzeug zur kontinuierlichen Integration für Guix. Es kann sowohl bei der Entwicklung helfen als auch beim Anbieten von Substituten für andere (siehe Abschnitt 6.3 [Substitute], Seite 56).

Das Modul `(gnu services cuirass)` stellt den folgenden Dienst zur Verfügung:

`cuirass-service-type` [Scheme-Prozedur]
: Der Diensttyp des Cuirass-Dienstes. Sein Wert muss ein `cuirass-configuration`-Verbundsobjekt sein, wie im Folgenden beschrieben.

Um Erstellungsaufträge („Build Jobs“) hinzuzufügen, müssen Sie sie im `specifications`-Feld der Konfiguration eintragen. Mit dem folgenden Beispiel würden alle Pakete aus dem Kanal `my-channel` erstellt.

```
(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 %cuirass-specs)))
```

Um das Paket `linux-libre`, das im vorgegebenen Guix-Kanal definiert ist, zu erstellen, schreibe man die folgende Konfiguration.

```
(define %cuirass-specs
  #~(list (specification
           (name "my-linux")
           (build '(packages "linux-libre")))))

(service cuirass-service-type
         (cuirass-configuration
          (specifications %cuirass-specs)))
```

Für eine Beschreibung der anderen Konfigurationsmöglichkeiten und des Spezifikations-Verbundsobjektes selbst, siehe das Abschnitt “Specifications” in *Handbuch zu Cuirass*.

Die Informationen, die sich auf Erstellungsaufträge beziehen, werden direkt in deren Spezifikation festgelegt, aber globale Einstellungen des `cuirass`-Prozesses sind über andere Felder der `cuirass-configuration` zugänglich.

`cuirass-configuration` [Datentyp]

Datentyp, der die Konfiguration von Cuirass repräsentiert.

`cuirass` (Vorgabe: `cuirass`)
: Das Cuirass-Paket, das benutzt werden soll.

`log-file` (Vorgabe: `"/var/log/cuirass.log"`)
: An welchen Ort die Protokolldatei geschrieben wird.

`web-log-file` (Vorgabe: `"/var/log/cuirass-web.log"`)
: An welchem Ort die Protokolldatei der Weboberfläche gespeichert wird.

`cache-directory` (Vorgabe: `"/var/cache/cuirass"`)
: Ort, wo Repositorys zwischengespeichert werden.

`user` (Vorgabe: `"cuirass"`)
: Besitzer des `cuirass`-Prozesses.

`group` (Vorgabe: `"cuirass"`)
: Gruppe des Besitzers des `cuirass`-Prozesses.

`interval` (Vorgabe: `60`)
: Anzahl der Sekunden, bevor ein Repository wieder neu geladen wird und danach Cuirass-Aufträge behandelt werden.

`parameters` (Vorgabe: `#f`)
: Parameter aus der angegebenen Parameterdatei lesen. Die unterstützten Parameter werden im Abschnitt “Parameters” in *Cuirass-Handbuch* beschrieben.

`remote-server` (Vorgabe: `#f`)
: Ein `cuirass-remote-server-configuration`-Verbundsobjekt, um den Mechanismus für entfernte Erstellungen zu benutzen, oder `#f`, um den voreingestellten Erstellungsmechanismus zu benutzen.

`database` (Vorgabe: `"dbname=cuirass host=/var/run/postgresql"`)
: *database* als die Datenbank mit den Aufträgen und bisherigen Erstellungsergebnissen benutzen. Weil Cuirass als Datenbanktreiber PostgreSQL benutzt, muss *database* eine Zeichenkette sein wie `"dbname=cuirass host=localhost"`.

`port` (Vorgabe: `8081`)
: Portnummer, die vom HTTP-Server benutzt wird.

`host` (Vorgabe: `"localhost"`)
: Auf der Netzwerkschnittstelle für den Rechnernamen *host* lauschen. Nach Vorgabe werden Verbindungen vom lokalen Rechner `localhost` akzeptiert.

`specifications` (Vorgabe: `#~'()`)
: Ein G-Ausdruck (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), der zu einer Liste von Spezifikations-Verbundsobjekten ausgewertet wird. Spezifikations-Verbundsobjekte werden im Abschnitt “Specifications” in *Cuirass-Handbuch* beschrieben.

`use-substitutes?` (Vorgabe: `#f`)
: Hierdurch wird zugelassen, Substitute zu benutzen, damit *nicht* jede Abhängigkeit eines Auftrags erst aus ihrem Quellcode heraus erstellt werden muss.

`one-shot?` (Vorgabe: `#f`)
: Spezifikationen nur einmal auswerten und Ableitungen nur einmal erstellen.

`fallback?` (Vorgabe: `#f`)
: Pakete lokal erstellen, wenn das Substituieren einer vorerstellten Binärdatei fehlschlägt.

`extra-options` (Vorgabe: `'()`)
: Zusätzliche Befehlszeilenoptionen, die beim Ausführen des Cuirass-Prozesses mitgegeben werden sollen.

Cuirass, entfernte Erstellung mit

Cuirass kennt zwei Mechanismen, wie damit Ableitungen erstellt werden können.

- Den lokalen Guix-Daemon benutzen. Dies ist der voreingestellte Erstellungsmechanismus. Sobald Erstellungsaufträge ausgewertet wurden, werden sie an den Guix-Daemon auf dem lokalen Rechner geschickt. Cuirass lauscht dann auf die Ausgabe des Guix-Daemons, um die verschiedenen Ereignisse bei der Erstellung zu erkennen.
- Den Mechanismus für entfernte Erstellungen benutzen. Die Erstellungsaufträge werden *nicht* beim lokalen Guix-Daemon eingereicht. Stattdessen teilt ein entfernter Server die Erstellungsanfragen den verbundenen entfernten Arbeitermaschinen zu, entsprechend ihren Erstellungsprioritäten.

Um diesen Erstellungsmodus zu aktivieren, übergeben Sie ein `cuirass-remote-server-configuration`-Verbundsobjekt für das Argument `remote-server` des `cuirass-configuration`-Verbundsobjektes. Der `cuirass-remote-server-configuration`-Verbund wird im Folgenden beschrieben.

Dieser Erstellungsmodus skaliert wesentlich besser als der Vorgabemodus. Mit diesem Erstellungsmodus wird auch die Erstellungfarm von GNU Guix auf `https://ci.guix.gnu.org` betrieben. Er sollte dann bevorzugt werden, wenn Sie mit Cuirass eine große Anzahl von Paketen erstellen möchten.

`cuirass-remote-server-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration des Cuirass-„remote-server“-Prozesses repräsentiert.

  `backend-port` (Vorgabe: `5555`)
  : Der TCP-Port, um mit `remote-worker`-Prozessen mit ZMQ zu kommunizieren.

  `log-port` (Vorgabe: `5556`)
  : Der TCP-Port des Protokollservers.

  `publish-port` (Vorgabe: `5557`)
  : Der TCP-Port des „Publish“-Servers, um Substitute mit anderen zu teilen.

`log-file` (Vorgabe: `"/var/log/cuirass-remote-server.log"`)
: An welchen Ort die Protokolldatei geschrieben wird.

`cache` (Vorgabe: `"/var/cache/cuirass/remote"`)
: Im *cache*-Verzeichnis Erstellungsprotokolldateien speichern.

`trigger-url` (Vorgabe: `#f`)
: Sobald ein Substitut erfolgreich heruntergeladen wurde, wird dessen Einlagerung als Narinfo („bake the substitute“) auf *trigger-url* direkt ausgelöst.

`publish?` (Vorgabe: `#t`)
: Wenn dies auf falsch gesetzt ist, wird kein „Publish“-Server gestartet und das `publish-port`-Argument ignoriert. Verwenden Sie es, wenn bereits ein eigenständiger „Publish“-Server für den „remote-server“ läuft.

`public-key`
`private-key`
: Die angegebenen *Datei*en als das Paar aus öffentlichem und privatem Schlüssel zum Signieren veröffentlichter Store-Objekte benutzen.

Mindestens eine entfernte Arbeitermaschine („remote worker“) muss auf irgendeiner Maschine im lokalen Netzwerk gestartet werden, damit tatsächlich Erstellungen durchgeführt werden und deren Status gemeldet wird.

`cuirass-remote-worker-configuration` [Datentyp]
Der Datentyp, der die Konfiguration des Cuirass-„remote-worker“-Prozesses repräsentiert.

`cuirass` (Vorgabe: `cuirass`)
: Das Cuirass-Paket, das benutzt werden soll.

`workers` (Vorgabe: `1`)
: *workers*-viele parallele Arbeiterprozesse starten.

`server` (Vorgabe: `#f`)
: Keine Maschinen über Avahi ermitteln, sondern stattdessen eine Verbindung zum Server unter der in `server` angegebenen IP-Adresse aufbauen.

`systems` (Vorgabe: `(list (%current-system))`)
: Nur Erstellungen für die in *systems* angegebenen Systeme anfragen.

`log-file` (Vorgabe: `"/var/log/cuirass-remote-worker.log"`)
: An welchen Ort die Protokolldatei geschrieben wird.

`publish-port` (Vorgabe: `5558`)
: Der TCP-Port des „Publish“-Servers, um Substitute mit anderen zu teilen.

`substitute-urls` (Vorgabe: `%default-substitute-urls`)
: Die Liste der URLs, auf denen nach Substituten gesucht wird, wenn nicht anders angegeben.

`public-key`
`private-key`
: Die angegebenen *Datei*en als das Paar aus öffentlichem und privatem Schlüssel zum Signieren veröffentlichter Store-Objekte benutzen.

Laminar

Laminar (`https://laminar.ohwg.net/`) ist ein sparsamer und modularer Dienst zur Kontinuierlichen Integration. Statt einer Weboberfläche zur Konfiguration werden versionskontrollierte Konfigurationsdateien und Skripte verwendet.

Laminar ermutigt dazu, bestehende Werkzeuge wie bash und cron zu benutzen, statt das Rad neu zu erfinden.

`laminar-service-type` [Scheme-Prozedur]
: Der Diensttyp des Laminar-Dienstes. Sein Wert muss ein `laminar-configuration`-Verbundsobjekt sein, wie im Folgenden beschrieben.

  Alle Konfigurationswerte haben bereits vorgegebene Einstellungen. Eine minimale Konfiguration, um Laminar aufzusetzen, sehen Sie unten. Nach Vorgabe läuft die Weboberfläche auf Port 8080.

  ```
  (service laminar-service-type)
  ```

`laminar-configuration` [Datentyp]
: Datentyp, der die Konfiguration von Laminar repräsentiert.

  `laminar` (Vorgabe: `laminar`)
  : Das Laminar-Paket, das benutzt werden soll.

  `home-directory` (Vorgabe: `"/var/lib/laminar"`)
  : Das Verzeichnis mit Auftragskonfigurationen und Verzeichnissen, in denen Aufträge ausgeführt werden („run“-Verzeichnissen).

  `bind-http` (Vorgabe: `"*:8080"`)
  : Die Netzwerkschnittstelle mit Port oder der Unix-Socket, wo laminard auf eingehende Verbindungen zur Web-Oberfläche lauschen soll.

  `bind-rpc` (Vorgabe: `"unix-abstract:laminar"`)
  : Die Netzwerkschnittstelle mit Port oder der Unix-Socket, wo laminard auf eingehende Befehle z.B. zum Auslösen einer Erstellung lauschen soll.

  `title` (Vorgabe: `"Laminar"`)
  : Der Seitentitel, der auf der Web-Oberfläche angezeigt wird.

  `keep-rundirs` (Vorgabe: `0`)
  : Setzen Sie dies auf eine ganze Zahl, die festlegt, wie viele Ausführungsverzeichnisse pro Auftrag vorgehalten werden. Die mit der niedrigsten Nummer werden gelöscht. Die Vorgabe ist 0, d.h. alle Ausführungsverzeichnisse werden sofort gelöscht.

  `archive-url` (Vorgabe: `#f`)
  : Die von laminard angebotene Weboberfläche wird aus dieser URL die Verweise auf Artefakte archivierter Aufträge zusammensetzen.

`base-url` (Vorgabe: `#f`)
: Was Laminar in Verweisen auf eigene Seiten als Basis-URL verwenden soll.

12.9.27 Dienste zur Stromverbrauchsverwaltung

TLP-Daemon

Das Modul `(gnu services pm)` stellt eine Guix-Dienstdefinition für das Linux-Werkzeug TLP zur Stromverbrauchsverwaltung zur Verfügung.

TLP macht mehrere Stromspar-Modi auf Anwendungsebene („User Space“) und im Kernel verfügbar. Im Gegensatz zum `upower-service` handelt es sich um kein passives Werkzeug zur Überwachung, sondern TLP passt selbst jedes Mal Einstellungen an, wenn eine neue Stromquelle erkannt wird. Mehr Informationen finden Sie auf der TLP-Homepage (`https://linrunner.de/en/tlp/tlp.html`).

`tlp-service-type` [Scheme-Variable]
: Der Diensttyp für das TLP-Werkzeug. In der Vorgabeeinstellung wird die Akkulaufzeit für die meisten Systeme optimiert, aber Sie können sie nach Belieben anpassen, indem Sie eine gültige `tlp-configuration` hinzufügen:

```
(service tlp-service-type
         (tlp-configuration
          (cpu-scaling-governor-on-ac (list "performance"))
          (sched-powersave-on-bat? #t)))
```

Im Folgenden ist jeder Parameterdefinition ihr Typ vorangestellt. Zum Beispiel bedeutet ‘`Boolescher-Ausdruck foo`’, dass der Parameter `foo` als boolescher Ausdruck festgelegt werden sollte. Typen, die mit `Vielleicht-` beginnen, bezeichnen Parameter, die nicht in der TLP-Konfigurationsdatei vorkommen, wenn Sie keinen Wert für sie setzen oder den Wert ausdrücklich auf `%unset-value` setzen.

Verfügbare `tlp-configuration`-Felder sind:

`„package“ tlp` [`tlp-configuration`-Parameter]
: Das TLP-Paket.

`Boolescher-Ausdruck tlp-enable?` [`tlp-configuration`-Parameter]
: Setzen Sie dies auf wahr, wenn Sie TLP aktivieren möchten.

  Die Vorgabe ist ‘`#t`’.

`Zeichenkette tlp-default-mode` [`tlp-configuration`-Parameter]
: Der vorgegebene Modus, wenn keine Stromversorgung gefunden werden kann. Angegeben werden können AC (am Stromnetz) und BAT (Batterie/Akku).

  Die Vorgabe ist ‘`"AC"`’.

`Nichtnegative-ganze-Zahl disk-idle-secs-on-ac` [`tlp-configuration`-Parameter]
: Die Anzahl an Sekunden, die der Linux-Kernel warten muss, bis er sich mit dem Plattenspeicher synchronisiert, wenn die Stromversorgung auf AC steht.

  Die Vorgabe ist ‘`0`’.

`Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`disk-idle-secs-on-bat`

Wie `disk-idle-ac`, aber für den BAT-Modus.

Die Vorgabe ist '`2`'.

`Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`max-lost-work-secs-on-ac`

Periodizität, mit der im Zwischenspeicher geänderte Speicherseiten („Dirty Pages“) synchronisiert werden („Cache Flush“), ausgedrückt in Sekunden.

Die Vorgabe ist '`15`'.

`Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`max-lost-work-secs-on-bat`

Wie `max-lost-work-secs-on-ac`, aber für den BAT-Modus.

Die Vorgabe ist '`60`'.

`Vielleicht-Leerzeichengetrennte-Zeichenketten-Liste` [`tlp-configuration`-Parameter]
`cpu-scaling-governor-on-ac`

Regulator der Frequenzskalierung der CPUs („Frequency Scaling Governor“) im AC-Modus. Beim intel_pstate-Treiber stehen powersave (stromsparend) und performance (leistungsfähig) zur Auswahl. Beim acpi-cpufreq-Treiber stehen ondemand, powersave, performance und conservative zur Auswahl.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Leerzeichengetrennte-Zeichenketten-Liste` [`tlp-configuration`-Parameter]
`cpu-scaling-governor-on-bat`

Wie `cpu-scaling-governor-on-ac`, aber für den BAT-Modus.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`cpu-scaling-min-freq-on-ac`

Legt die minimale verfügbare Frequenz für den Skalierungsregulator im AC-Modus fest.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`cpu-scaling-max-freq-on-ac`

Legt die maximale verfügbare Frequenz für den Skalierungsregulator im AC-Modus fest.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`cpu-scaling-min-freq-on-bat`

Legt die minimale verfügbare Frequenz für den Skalierungsregulator im BAT-Modus fest.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`cpu-scaling-max-freq-on-bat`

Legt die maximale verfügbare Frequenz für den Skalierungsregulator im BAT-Modus fest.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`cpu-min-perf-on-ac`

Beschränkt den minimalen Leistungszustand („P-State"), um die Stromverteilung („Power Dissipation") der CPU im AC-Modus zu regulieren. Werte können als Prozentsatz bezüglich der verfügbaren Leistung angegeben werden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`cpu-max-perf-on-ac`

Beschränkt den maximalen Leistungszustand („P-State"), um die Stromverteilung („Power Dissipation") der CPU im AC-Modus zu regulieren. Werte können als Prozentsatz bezüglich der verfügbaren Leistung angegeben werden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`cpu-min-perf-on-bat`

Wie `cpu-min-perf-on-ac` im BAT-Modus.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`cpu-max-perf-on-bat`

Wie `cpu-max-perf-on-ac` im BAT-Modus.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck` [`tlp-configuration`-Parameter]
`cpu-boost-on-ac?`

Die CPU-Turbo-Boost-Funktionen im AC-Modus aktivieren.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck` [`tlp-configuration`-Parameter]
`cpu-boost-on-bat?`

Wie `cpu-boost-on-ac?` im BAT-Modus.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Boolescher-Ausdruck` [`tlp-configuration`-Parameter]
`sched-powersave-on-ac?`

Dem Linux-Kernel erlauben, die Anzahl benutzter CPU-Kerne und Hyperthreads anzupassen, wenn er unter leichter Last steht.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck` [`tlp-configuration`-Parameter]
`sched-powersave-on-bat?`
Wie `sched-powersave-on-ac?`, aber für den BAT-Modus.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck nmi-watchdog?` [`tlp-configuration`-Parameter]
Ob die rechtzeitige Behandlung nichtmaskierbarer Unterbrechungen durch den „NMI-Watchdog“ des Linux-Kernels überprüft werden soll.

Vorgegeben ist '`#f`'.

`Vielleicht-Zeichenkette phc-controls` [`tlp-configuration`-Parameter]
Auf Linux-Kernels, auf die der PHC-Patch angewandt wurde, wird hierdurch die Prozessorspannung angepasst. Ein Beispielwert wäre '`"F:V F:V F:V F:V"`'.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Zeichenkette energy-perf-policy-on-ac` [`tlp-configuration`-Parameter]
Legt das Verhältnis von Prozessorleistung zu Stromsparsamkeit im AC-Modus fest. Angegeben werden können performance (hohe Leistung), normal, powersave (wenig Stromverbrauch).

Die Vorgabe ist '`"performance"`'.

`Zeichenkette energy-perf-policy-on-bat` [`tlp-configuration`-Parameter]
Wie `energy-perf-policy-ac`, aber für den BAT-Modus.

Die Vorgabe ist '`"powersave"`'.

`Leerzeichengetrennte-Zeichenketten-Liste` [`tlp-configuration`-Parameter]
`disks-devices`
Festplattengeräte.

`Leerzeichengetrennte-Zeichenketten-Liste` [`tlp-configuration`-Parameter]
`disk-apm-level-on-ac`
Stufe für das „Advanced Power Management“ auf Festplatten.

`Leerzeichengetrennte-Zeichenketten-Liste` [`tlp-configuration`-Parameter]
`disk-apm-level-on-bat`
Wie `disk-apm-bat`, aber für den BAT-Modus.

`Vielleicht-Leerzeichengetrennte-Zeichenketten-Liste` [`tlp-configuration`-Parameter]
`disk-spindown-timeout-on-ac`
Zeitspanne, bis die Festplatte inaktiv wird (ein „Spin-Down“). Für jede deklarierte Festplatte muss hier je ein Wert angegeben werden.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Leerzeichengetrennte-Zeichenketten-Liste` [`tlp-configuration`-Parameter]
`disk-spindown-timeout-on-bat`
Wie `disk-spindown-timeout-on-ac`, aber für den BAT-Modus.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Leerzeichengetrennte-Zeichenkette` [`tlp-configuration`-Parameter]
`disk-iosched`

Ein-/Ausgaben-Planungsprogramm für Plattengeräte auswählen. Für jede deklarierte Festplatte muss ein Wert angegeben werden. Möglich sind zum Beispiel cfq, deadline und noop.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Zeichenkette sata-linkpwr-on-ac` [`tlp-configuration`-Parameter]

Stufe des „Aggressive Link Power Management" (ALPM) für SATA. Angegeben werden können min_power (wenigster Stromverbrauch), medium_power (mittlerer Stromverbrauch), max_performance (maximale Leistung).

Die Vorgabe ist '`"max_performance"`'.

`Zeichenkette sata-linkpwr-on-bat` [`tlp-configuration`-Parameter]

Wie `sata-linkpwr-ac`, aber für den BAT-Modus.

Die Vorgabe ist '`"min_power"`'.

`Vielleicht-Zeichenkette` [`tlp-configuration`-Parameter]
`sata-linkpwr-blacklist`

Bestimmte SATA-Geräte („SATA-Host-Devices") vom Link Power Management ausschließen.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-An-Aus-Boolescher-Ausdruck` [`tlp-configuration`-Parameter]
`ahci-runtime-pm-on-ac?`

Verwaltung des Stromverbrauchs zur Laufzeit für AHCI-Steuerungseinheiten („Controller") und AHCI-Platten im AC-Modus aktivieren.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-An-Aus-Boolescher-Ausdruck` [`tlp-configuration`-Parameter]
`ahci-runtime-pm-on-bat?`

Wie `ahci-runtime-pm-on-ac` im BAT-Modus.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`ahci-runtime-pm-timeout`

Nach wie vielen Sekunden der Inaktivität die Platten in den Ruhezustand gehen („Suspended").

Die Vorgabe ist '`15`'.

`Zeichenkette pcie-aspm-on-ac` [`tlp-configuration`-Parameter]

Stufe des „PCI Express Active State Power Management". Zur Auswahl stehen default (Voreinstellung), performance (hohe Leistung), powersave (wenig Stromverbrauch).

Die Vorgabe ist '`"performance"`'.

`Zeichenkette pcie-aspm-on-bat` [`tlp-configuration`-Parameter]

Wie `pcie-aspm-ac`, aber für den BAT-Modus.

Die Vorgabe ist '`"powersave"`'.

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`start-charge-thresh-bat0`
Ab welchem Prozentwert die Batterie 0 anfangen soll, zu laden. Wird nur auf manchen Laptops unterstützt.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`stop-charge-thresh-bat0`
Ab welchem Prozentwert die Batterie 0 aufhören soll, zu laden. Wird nur auf manchen Laptops unterstützt.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`start-charge-thresh-bat1`
Ab welchem Prozentwert die Batterie 1 anfangen soll, zu laden. Wird nur auf manchen Laptops unterstützt.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Nichtnegative-ganze-Zahl` [`tlp-configuration`-Parameter]
`stop-charge-thresh-bat1`
Ab welchem Prozentwert die Batterie 1 aufhören soll, zu laden. Wird nur auf manchen Laptops unterstützt.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Zeichenkette radeon-power-profile-on-ac` [`tlp-configuration`-Parameter]
Taktgeschwindigkeitsstufe („Clock Speed Level“) für Radeon-Grafik. Zur Auswahl stehen low (niedrig), mid (mittel), high (hoch), auto (automatisch), default (Voreinstellung).

Die Vorgabe ist '`"high"`'.

`Zeichenkette radeon-power-profile-on-bat` [`tlp-configuration`-Parameter]
Wie `radeon-power-ac`, aber für den BAT-Modus.

Die Vorgabe ist '`"low"`'.

`Zeichenkette radeon-dpm-state-on-ac` [`tlp-configuration`-Parameter]
Methode für die dynamische Energieverwaltung („Dynamic Power Management“, DPM) auf Radeon. Zur Auswahl stehen battery (Batterie), performance (Leistung).

Die Vorgabe ist '`"performance"`'.

`Zeichenkette radeon-dpm-state-on-bat` [`tlp-configuration`-Parameter]
Wie `radeon-dpm-state-ac`, aber für den BAT-Modus.

Die Vorgabe ist '`"battery"`'.

`Zeichenkette radeon-dpm-perf-level-on-ac` [`tlp-configuration`-Parameter]
Leistungsstufe („Performance Level“) des Radeon-DPM. Zur Auswahl stehen auto (automatisch), low (niedrig), high (hoch).

Die Voreinstellung ist '`"auto"`'.

`Zeichenkette radeon-dpm-perf-level-on-bat` [`tlp-configuration`-Parameter]
Wie `radeon-dpm-perf-ac`, aber für den BAT-Modus.

Die Voreinstellung ist '`"auto"`'.

`An-Aus-Boolescher-Ausdruck wifi-pwr-on-ac?` [`tlp-configuration`-Parameter]
WLAN-Stromsparmodus.

Vorgegeben ist '`#f`'.

`An-Aus-Boolescher-Ausdruck wifi-pwr-on-bat?` [`tlp-configuration`-Parameter]
Wie `wifi-power-ac?`, aber für den BAT-Modus.

Die Vorgabe ist '`#t`'.

`Ja-Nein-Boolescher-Ausdruck wol-disable?` [`tlp-configuration`-Parameter]
Rechnerstart nach Netzwerkanforderung („Wake on LAN“) deaktivieren.

Die Vorgabe ist '`#t`'.

`Nichtnegative-ganze-Zahl sound-power-save-on-ac` [`tlp-configuration`-Parameter]
Nach wie vielen Sekunden der Stromsparmodus für die Audioverarbeitung auf Intel-HDA- und AC97-Geräten aktiviert wird. Ein Wert von 0 deaktiviert den Stromsparmodus.

Die Vorgabe ist '`0`'.

`Nichtnegative-ganze-Zahl sound-power-save-on-bat` [`tlp-configuration`-Parameter]
Wie `sound-powersave-ac`, aber für den BAT-Modus.

Die Vorgabe ist '`1`'.

`Ja-Nein-Boolescher-Ausdruck sound-power-save-controller?` [`tlp-configuration`-Parameter]
Steuerungseinheit („Controller“) im Stromsparmodus auf Intel-HDA-Geräten deaktivieren.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck bay-poweroff-on-bat?` [`tlp-configuration`-Parameter]
Optisches Laufwerk in einer UltraBay/MediaBay im BAT-Modus aktivieren. Laufwerke können erneut gestartet werden, indem Sie den Hebel zum Auswerfen lösen (und wieder einsetzen) oder, auf neueren Modellen, indem Sie den Knopf zum Auswerfen des eingelegten Datenträgers drücken.

Vorgegeben ist '`#f`'.

`Zeichenkette bay-device` [`tlp-configuration`-Parameter]
Name des Geräts für das optische Laufwerk, das gestartet werden soll.

Die Vorgabe ist '`"sr0"`'.

`Zeichenkette runtime-pm-on-ac` [`tlp-configuration`-Parameter]
Laufzeitenergieverwaltung („Runtime Power Management“) von PCI(e)-Bus-Geräten. Zur Auswahl stehen on (angeschaltet) und auto (automatisch).

Die Vorgabe ist ‘`"on"`’.

`Zeichenkette runtime-pm-on-bat` [`tlp-configuration`-Parameter]
Wie `runtime-pm-ac`, aber für den BAT-Modus.

Die Voreinstellung ist ‘`"auto"`’.

`Boolescher-Ausdruck runtime-pm-all?` [`tlp-configuration`-Parameter]
Runtime Power Management for all PCI(e) bus devices, except blacklisted ones.

Die Vorgabe ist ‘`#t`’.

`Vielleicht-Leerzeichengetrennte-Zeichenketten-Liste runtime-pm-blacklist` [`tlp-configuration`-Parameter]
Die angegebenen PCI(e)-Geräteadressen von der Laufzeitenergieverwaltung („Runtime Power Management“) ausnehmen.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Leerzeichengetrennte-Zeichenketten-Liste runtime-pm-driver-blacklist` [`tlp-configuration`-Parameter]
PCI(e)-Geräte von der Laufzeitenergieverwaltung („Runtime Power Management“) ausnehmen, wenn sie den angegebenen Treibern zugeordnet sind.

`Boolescher-Ausdruck usb-autosuspend?` [`tlp-configuration`-Parameter]
USB-Geräte automatisch in den Ruhezustand versetzen („USB-Autosuspend“).

Die Vorgabe ist ‘`#t`’.

`Vielleicht-Zeichenkette usb-blacklist` [`tlp-configuration`-Parameter]
Die angegebenen Geräte vom USB-Autosuspend ausnehmen.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Boolescher-Ausdruck usb-blacklist-wwan?` [`tlp-configuration`-Parameter]
WWAN-Geräte vom USB-Autosuspend ausnehmen.

Die Vorgabe ist ‘`#t`’.

`Vielleicht-Zeichenkette usb-whitelist` [`tlp-configuration`-Parameter]
Für die angegebenen Geräte USB-Autosuspend aktivieren, selbst wenn Autosuspend durch den Treiber oder wegen `usb-blacklist-wwan?` deaktiviert werden würde.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Vielleicht-Boolescher-Ausdruck usb-autosuspend-disable-on-shutdown?` [`tlp-configuration`-Parameter]
USB-Autosuspend vor dem Herunterfahren aktivieren.

Der Vorgabewert ist ‘`disabled`’ (d.h. deaktiviert).

`Boolescher-Ausdruck` [`tlp-configuration`-Parameter]
`restore-device-state-on-startup?`
Zustand von funkfähigen Geräten (Bluetooth, WLAN, WWAN) vom letzten Herunterfahren beim Hochfahren des Systems wiederherstellen.

Vorgegeben ist '`#f`'.

Thermald-Daemon

Das Modul `(gnu services pm)` stellt eine Schnittstelle zu Thermald zur Verfügung, einem Dienst zur CPU-Frequenzskalierung („CPU Frequency Scaling"), mit dem Überhitzung verhindert wird.

`thermald-service-type` [Scheme-Variable]
Dies ist der Diensttyp für Thermald (`https://01.org/linux-thermal-daemon/`), den *Linux Thermal Daemon*, der für die Hitzeregulierung von Prozessoren zuständig ist. Er ändert deren thermischen Zustand („Thermal State") und verhindert, dass sie überhitzen.

`thermald-configuration` [Datentyp]
Datentyp, der die Konfiguration des `thermald-service-type` repräsentiert.

`adaptive?` (Vorgabe: `#f`)
Die sich anpassenden Tabellen aus DPTF (Dynamic Power and Thermal Framework) benutzen, falls vorhanden.

`ignore-cpuid-check?` (Vorgabe: `#f`)
Ergebnis der Prüfung per CPUID auf unterstützte Prozessormodelle ignorieren.

`thermald` (Vorgabe: *thermald*)
Paketobjekt von thermald.

12.9.28 Audio-Dienste

Das Modul `(gnu services audio)` stellt einen Dienst zur Verfügung, um MPD (den Music Player Daemon) zu starten.

Music Player Daemon

Der Music Player Daemon (MPD) ist ein Dienst, der Musik abspielen kann und der dabei vom lokalen Rechner oder über das Netzwerk durch verschiedene Clients angesteuert werden kann.

Das folgende Beispiel zeigt, wie man `mpd` als Benutzer `"bob"` auf Port `6666` ausführen könnte. Dabei wird Pulseaudio zur Ausgabe verwendet.

```
(service mpd-service-type
         (mpd-configuration
          (user "bob")
          (port "6666")))
```

`mpd-service-type` [Scheme-Variable]
Der Diensttyp für `mpd`.

`mpd-configuration` [Datentyp]

Datentyp, der die Konfiguration von `mpd` repräsentiert.

`user` (Vorgabe: `"mpd"`)
: Das Benutzerkonto, mit dem mpd ausgeführt wird.

`music-dir` (Vorgabe: `"~/Music"`)
: Das Verzeichis, in dem nach Musikdateien gesucht wird.

`playlist-dir` (Vorgabe: `"~/.mpd/playlists"`)
: Das Verzeichnis, um Wiedergabelisten („Playlists“) zu speichern.

`db-file` (Vorgabe: `"~/.mpd/tag_cache"`)
: Der Ort, an dem die Musikdatenbank gespeichert wird.

`state-file` (Vorgabe: `"~/.mpd/state"`)
: Der Ort, an dem die Datei mit dem aktuellen Zustand von MPD gespeichert wird.

`sticker-file` (Vorgabe: `"~/.mpd/sticker.sql"`)
: Der Ort, an dem die Sticker-Datenbank gespeichert wird.

`port` (Vorgabe: `"6600"`)
: Der Port, auf dem mpd ausgeführt wird.

`address` (Vorgabe: `"any"`)
: Die Adresse, an die sich mpd binden wird. Um einen Unix-Socket zu benutzen, kann hier ein absoluter Pfad angegeben werden.

`outputs` (Vorgabe: `"(list (mpd-output))"`)
: Welche Tonausgaben MPD benutzen kann. Vorgegeben ist eine einzelne Ausgabe, die Pulseaudio benutzt.

`mpd-output` [Datentyp]

Datentyp, der eine Tonausgabe von `mpd` repräsentiert.

`name` (Vorgabe: `"MPD"`)
: Der Name der Tonausgabe.

`type` (Vorgabe: `"pulse"`)
: Der Typ der Tonausgabe.

`enabled?` (Vorgabe: `#t`)
: Gibt an, ob diese Tonausgabe aktiviert sein soll, wenn MPD gestartet wird. Vorgegeben ist, alle Tonausgaben zu aktivieren. Das entspricht der Voreinstellung, wenn keine Zustandsdatei existiert; mit Zustandsdatei wird der Zustand von früher wiederhergestellt.

`tags?` (Vorgabe: `#t`)
: Wenn es auf `#f` steht, sendet MPD keine Tags an diese Ausgabe. Dies wird nur berücksichtigt, wenn das Ausgabe-Plugin Tags empfangen kann, wie beim `httpd`-Ausgabe-Plugin.

`always-on?` (Vorgabe: `#f`)
: Wenn es auf `#t` steht, versucht MPD, diese Tonausgabe immer offen zu lassen. Das kann bei Streaming-Servern helfen, wo man *nicht* will, dass

die Verbindung zu allen Zuhörern abbricht, nur weil das Abspielen aus irgendeinem Grund angehalten wurde.

`mixer-type`
: Für dieses Feld wird ein Symbol akzeptiert, das das für diese Tonausgabe zu benutzende Mischpult („Mixer“) bezeichnet. Zur Wahl stehen das `hardware`-Mischpult, das `software`-Mischpult, das `null`-Mischpult oder kein Mischpult (`none`). Mit dem `null`-Mischpult kann die Lautstärke eingestellt werden, aber ohne Auswirkung; das kann als Trick benutzt werden, um ein externes Mischpult zu implementieren.

`extra-options` (Vorgabe: `'()`)
: Eine assoziative Liste, die Optionssymbole auf Zeichenketten abbildet. Sie wird an die Tonausgabenkonfiguration angehängt.

Das folgende Beispiel zeigt eine Konfiguration von `mpd`, die eine HTTP-Audiostreaming-Tonausgabe anbietet.

```
(service mpd-service-type
         (mpd-configuration
           (outputs
             (list (mpd-output
                     (name "streaming")
                     (type "httpd")
                     (mixer-type 'null)
                     (extra-options
                      `((encoder . "vorbis")
                        (port    . "8080"))))))))
```

12.9.29 Virtualisierungsdienste

Das Modul `(gnu services virtualization)` bietet Dienste für die Daemons von libvirt und virtlog, sowie andere virtualisierungsbezogene Dienste.

Libvirt-Daemon

`libvirtd` ist die serverseitige Daemon-Komponente des libvirt-Systems zur Virtualisierungsverwaltung. Dieser Daemon läuft auf als Wirt dienenden Servern und führt anfallende Verwaltungsaufgaben für virtualisierte Gäste durch.

`libvirt-service-type` [Scheme-Variable]
: Dies ist der Diensttyp des libvirt-Daemons (`https://libvirt.org`). Sein Wert muss ein `libvirt-configuration`-Verbundsobjekt sein.

```
(service libvirt-service-type
         (libvirt-configuration
          (unix-sock-group "libvirt")
          (tls-port "16555")))
```

Verfügbare `libvirt-configuration`-Felder sind:

„`package`“ `libvirt` [`libvirt-configuration`-Parameter]
: Libvirt-Paket.

`Boolescher-Ausdruck listen-tls?` [`libvirt-configuration`-Parameter]
Option zum Lauschen auf sichere TLS-Verbindungen über den öffentlichen TCP/IP-Port. `listen` muss gesetzt sein, damit dies eine Wirkung hat.

Bevor Sie diese Funktionalität nutzen können, muss eine Zertifikatsautorität eingerichtet worden sein und Server-Zertifikate ausgestellt worden sein.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck listen-tcp?` [`libvirt-configuration`-Parameter]
Auf unverschlüsselte TCP-Verbindungen auf dem öffentlichen TCP/IP-Port lauschen. `listen` muss gesetzt sein, damit dies eine Wirkung hat.

Nach Voreinstellung kann auf dem TCP-Socket nur gelauscht werden, wenn SASL-Authentifizierung möglich ist. Nur solche SASL-Mechanismen, die Datenverschlüsselung unterstützen, sind zugelassen. Das sind DIGEST_MD5 und GSSAPI (Kerberos5).

Vorgegeben ist '`#f`'.

`Zeichenkette tls-port` [`libvirt-configuration`-Parameter]
Der Port, um sichere TLS-Verbindungen zu akzeptieren. Dies kann eine Portnummer oder ein Dienstname sein.

Die Vorgabe ist '`"16514"`'.

`Zeichenkette tcp-port` [`libvirt-configuration`-Parameter]
Der Port, um unsichere TCP-Verbindungen zu akzeptieren. Dies kann eine Portnummer oder ein Dienstname sein.

Die Vorgabe ist '`"16509"`'.

`Zeichenkette listen-addr` [`libvirt-configuration`-Parameter]
IP-Adresse oder Rechnername („Hostname“), der für von Clients ausgehende Verbindungen benutzt wird.

Die Vorgabe ist '`"0.0.0.0"`'.

`Boolescher-Ausdruck mdns-adv?` [`libvirt-configuration`-Parameter]
Einstellung, ob der libvirt-Dienst mDNS-Mitteilungen sendet.

Dies kann alternativ für alle Dienste auf einem Rechner deaktiviert werden, indem man den Avahi-Daemon anhält.

Vorgegeben ist '`#f`'.

`Zeichenkette mdns-name` [`libvirt-configuration`-Parameter]
Der voreingestellte Name in mDNS-Mitteilungen. Er muss auf dem direkten Broadcast-Netzwerk eindeutig sein.

Die Vorgabe ist '`"Virtualization Host <Rechnername>"`'.

`Zeichenkette unix-sock-group` [`libvirt-configuration`-Parameter]
Besitzergruppe des UNIX-Sockets. Diese Einstellung kann benutzt werden, um einer als vertrauenswürdig geltenden Gruppe von Benutzern Zugriff auf Verwaltungsfunktionen zu gewähren, ohne dass diese als Administratornutzer root ausgeführt werden müssen.

Die Vorgabe ist '`"root"`'.

`Zeichenkette unix-sock-ro-perms` [`libvirt-configuration`-Parameter]
UNIX-Socket-Berechtigungen für den Socket nur mit Lesezugriff („read only“). Dies wird nur zur Überwachung des Zustands der VM benutzt.

Die Vorgabe ist ‘`"0777"`’.

`Zeichenkette unix-sock-rw-perms` [`libvirt-configuration`-Parameter]
UNIX-Socket-Berechtigungen für den Socket mit Schreib- und Lesezugriff („read/write“). Nach Vorgabe kann nur der Administratornutzer root zugreifen. Wenn auf dem Socket PolicyKit aktiviert ist, wird die Vorgabe geändert, dass jeder zugreifen kann (z.B. zu 0777)

Die Vorgabe ist ‘`"0770"`’.

`Zeichenkette unix-sock-admin-perms` [`libvirt-configuration`-Parameter]
UNIX-Socket-Berechtigungen für den Administrator-Socket. Nach Vorgabg hat nur der Besitzer (der Administratornutzer root) hierauf Zugriff; ändern Sie es nur, wenn Sie sicher wissen, wer dann alles Zugriff bekommt.

Die Vorgabe ist ‘`"0777"`’.

`Zeichenkette unix-sock-dir` [`libvirt-configuration`-Parameter]
Das Verzeichnis, in dem Sockets gefunden werden können bzw. erstellt werden.

Die Vorgabe ist ‘`"/var/run/libvirt"`’.

`Zeichenkette auth-unix-ro` [`libvirt-configuration`-Parameter]
Authentifizierungsschema für nur lesbare UNIX-Sockets. Nach Vorgabe gestatten es die Socket-Berechtigungen jedem Nutzer, sich zu verbinden.

Die Vorgabe ist ‘`"polkit"`’.

`Zeichenkette auth-unix-rw` [`libvirt-configuration`-Parameter]
Authentifizierungsschema für UNIX-Sockets mit Schreib- und Lesezugriff. Nach Vorgabe erlauben die Socket-Berechtigungen nur dem Administratornutzer root Zugriff. Wenn libvirt mit Unterstützung für PolicyKit kompiliert wurde, ist die Vorgabe, Authentifizierung über „polkit“ durchzuführen.

Die Vorgabe ist ‘`"polkit"`’.

`Zeichenkette auth-tcp` [`libvirt-configuration`-Parameter]
Authentifizierungsschema für TCP-Sockets. Wenn Sie SASL nicht aktivieren, dann wird alle TCP-Kommunikation im Klartext verschickt. Tun Sie dies *nicht*, außer Sie benutzen libvirt nur als Entwickler oder zum Testen.

Die Vorgabe ist ‘`"sasl"`’.

`Zeichenkette auth-tls` [`libvirt-configuration`-Parameter]
Authentifizierungsschema für TLS-Sockets. Für TLS-Sockets wird bereits durch die TLS-Schicht Verschlüsselung bereitgestellt und eingeschränkte Authentifizierung wird über Zertifikate durchgeführt.

Es ist möglich, auch hier den SASL-Authentifizierungsmechanismus anzuwenden, indem Sie für diese Option „sasl“ eintragen.

Die Vorgabe ist ‘`"none"`’, d.h. keine zusätzliche Authentifizierung.

`Optional-nichtleere-Liste` [`libvirt-configuration`-Parameter]
`access-drivers`

Welche Schemata zur Zugriffskontrolle auf Programmierschnittstellen (APIs) benutzt werden.

Nach Vorgabe kann ein authentifizierter Nutzer auf alle Programmierschnittstellen zugreifen. Zugriffstreiber können dies einschränken.

Die Vorgabe ist '`()`'.

`Zeichenkette key-file` [`libvirt-configuration`-Parameter]

Pfad zur Schlüsseldatei für den Server. Wenn er auf eine leere Zeichenkette gesetzt ist, dann wird kein privater Schlüssel geladen.

Die Vorgabe ist '`""`'.

`Zeichenkette cert-file` [`libvirt-configuration`-Parameter]

Pfad zur Zertifikatsdatei für den Server. Wenn er auf eine leere Zeichenkette gesetzt ist, dann wird kein Zertifikat geladen.

Die Vorgabe ist '`""`'.

`Zeichenkette ca-file` [`libvirt-configuration`-Parameter]

Pfad zur Datei mit dem Zertifikat der Zertifikatsautorität. Wenn er auf eine leere Zeichenkette gesetzt ist, dann wird kein Zertifikat der Zertifikatsautorität geladen.

Die Vorgabe ist '`""`'.

`Zeichenkette crl-file` [`libvirt-configuration`-Parameter]

Pfad zur Zertifikatssperrliste („Certificate Revocation List“). Wenn er auf eine leere Zeichenkette gesetzt ist, dann wird keine Zertifikatssperrliste geladen.

Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck` [`libvirt-configuration`-Parameter]
`tls-no-sanity-cert`

Keine Überprüfung unseres eigenen Serverzertifikats durchführen.

Beim Start vom libvirtd prüft dieser, ob bei seinem eigenen Zertifikat alles in Ordnung ist.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck` [`libvirt-configuration`-Parameter]
`tls-no-verify-cert`

Keine Überprüfung von Clientzertifikaten durchführen.

Die Überprüfung des Zertifikats eines Clients ist der primäre Authentifizierungsmechanismus. Jeder Client, der kein von der Zertifikatsautorität signiertes Zertifikat vorweist, wird abgelehnt.

Vorgegeben ist '`#f`'.

`Optional-nichtleere-Liste` [`libvirt-configuration`-Parameter]
`tls-allowed-dn-list`

Liste der erlaubten Einträge für den „Distinguished Name“ bei x509.

Die Vorgabe ist '`()`'.

`Optional-nichtleere-Liste sasl-allowed-usernames` [`libvirt-configuration`-Parameter]
: Liste der erlaubten Einträge für SASL-Benutzernamen. Wie Benutzernamen aussehen müssen, ist abhängig vom jeweiligen SASL-Mechanismus.

  Die Vorgabe ist '`()`'.

`Zeichenkette tls-priority` [`libvirt-configuration`-Parameter]
: Dies wird vorrangig statt der beim Kompilieren voreingestellten TLS-Prioritätszeichenkette verwendet. Die Voreinstellung ist in der Regel '`"NORMAL"`', solange dies nicht bei der Erstellung geändert wurde. Ändern Sie dies nur, wenn die Einstellungen für libvirt von den globalen Voreinstellungen abweichen sollen.

  Die Vorgabe '`"NORMAL"`'.

`Ganze-Zahl max-clients` [`libvirt-configuration`-Parameter]
: Maximalzahl gleichzeitiger Client-Verbindungen, die für alle Sockets zusammen zugelassen werden sollen.

  Die Vorgabe ist '`5000`'.

`Ganze-Zahl max-queued-clients` [`libvirt-configuration`-Parameter]
: Maximale Länge der Warteschlange für Verbindungen, die darauf warten, vom Daemon angenommen zu werden. Beachten Sie, dass sich manche Protokolle, die Neuübertragung unterstützen, danach richten könnten, damit ein erneuter Verbindungsversuch angenommen wird.

  Die Vorgabe ist '`1000`'.

`Ganze-Zahl max-anonymous-clients` [`libvirt-configuration`-Parameter]
: Maximale Länge der Warteschlange für Clients, die angenommen wurden, aber noch nicht authentifiziert wurden. Setzen Sie dies auf null, um diese Funktionalität abzuschalten.

  Die Vorgabe ist '`20`'.

`Ganze-Zahl min-workers` [`libvirt-configuration`-Parameter]
: Anzahl an Arbeiter-Threads, die am Anfang gestartet werden sollen.

  Die Vorgabe ist '`5`'.

`Ganze-Zahl max-workers` [`libvirt-configuration`-Parameter]
: Maximale Anzahl an Arbeiter-Threads.

  Wenn die Anzahl aktiver Clients die `min-workers` übersteigt, werden weitere Threads erzeugt, bis die max_workers-Beschränkung erreicht wurde. Typischerweise würden Sie für max_workers die maximale Anzahl zugelassener Clients angeben.

  Die Vorgabe ist '`20`'.

`Ganze-Zahl prio-workers` [`libvirt-configuration`-Parameter]
: Die Anzahl priorisierter Arbeiter-Threads. Wenn alle Arbeiter aus diesem Pool festhängen, können manche, mit hoher Priorität versehene Aufrufe (speziell domainDestroy) in diesem Pool hier ausgeführt werden.

  Die Vorgabe ist '`5`'.

`Ganze-Zahl max-requests` [`libvirt-configuration`-Parameter]
Wie viele nebenläufige RPC-Aufrufe global ausgeführt werden können.

Die Vorgabe ist '`20`'.

`Ganze-Zahl max-client-requests` [`libvirt-configuration`-Parameter]
Wie viele nebenläufige Anfragen von einer einzelnen Client-Verbindung ausgehen können. Um zu verhindern, dass ein einzelner Client den gesamten Server für sich beansprucht, sollte der Wert hier nur einen kleinen Teil der globalen max_requests- und max_workers-Parameter ausmachen.

Die Vorgabe ist '`5`'.

`Ganze-Zahl admin-min-workers` [`libvirt-configuration`-Parameter]
Wie bei `min-workers`, aber für die Administratorschnittstelle.

Die Vorgabe ist '`1`'.

`Ganze-Zahl admin-max-workers` [`libvirt-configuration`-Parameter]
Wie bei `max-workers`, aber für die Administratorschnittstelle.

Die Vorgabe ist '`5`'.

`Ganze-Zahl admin-max-clients` [`libvirt-configuration`-Parameter]
Wie bei `max-clients`, aber für die Administratorschnittstelle.

Die Vorgabe ist '`5`'.

`Ganze-Zahl admin-max-queued-clients` [`libvirt-configuration`-Parameter]
Wie bei `max-queued-clients`, aber für die Administratorschnittstelle.

Die Vorgabe ist '`5`'.

`Ganze-Zahl admin-max-client-requests` [`libvirt-configuration`-Parameter]
Wie bei `max-client-requests`, aber für die Administratorschnittstelle.

Die Vorgabe ist '`5`'.

`Ganze-Zahl log-level` [`libvirt-configuration`-Parameter]
Protokollstufe. 4 für Fehler, 3 für Warnungen, 2 für Informationen, 1 zur Fehlersuche.

Die Vorgabe ist '`3`'.

`Zeichenkette log-filters` [`libvirt-configuration`-Parameter]
Protokollfilter.

Ein Filter ermöglicht es, für eine bestimmte Kategorie von Protokollen eine andere Protokollierungsstufe festzulegen. Filter müssen eines der folgenden Formate haben:

- x:Name
- x:+Name

wobei `Name` eine Zeichenkette ist, die zu einer in der Umgebungsvariablen `VIR_LOG_INIT()` am Anfang jeder Quelldatei von libvirt angegebenen Kategorie passen muss, z.B. '`"remote"`', '`"qemu"`' oder '`"util.json"`' (der Name im Filter kann auch nur ein Teil des vollständigen Kategoriennamens sein, wodurch mehrere, ähnliche passende Kategoriennamen möglich sind). Das optionale Präfix „`+`“ bedeutet, dass libvirt

eine Rückverfolgung (d.h. ein „Stack Trace“) für jede zum Namen passende Nachricht ins Protokoll schreiben soll. `x` benennt jeweils die kleinste Stufe, deren passende Nachrichten protokolliert werden sollen.

- 1: Fehlersuche („DEBUG“)
- 2: Informationen („INFO“)
- 3: Warnungen („WARNING“)
- 4: Fehler („ERROR“)

Mehrere Filter können in einer einzelnen Filteranweisung definiert werden; sie müssen nur durch Leerzeichen voneinander getrennt werden.

Die Vorgabe ist ‘`"3:remote 4:event"`’.

`Zeichenkette log-outputs` [`libvirt-configuration`-Parameter]

Ausgaben für die Protokollierung.

Eine Ausgabe ist einer der Orte, wohin Informationen aus der Protokollierung gespeichert werden. Eine Ausgabe kann auf eine der folgenden Arten angegeben werden:

`x:stderr` Protokolle werden auf der Standardausgabe („Stderr“) ausgegeben.

`x:syslog:Name`
: Syslog wird zur Ausgabe benutzt. Der Name dient dabei als Identifikator für libvirt-Protokolle.

`x:file:Dateipfad`
: Protokolle werden in die Datei unter dem angegebenen Dateipfad ausgegeben.

`x:journald`
: Die Ausgabe läuft über das journald-Protokollsystem.

In allen Fällen steht das x vorne für die kleinste Stufe und wirkt als Filter.

- 1: Fehlersuche („DEBUG“)
- 2: Informationen („INFO“)
- 3: Warnungen („WARNING“)
- 4: Fehler („ERROR“)

Mehrere Ausgaben können definiert werden, dazu müssen sie nur durch Leerzeichen getrennt hier angegeben werden.

Die Vorgabe ist ‘`"3:stderr"`’.

`Ganze-Zahl audit-level` [`libvirt-configuration`-Parameter]

Ermöglicht Anpassungen am Auditierungs-Subsystem.

- 0: Jegliche Auditierung deaktivieren.
- 1: Auditierung nur aktivieren, wenn sie beim Wirtssystem aktiviert ist.
- 2: Auditierung aktivieren. Beenden, wenn das Wirtssystem Auditierung deaktiviert hat.

Die Vorgabe ist ‘`1`’.

`Boolescher-Ausdruck audit-logging` [`libvirt-configuration`-Parameter]
Audit-Nachrichten über die Protokollinfrastruktur von libvirt versenden.

Vorgegeben ist '`#f`'.

`Optional-nichtleere-Zeichenkette host-uuid` [`libvirt-configuration`-Parameter]
Für das Wirtssystem zu verwendende UUID. Bei der UUID dürfen nicht alle Ziffern gleich sein.

Die Vorgabe ist '`""`'.

`Zeichenkette host-uuid-source` [`libvirt-configuration`-Parameter]
Die Quelle, von der die UUID des Wirtssystems genommen wird.

- `smbios`: Die UUID von `dmidecode -s system-uuid` holen.
- `machine-id`: Die UUID aus `/etc/machine-id` holen.

Falls `dmidecode` keine gültige UUID liefert, wird eine temporäre UUID generiert.

Die Vorgabe ist '`"smbios"`'.

`Ganze-Zahl keepalive-interval` [`libvirt-configuration`-Parameter]
Einem Client wird eine Nachricht zum Aufrechterhalten der Verbindung gesendet, nachdem `keepalive_interval` Sekunden lang keine Aktivität stattgefunden hat. Damit kann überprüft werden, ob der Client noch antwortet. Wird dieses Feld auf -1 gesetzt, wird libvirtd niemals Aufrechterhaltungsanfragen senden; Clients können diese aber weiterhin dem Daemon schicken und er wird auf diese antworten.

Die Vorgabe ist '`5`'.

`Ganze-Zahl keepalive-count` [`libvirt-configuration`-Parameter]
Wie viele Aufrechterhaltungsnachrichten höchstens zum Client geschickt werden dürfen, ohne dass eine Antwort zurückgekommen ist, bevor die Verbindung als abgebrochen gilt.

Mit anderen Worten wird die Verbindung ungefähr dann automatisch geschlossen, wenn `keepalive_interval * (keepalive_count + 1)` Sekunden seit der letzten vom Client empfangenen Nachricht vergangen sind. Wenn `keepalive-count` auf 0 gesetzt ist, werden Verbindungen dann automatisch geschlossen, wenn `keepalive-interval` Sekunden der Inaktivität vorausgegangen sind, ohne dass eine Aufrechterhaltungsnachricht versandt wurde.

Die Vorgabe ist '`5`'.

`Ganze-Zahl admin-keepalive-interval` [`libvirt-configuration`-Parameter]
Wie oben, aber für die Administratorschnittstelle.

Die Vorgabe ist '`5`'.

`Ganze-Zahl admin-keepalive-count` [`libvirt-configuration`-Parameter]
Wie oben, aber für die Administratorschnittstelle.

Die Vorgabe ist '`5`'.

`Ganze-Zahl ovs-timeout` [`libvirt-configuration`-Parameter]
Zeitbeschränkung für Aufrufe über Open vSwitch.

Das Werkzeug `ovs-vsctl` wird zur Konfiguration benutzt; die dort eingestellte Zeitbeschränkung ist nach Voreinstellung auf 5 Sekunden festgelegt, um zu verhindern, dass libvirt durch unbegrenztes Warten blockiert werden kann.

Die Vorgabe ist '`5`'.

Virtlog-Daemon

Der virtlogd-Dienst ist eine serverseitige Daemon-Komponente von libvirt, die benutzt wird, um Protokolle der Konsolen von virtuellen Maschinen zu verwalten.

Dieser Daemon wird von libvirt-Clientanwendungen nicht direkt benutzt, sondern wird an deren Stelle vom `libvirtd` aufgerufen. Indem die Protokolle in einem eigenständigen Daemon vorgehalten werden, kann der eigentliche `libvirtd`-Daemon neu gestartet werden, ohne dass man riskiert, Protokolle zu verlieren. Der `virtlogd`-Daemon hat die Fähigkeit, sich selbst erneut mit exec() zu starten, wenn er `SIGUSR1` empfängt, damit Aktualisierungen ohne Ausfall möglich sind.

`virtlog-service-type` [Scheme-Variable]
Dies ist der Diensttyp des virtlog-Daemons. Sein Wert muss eine `virtlog-configuration` sein.

```
(service virtlog-service-type
         (virtlog-configuration
          (max-clients 1000)))
```

`libvirt-Parameter „package“` *libvirt* [Variable]
Libvirt-Paket.

`Ganze-Zahl log-level` [`virtlog-configuration`-Parameter]
Protokollstufe. 4 für Fehler, 3 für Warnungen, 2 für Informationen, 1 zur Fehlersuche.

Die Vorgabe ist '`3`'.

`Zeichenkette log-filters` [`virtlog-configuration`-Parameter]
Protokollfilter.

Ein Filter ermöglicht es, für eine bestimmte Kategorie von Protokollen eine andere Protokollierungsstufe festzulegen. Filter müssen eines der folgenden Formate haben:

- x:Name
- x:+Name

wobei `Name` eine Zeichenkette ist, die zu einer in der Umgebungsvariablen `VIR_LOG_INIT()` am Anfang jeder Quelldatei von libvirt angegebenen Kategorie passen muss, z.B. „remote“, „qemu“ oder „util.json“ (der Name im Filter kann auch nur ein Teil des vollständigen Kategoriennamens sein, wodurch mehrere, ähnliche passende Kategoriennamen möglich sind). Das optionale Präfix „`+`“ bedeutet, dass libvirt eine Rückverfolgung (d.h. ein „Stack Trace“) für jede zum Namen passende Nachricht ins Protokoll schreiben soll. `x` benennt jeweils die kleinste Stufe, deren passende Nachrichten protokolliert werden sollen.

- 1: Fehlersuche („DEBUG“)

- 2: Informationen („INFO“)
- 3: Warnungen („WARNING“)
- 4: Fehler („ERROR“)

Mehrere Filter können in einer einzelnen Filteranweisung definiert werden; sie müssen nur durch Leerzeichen voneinander getrennt werden.

Die Vorgabe ist ‘`"3:remote 4:event"`’.

`Zeichenkette log-outputs` [`virtlog-configuration`-Parameter]

Ausgaben für die Protokollierung.

Als Ausgabe bezeichnen wir einen der Orte, an denen Protokollinformationen gespeichert werden. Eine Ausgabe wird auf eine der folgenden Arten angegeben:

`x:stderr` Protokolle werden auf der Standardausgabe („Stderr“) ausgegeben.

`x:syslog:Name`
: Syslog wird zur Ausgabe benutzt. Der Name dient dabei als Identifikator für libvirt-Protokolle.

`x:file:Dateipfad`
: Protokolle werden in die Datei unter dem angegebenen Dateipfad ausgegeben.

`x:journald`
: Die Ausgabe läuft über das journald-Protokollsystem.

In allen Fällen steht das x vorne für die kleinste Stufe und wirkt als Filter.

- 1: Fehlersuche („DEBUG“)
- 2: Informationen („INFO“)
- 3: Warnungen („WARNING“)
- 4: Fehler („ERROR“)

Mehrere Ausgaben können definiert werden, dazu müssen sie nur durch Leerzeichen getrennt hier angegeben werden.

Die Vorgabe ist ‘`"3:stderr"`’.

`Ganze-Zahl max-clients` [`virtlog-configuration`-Parameter]

Maximalzahl gleichzeitiger Client-Verbindungen, die für alle Sockets zusammen zugelassen werden sollen.

Die Vorgabe ist ‘`1024`’.

`Ganze-Zahl max-size` [`virtlog-configuration`-Parameter]

Wie groß eine Protokolldatei werden darf, bevor eine neue begonnen wird.

Die Vorgabe ist ‘`2MB`’.

`Ganze-Zahl max-backups` [`virtlog-configuration`-Parameter]

Wie viele Dateien mit Sicherungskopien gespeichert bleiben sollen.

Die Vorgabe ist ‘`3`’.

Transparente Emulation mit QEMU

Mit `qemu-binfmt-service-type` wird transparente Emulation von Programm-Binärdateien, die für unterschiedliche Architekturen erstellt wurden, ermöglicht. Z.B. können Sie ein ARMv7-Programm „einfach so“ transparent auf einer x86_64-Maschine ausführen. Dazu wird der QEMU-Emulator (`https://www.qemu.org`) mit der `binfmt_misc`-Funktionalität des Kernels Linux kombiniert. Mit dieser Funktionalität können Sie jedoch nur ein GNU/Linux-System, das auf einer anderen Architektur läuft, emulieren. Unterstützung für GNU/Hurd ist auch möglich, lesen Sie dazu unten weiter.

`qemu-binfmt-service-type` [Scheme-Variable]

Dies ist der Diensttyp des QEMU/binfmt-Dienstes für transparente Emulation. Sein Wert muss ein `qemu-binfmt-configuration`-Objekt sein, das das QEMU-Paket angibt, das benutzt werden soll, sowie die Architektur, die wir emulieren möchten.

```
(service qemu-binfmt-service-type
         (qemu-binfmt-configuration
           (platforms (lookup-qemu-platforms "arm" "aarch64"))))
```

In diesem Beispiel aktivieren wir transparente Emulation für die Plattformen ARM und aarch64. Wenn wir `herd stop qemu-binfmt` ausführen, wird diese abgeschaltet, und mit `herd start qemu-binfmt` wird sie wieder aktiv (siehe Abschnitt “Invoking herd” in *The GNU Shepherd Manual*).

`qemu-binfmt-configuration` [Datentyp]

Dies ist die Konfiguration des `qemu-binfmt`-Dienstes.

`platforms` (Vorgabe: `'()`)

Die Liste der emulierten QEMU-Plattformen. Jeder Eintrag muss ein *Plattformobjekt* sein, wie `lookup-qemu-platforms` eines zurückliefert (siehe unten).

Wenn wir zum Beispiel annehmen, Sie arbeiten auf einer x86_64-Maschine und haben diesen Dienst eingerichtet:

```
(service qemu-binfmt-service-type
         (qemu-binfmt-configuration
           (platforms (lookup-qemu-platforms "arm"))))
```

Dann können Sie das hier ausführen:

```
guix build -s armhf-linux inkscape
```

und alles verhält sich so, als würden Sie Inkscape für ARMv7 *wie „nativ“ auf einem ARM-Rechner erstellen*, wozu QEMU transparent benutzt wird, um den ARMv7-Prozessor zu emulieren. Das ist ganz schön praktisch, wenn Sie testen wollen, ob ein Paket für eine Architektur erstellt werden kann, die Ihnen nicht zur Verfügung steht.

`qemu` (Vorgabe: `qemu`)

Das QEMU-Paket, das benutzt werden soll.

`lookup-qemu-platforms` *`Plattformen...`* [Scheme-Prozedur]

Liefert die Liste der QEMU-Plattformobjekte, die den *Plattformen...* entsprechen. *Plattformen* muss eine Liste von Zeichenketten sein, die den Namen der Plattformen entsprechen, wie z.B. `"arm"`, `"sparc"`, `"mips64el"` und so weiter.

`qemu-platform?` *`Objekt`* [Scheme-Prozedur]
Liefert wahr, wenn das *Objekt* ein Plattformobjekt ist.

`qemu-platform-name` *`Plattform`* [Scheme-Prozedur]
Liefert den Namen der *Plattform*, also eine Zeichenkette wie z.B. `"arm"`.

QEMU Guest Agent

Mit dem QEMU Guest Agent kann das emulierte System vom Wirtssystem aus gesteuert werden. Der Dienst für den `qemu-guest-agent` führt den Agenten auf einem Guix-basierten Gastsystem aus. Um den Agenten vom Wirt aus zu steuern, öffnen Sie einen Socket, indem Sie QEMU mit folgenden Argumenten aufrufen:

```
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 \
...
```

Dadurch wird auf dem Wirtssystem ein Socket unter `/tmp/qga.sock` angelegt. Sobald der Guest Agent gestartet hat, können Sie mit `socat` Befehle erteilen:

```
$ guix shell socat -- socat unix-connect:/tmp/qga.sock stdio
{"execute": "guest-get-host-name"}
{"return": {"host-name": "guix"}}
```

Siehe die Dokumentation zum QEMU Guest Agent (`https://wiki.qemu.org/Features/GuestAgent`) für mehr Optionen und Befehle.

`qemu-guest-agent-service-type` [Scheme-Variable]
Diensttyp des Dienstes für den QEMU Guest Agent.

`qemu-guest-agent-configuration` [Datentyp]
Die Konfiguration des `qemu-guest-agent`-Dienstes.

`qemu` (Vorgabe: `qemu-minimal`)
Das QEMU-Paket, das benutzt werden soll.

`device` (Vorgabe: `""`)
Der Dateiname des Geräts bzw. Sockets, über das der Agent mit dem Wirtssystem kommuniziert. Wird nichts angegeben, verwendet QEMU einen voreingestellten Dateinamen.

GNU Hurd in einer virtuellen Maschine

Der `hurd-vm`-Dienst bietet Unterstützung, um GNU/Hurd in einer virtuellen Maschine (VM) laufen zu lassen, als eine sogenannte „Kind-Hurd“, englisch *Childhurd*. Der Dienst ist für die Nutzung mit GNU/Linux-Systemen gedacht; die angegebene GNU/Hurd-Konfiguration wird cross-kompiliert. Die virtuelle Maschine läuft als ein Shepherd-Dienst, der durch Namen wie `hurd-vm` und `childhurd` bezeichnet wird und mit Befehlen wie den folgenden gesteuert werden kann:

```
herd start hurd-vm
herd stop childhurd
```

Während der Dienst läuft, können Sie seine Konsole anzeigen lassen, indem Sie sich über einen VNC-Client mit ihm verbinden, z.B. durch den Befehl:

```
guix shell tigervnc-client -- vncviewer localhost:5900
```

Die Vorgabekonfiguration (siehe `hurd-vm-configuration` unten) lässt einen Server für Secure Shell (SSH) in Ihrem GNU/Hurd-System laufen, welchen QEMU (der Emulator für virtuelle Maschinen) an Port 10222 des Wirtssystems weitergibt. So können Sie sich mit der Childhurd über SSH verbinden:

```
ssh root@localhost -p 10022
```

Die Childhurd ist flüchtig und zustandslos: Jedes Mal, wenn Sie sie neu starten, hat sie wieder ihr ursprüngliches Wurzeldateisystem. Nur die Dateien unter `/etc/childhurd` im Wirtssystem werden nach Vorgabe in das Wurzeldateisystem der Childhurd übernommen, wenn sie startet. So können Sie einen Startwert für „Geheimnisse“ in der VM angeben: SSH-Rechnerschlüssel („Host Keys“), Autorisierungsschlüssel für Substitute und so weiter – siehe die Erklärung zu `secret-root` unten.

`hurd-vm-service-type` [Scheme-Variable]

Dies ist der Diensttyp zum Hurd-in-einer-virtuellen-Maschine-Dienst. Dessen Wert muss ein `hurd-vm-configuration`-Objekt sein, das das Betriebssystem festlegt (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258) und die Plattengröße für die virtuelle Hurd-Maschine, das zu nutzende QEMU-Paket sowie die Befehlszeilenoptionen, um sie auszuführen.

Zum Beispiel:

```
(service hurd-vm-service-type
         (hurd-vm-configuration
          (disk-size (* 5000 (expt 2 20))) ;5G
          (memory-size 1024)))             ;1024MiB
```

würde zur Erzeugung eines Disk-Images führen, dessen Größe für die Erstellung von GNU Hello ausreichend ist und das noch ein wenig zusätzlichen Arbeitsspeicher zur Verfügung stellt.

`hurd-vm-configuration` [Datentyp]

Der Datentyp, der die Konfiguration des `hurd-vm-service-type` repräsentiert.

`os` (Vorgabe: *%hurd-vm-operating-system*)
: Welches Betriebssystem instanziiert werden soll. Nach Vorgabe wird ein minimales „Bare-Bones“-System mit uneingeschränktem OpenSSH-Secure-Shell-Daemon, der auf Port 2222 lauscht (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314).

`qemu` (Vorgabe: `qemu-minimal`)
: Das QEMU-Paket, das benutzt werden soll.

`image` (Vorgabe: *hurd-vm-disk-image*)
: Die Prozedur, um das Disk-Image aus dieser Konfiguration zu erstellen.

`disk-size` (Vorgabe: `'guess`)
: Die Größe des Disk-Images.

`memory-size` (Vorgabe: `512`)
: Die Größe des Arbeitsspeichers der virtuellen Maschine in Mebibyte.

`options` (Vorgabe: `'("--snapshot")`)
: Weitere Befehlszeilenoptionen, mit denen QEMU ausgeführt wird.

`id` (Vorgabe: `#f`)
: Wenn das Feld gesetzt ist, muss es eine positive ganze Zahl größer null sein, womit Childhurd-Instanzen parametrisiert werden. Es wird an den Dienstnamen angehängt, z.B. `childhurd1`.

`net-options` (Vorgabe: *hurd-vm-net-options*)
: Die Prozedur, um die Liste der QEMU-Netzwerkoptionen zu erzeugen.

  Nach Vorgabe liefert sie

```
'("--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"))
```

  mit Portweiterleitungen:

```
secrets-port: (+ 11004 (* 1000 ID))
ssh-port: (+ 10022 (* 1000 ID))
vnc-port: (+ 15900 (* 1000 ID))
```

`secret-root` (Vorgabe: `/etc/childhurd`)
: Das Wurzelverzeichnis von Geheimnissen, die auf der Childhurd „out of band“, d.h. abseits vom Zuständigkeitsbereich der Childhurd, installiert werden sollen, sobald sie gestartet wurde. Childhurds sind flüchtig, d.h. bei jedem Start würden Geheimnisse wie die SSH-Rechnerschlüssel und Signierschlüssel von Guix neu angelegt.

  Wenn das Verzeichnis `/etc/childhurd` *nicht* existiert, wird dem im Childhurd laufenden `secret-service`-Dienst eine leere Liste von Geheimnissen geschickt.

  Üblicherweise wird er benutzt, um in `"/etc/childhurd"` einen Dateibaum aus *nicht* flüchtigen Geheimnissen einzufügen, wenn diese noch *nicht* existieren:

```
/etc/childhurd/etc/guix/acl
/etc/childhurd/etc/guix/signing-key.pub
/etc/childhurd/etc/guix/signing-key.sec
/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
```

  Diese Dateien werden automatisch an die Gast-Hurd-VM geschickt, wenn sie bootet, einschließlich der Dateiberechtigungen.

Wenn diese Dateien platziert wurden, fehlen nur noch wenige Dinge, damit das Wirtssystem `i586-gnu`-Erstellungen auf die Childhurd auslagern kann:

1. Der Schlüssel der Childhurd muss auf dem Wirtssystem authentifiziert werden, damit das Wirtssystem von der Childhurd stammende Erstellungsergebnisse annimmt. Das geht so:

   ```
   guix archive --authorize < \
     /etc/childhurd/etc/guix/signing-key.pub
   ```

2. Die Childhurd muss zu `/etc/guix/machines.scm` hinzugefügt werden (siehe Abschnitt 2.4.2 [Auslagern des Daemons einrichten], Seite 13).

Wir arbeiten daran, dass das automatisch passiert. Reden Sie mit uns unter `guix-devel@gnu.org`, wenn Sie dabei mitdiskutieren möchten!

Beachten Sie, dass nach Vorgabe ein flüchtiges Abbild für die virtuelle Maschine benutzt wird, also dessen Inhalt verloren geht, sobald sie gestoppt wurde. Wenn Sie stattdessen ein zustandsbehaftetes Abbild möchten, ändern Sie die Felder `image` und `options` in der Konfiguration, so dass keine `--snapshot`-Befehlszeilenoption übergeben wird, z.B. so:

```
(service hurd-vm-service-type
         (hurd-vm-configuration
          (image   (const "/nicht/im/store/mit/schreibzugriff/hurd.img"))
          (options '())))
```

Ganeti

Anmerkung: Dieser Dienst gilt als experimentell. Die Konfigurationsoptionen könnten in Zukunft noch auf eine Weise abgeändert werden, die keine Kompatibilität mit dem momentanen Verhalten gewährleistet, und nicht alle Funktionalitäten wurden gründlich getestet. Wenn Sie ihn benutzen, ermutigen wir Sie, Ihre Erfahrungen mit uns über `guix-devel@gnu.org` zu teilen.

Ganeti ist ein System zur Verwaltung virtueller Maschinen. Es ist dafür gedacht, virtuelle Maschinen auf einem Server-Cluster am Laufen zu halten, selbst wenn Hardware ausfällt, und deren Wartung und Wiederherstellung einfach zu machen. Es besteht aus mehreren Diensten, die später in diesem Abschnitt beschrieben werden. Zusätzlich zum Ganeti-Dienst brauchen Sie den OpenSSH-Dienst (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314) und müssen in die Datei `/etc/hosts` (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258) den Namen des Clusters mit Adresse eintragen lassen (oder Sie verwenden einen eigenen DNS-Server).

Alle Knoten, die an einem Ganeti-Cluster teilnehmen, sollten dieselbe Konfiguration von Ganeti und `/etc/hosts` aufweisen. Hier ist eine Beispielkonfiguration für einen Ganeti-Clusterknoten, der mehrere Hintergrundsysteme für Speichergeräte unterstützt und die *Betriebssystemanbieter* („OS providers“) `debootstrap` sowie `guix` installiert hat:

```
(use-package-modules virtualization)
(use-service-modules base ganeti networking ssh)
(operating-system
  ;; ...
```

```
  (host-name "node1")
  (hosts-file (plain-file "hosts" (format #f "
127.0.0.1       localhost
::1             localhost

192.168.1.200   ganeti.example.com
192.168.1.201   node1.example.com node1
192.168.1.202   node2.example.com node2
")))

  ;; QEMU installieren, damit wir auf KVM aufsetzende Instanzen benutzen
  ;; können, sowie LVM, DRBD und Ceph, damit wir die Speicher-Hintergrundsysteme
  ;; "plain", "drbd" und "rbd" benutzen können.
  (packages (append (map specification->package
                         '("qemu" "lvm2" "drbd-utils" "ceph"
                           ;; Betriebssystemanbieter debootstrap und guix:
                           "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 benutzt SSH für die Kommunikation zwischen Knoten.
                 (service openssh-service-type
                          (openssh-configuration
                           (permit-root-login 'prohibit-password)))

                 (service ganeti-service-type
                          (ganeti-configuration
                           ;; Diese Liste führt die Pfade im Dateisystem auf,
                           ;; wo Abbilder virtueller Maschinen gespeichert
                           ;; werden.
                           (file-storage-paths '("/srv/ganeti/file-storage"))
                           ;; In dieser Variablen konfigurieren wir eine einzige
                           ;; „Variante“ jeweils für sowohl Debootstrap als auch
                           ;; Guix, die KVM benutzt.
                           (os %default-ganeti-os))))
```

```
                %base-services)))
```

Benutzern wird geraten, die Anleitung für Ganeti-Administratoren (`https://docs.ganeti.org/docs/ganeti/3.0/html/admin.html`) zu lesen, um die verschiedenen Optionen für Ganeti-Cluster und die häufigen Operationen zu erlernen. Es gibt auch einen Blogeintrag (`https://guix.gnu.org/blog/2020/running-a-ganeti-cluster-on-guix/`), der beschreibt, wie man einen kleinen Cluster konfiguriert und initialisiert.

`ganeti-service-type` [Scheme-Variable]

Dieser Diensttyp umfasst all die verschiedenen Dienste, die auf Ganeti-Knoten ausgeführt werden sollen.

Sein Wert ist ein `ganeti-configuration`-Objekt, das das Paket für den Betrieb über die Befehlszeile sowie die Konfiguration der einzelnen Daemons festlegt. Auch werden zulässige Pfade für die Speicherung in Dateien sowie verfügbare Gastbetriebssysteme über diesen Datentyp konfiguriert.

`ganeti-configuration` [Datentyp]

Der `ganeti`-Dienst akzeptiert folgende Konfigurationsoptionen:

`ganeti` (Vorgabe: `ganeti`)
: Welches `ganeti`-Paket benutzt werden soll. Es wird ins Systemprofil installiert und macht die Befehlszeilenwerkzeuge `gnt-cluster`, `gnt-instance` usw. verfügbar. Beachten Sie, dass der hier angegebene Wert keinen Einfluss auf die anderen Dienste hat; jeder legt sein eigenes `ganeti`-Paket fest (siehe unten).

`noded-configuration` (Vorgabe: `(ganeti-noded-configuration)`)
`confd-configuration` (Vorgabe: `(ganeti-confd-configuration)`)
`wconfd-configuration` (Vorgabe: `(ganeti-wconfd-configuration)`)
`luxid-configuration` (Vorgabe: `(ganeti-luxid-configuration)`)
`rapi-configuration` (Vorgabe: `(ganeti-rapi-configuration)`)
`kvmd-configuration` (Vorgabe: `(ganeti-kvmd-configuration)`)
`mond-configuration` (Vorgabe: `(ganeti-mond-configuration)`)
`metad-configuration` (Vorgabe: `(ganeti-metad-configuration)`)
`watcher-configuration` (Vorgabe: `(ganeti-watcher-configuration)`)
`cleaner-configuration` (Vorgabe: `(ganeti-cleaner-configuration)`)
: Mit diesen Optionen stellen Sie die verschiedenen Daemons und Cron-Aufträge ein, die mit Ganeti eingerichtet werden. Die möglichen Werte dafür werden im Folgenden genau beschrieben. Um eine Einstellung zu ändern, müssen Sie den Konfigurationstyp für den jeweiligen Dienst verwenden:

```
(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` (Vorgabe: `'()`)
: Liste zulässiger Verzeichnisse für das in Dateien speichernde Hintergrundsystem („File Storage Backend“).

`os` (Vorgabe: `%default-ganeti-os`)
: Liste von `<ganeti-os>`-Verbundsobjekten.

Im Kern ist `ganeti-service-type` eine Kurzform davon, jeden Dienst einzeln zu deklarieren:

```
(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)
```

Außerdem enthalten ist eine Diensterweiterung für den `etc-service-type`, der das Hintergrundsystem für die Speicherung in Dateien und die Betriebssystemvarianten festlegt.

`ganeti-os` [Datentyp]
Dieser Datentyp wird verwendet, um ihn an den `os`-Parameter der `ganeti-configuration` zu übergeben. Er umfasst die folgenden Parameter:

`name`
: Der Name dieses Betriebssystemanbieters. Sein einziger Zweck ist, anzugeben, wohin die Konfigurationsdateien geschrieben werden. Wird „debootstrap“ angegeben, wird `/etc/ganeti/instance-debootstrap` erstellt.

`extension` (Vorgabe: `#f`)
: Welche Dateinamenserweiterung die Varianten dieser Art von Betriebssystem benutzen, zum Beispiel `.conf` oder `.scm`. Wenn Sie Variantendateinamen angeben, wird sie angehängt.

`variants` (Vorgabe: `'()`)
: Entweder eine Liste der `ganeti-os-variant`-Objekte für dieses Betriebssystem oder ein „dateiartiges“ Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), das das Variantenverzeichnis darstellt.

  Wenn Sie den Betriebssystemanbieter für Guix möchten, die Definitionen Ihrer Varianten aber aus einem lokalen Verzeichnis lesen möchten, statt sie einzeln zu deklarieren (siehe dazu *guix-variants* weiter unten), dann geht das so:

```
(ganeti-os
 (name "guix")
 (variants (local-file "ganeti-guix-variants"
                       #:recursive? #true)))
```

Allerdings werden Sie sich um die Datei `variants.list` darin in diesem Fall selbst kümmern müssen (siehe `ganeti-os-interface(7)` (`https://docs.ganeti.org/docs/ganeti/3.0/man/ganeti-os-interface.html`)).

`ganeti-os-variant` [Datentyp]

Der Datentyp für eine Variante einer Art von Betriebssystem. Er nimmt die folgenden Parameter:

`name` Der Name dieser Variante.

`configuration`
Eine Konfigurationsdatei für diese Variante.

`%default-debootstrap-hooks` [Scheme-Variable]

Diese Variable enthält Anknüpfungspunkte („Hooks“), um das Netzwerk zu konfigurieren und den GRUB-Bootloader einzurichten.

`%default-debootstrap-extra-pkgs` [Scheme-Variable]

Diese Variable führt eine Liste von Paketen auf, die für ein gänzlich virtualisiertes Gastsystem sinnvoll sind.

`debootstrap-configuration` [Datentyp]

Mit diesem Datentyp werden Konfigurationsdateien erzeugt, die für den debootstrap-Betriebssystemanbieter geeignet sind.

`hooks` (Vorgabe: `%default-debootstrap-hooks`)
Wenn es nicht auf `#f` steht, muss hier ein G-Ausdruck angegeben werden, der ein Verzeichnis mit Scripts (sogenannten „Hooks“, d.h. Anknüpfungspunkte) festlegt, welche bei der Installation des Betriebssystems ausgeführt werden. Es kann auch eine Liste von Paaren der Form `(Name . dateiartiges-Objekt)` angegeben werden. Zum Beispiel:

```
`((99-hallo-welt . ,(plain-file "#!/bin/sh\necho Hallo Welt")))
```

Damit wird ein Verzeichnis mit einer ausführbaren Datei namens `99-hallo-welt` festgelegt, die jedes Mal ausgeführt wird, wenn diese Variante installiert wird. Wird hier `#f` angegeben, werden die in `/etc/ganeti/instance-debootstrap/hooks` vorgefundenen Anknüpfungspunkte benutzt, falls vorhanden.

`proxy` (Vorgabe: `#f`)
Optional; welcher HTTP-Proxy genutzt werden soll.

`mirror` (Vorgabe: `#f`)
Der Spiegelserver für Debian. Üblicherweise wird etwas wie `http://ftp.no.debian.org/debian` angegeben. Die Voreinstellung hängt von der Distribution ab.

`arch` (Vorgabe: `#f`)
Die dpkg-Architektur. Setzen Sie dies z.B. auf `armhf`, um eine ARMv7-Instanz auf einem AArch64-Wirtssystem *via* debootstrap einzurichten. Bei der Vorgabeeinstellung wird die aktuelle Systemarchitektur übernommen.

`suite` (Vorgabe: `"stable"`)
: Wenn dieses Feld gesetzt ist, muss dafür eine Debian-Distributions-„Suite“ wie `buster` oder `focal` angegeben werden. Steht es auf `#f`, wird die Voreinstellung für den Betriebssystemanbieter benutzt.

`extra-pkgs` (Vorgabe: `%default-debootstrap-extra-pkgs`)
: Liste zusätzlicher Pakete, die durch dpkg zusätzlich zum minimalen System installiert werden.

`components` (Vorgabe: `#f`)
: Ist dies gesetzt, muss es eine Liste von Bereichen eines Debian-Repositorys sein, etwa `'("main" "contrib")`.

`generate-cache?` (Vorgabe: `#t`)
: Ob das generierte debootstrap-Archiv automatisch zwischengespeichert werden soll.

`clean-cache` (Vorgabe: `14`)
: Nach wie vielen Tagen der Zwischenspeicher verworfen werden soll. Geben Sie `#f` an, damit der Zwischenspeicher niemals bereinigt wird.

`partition-style` (Vorgabe: `'msdos`)
: Der Partitionstyp der anzulegenden Partition. Wenn er festgelegt ist, muss er entweder `'msdos` oder `'none` lauten oder eine Zeichenkette angegeben werden.

`partition-alignment` (Vorgabe: `2048`)
: Auf wie viele Sektoren die Partition ausgerichtet werden soll.

`debootstrap-variant` *`Name Konfiguration`* [Scheme-Prozedur]
: Diese Hilfsprozedur erzeugt ein `ganeti-os-variant`-Verbundsobjekt. Sie nimmt zwei Parameter entgegen: einen Namen und ein `debootstrap-configuration`-Objekt.

`debootstrap-os` *`Varianten...`* [Scheme-Prozedur]
: Diese Hilfsprozedur erzeugt ein `ganeti-os`-Verbundsobjekt. Sie nimmt eine Liste von mit `debootstrap-variant` erzeugten Varianten entgegen.

`guix-variant` *`Name Konfiguration`* [Scheme-Prozedur]
: Diese Hilfsprozedur erzeugt ein `ganeti-os-variant`-Verbundsobjekt, das für den vorgegebenen Guix-Betriebssystemanbieter gedacht ist. Als Parameter anzugeben sind ein Name und ein G-Ausdruck, der ein „dateiartiges“ Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) mit einer Konfiguration für Guix System liefert.

`guix-os` *`Varianten...`* [Scheme-Prozedur]
: Diese Hilfsprozedur erzeugt ein `ganeti-os`-Verbundsobjekt. Sie nimmt eine Liste von durch `guix-variant` erzeugten Varianten entgegen.

`%default-debootstrap-variants` [Scheme-Variable]
: Zur Erleichterung kann diese Variable benutzt werden, damit der debootstrap-Anbieter sofort funktioniert, ohne dass seine Nutzer Varianten selbst deklarieren müssen. Enthalten ist eine einzige debootstrap-Variante mit der Standardkonfiguration:

```
(list (debootstrap-variant
```

```
              "default"
              (debootstrap-configuration)))
```

`%default-guix-variants` [Scheme-Variable]

Zur Erleichterung kann diese Variable benutzt werden, damit der Guix-Betriebssystemanbieter sofort und ohne weitere Konfiguration funktioniert. Damit wird eine virtuelle Maschine mit einem SSH-Server, einer seriellen Konsole und bereits autorisierten Ganeti-Wirts-SSH-Schlüsseln erzeugt.

```
(list (guix-variant
       "default"
       (file-append ganeti-instance-guix
                    "/share/doc/ganeti-instance-guix/examples/dynamic.scm")))
```

Benutzer können die Unterstützung für Guix *nicht* bekannte Betriebssystemanbieter implementieren, indem sie entsprechende Erweiterungen für die `ganeti-os-` und `ganeti-os-variant`-Verbundstypen implementieren. Ein Beispiel:

```
(ganeti-os
 (name "eigenes-os")
 (extension ".conf")
 (variants
  (list (ganeti-os-variant
         (name "foo")
         (configuration (plain-file "bar" "das genügt"))))))
```

Damit wird `/etc/ganeti/instance-eigenes-os/variants/foo.conf` erzeugt, das auf eine Datei im Store mit dem Inhalt `das genügt` verweist. Außerdem wird `/etc/ganeti/instance-eigenes-os/variants/variants.list` mit dem Inhalt `foo` erzeugt.

Natürlich kann man Betriebssystemanbieter finden, für die das nicht ausreicht. Wenn Sie sich durch die Schnittstelle in Ihren Möglichkeiten eingeschränkt fühlen, schreiben Sie bitte an `guix-devel@gnu.org`.

Der übrige Teil dieses Abschnitts beschreibt die verschiedenen Dienste, aus denen sich der `ganeti-service-type` zusammensetzt.

`ganeti-noded-service-type` [Scheme-Variable]

`ganeti-noded` ist der Daemon, der für knotenbezogene Funktionen des Ganeti-Systems da ist. Sein Wert muss ein `ganeti-noded-configuration`-Objekt sein.

`ganeti-noded-configuration` [Datentyp]

Dies ist die Konfiguration des `ganeti-noded`-Dienstes für Ganetis Knoten-Daemon.

`ganeti` (Vorgabe: `ganeti`)
: Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

`port` (Vorgabe: `8081`)
: Der TCP-Port, auf dem der Knoten-Daemon (`noded`) auf Netzwerkanfragen lauscht.

`address` (Vorgabe: `"0.0.0.0"`)
: An welche Netzwerkadresse sich der Daemon binden wird. Die Vorgabeeinstellung bedeutet, sich an alle verfügbaren Adressen zu binden.

`interface` (Vorgabe: `#f`)
: Wenn dieses Feld gesetzt ist, muss es eine bestimmte Netzwerkschnittstelle angeben (z.B. `eth0`), an die sich der Daemon binden wird.

`max-clients` (Vorgabe: `20`)
: Legt eine Maximalzahl gleichzeitiger Client-Verbindungen fest, um die sich der Daemon kümmert. Darüber hinaus werden Verbindungen zwar akzeptiert, aber erst beantwortet, wenn genügend viele Verbindungen wieder geschlossen wurden.

`ssl?` (Vorgabe: `#t`)
: Ob SSL/TLS benutzt werden soll, um Netzwerkkommunikation zu verschlüsseln. Das Zertifikat wird automatisch vom Cluster eingespielt und kann über `gnt-cluster renew-crypto` rotiert werden.

`ssl-key` (Vorgabe: `"/var/lib/ganeti/server.pem"`)
: Hiermit kann ein bestimmter Schlüssel für mit TLS verschlüsselte Kommunikation festgelegt werden.

`ssl-cert` (Vorgabe: `"/var/lib/ganeti/server.pem"`)
: Hiermit kann ein bestimmtes Zertifikat für mit TLS verschlüsselte Kommunikation festgelegt werden.

`debug?` (Vorgabe: `#f`)
: Steht dies auf wahr, führt der Daemon zum Zweck der Fehlersuche ausführlichere Protokolle. Beachten Sie, dass dadurch Details der Verschlüsselung in die Protokolldateien fließen können. Seien Sie vorsichtig!

`ganeti-confd-service-type` [Scheme-Variable]
: `ganeti-confd` beantwortet Anfragen, die mit der Konfiguration eines Ganeti-Clusters zu tun haben. Der Zweck dieses Daemons ist, eine hochverfügbare und schnelle Methode zum Anfragen von Cluster-Konfigurationswerten zu bieten. Er wird automatisch auf allen Kandidaten für die Rolle des „Master“-Knotens aktiviert. Der Wert dieses Dienstes muss ein `ganeti-confd-configuration`-Objekt sein.

`ganeti-confd-configuration` [Datentyp]
: Dies ist die Konfiguration des `ganeti-confd`-Dienstes.

`ganeti` (Vorgabe: `ganeti`)
: Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

`port` (Vorgabe: `8081`)
: Der UDP-Port, auf dem auf Netzwerkanfragen gelauscht werden soll.

`address` (Vorgabe: `"0.0.0.0"`)
: An welche Netzwerkadresse sich der Daemon binden soll.

`debug?` (Vorgabe: `#f`)
: Steht es auf wahr, wird der Daemon zum Zweck der Fehlersuche ausführlicher protokollieren.

`ganeti-wconfd-service-type` [Scheme-Variable]
: `ganeti-wconfd` ist der Daemon, der mit autoritativem Wissen über die Cluster-Konfiguration ausgestattet ist, und die einzige Entität, die Änderungen daran akzeptieren kann. Alle Aufträge, die die Konfiguration ändern müssen, tun dies, indem sie entsprechende Anfragen an den Daemon stellen. Er läuft nur auf dem „Master"-Knoten und deaktiviert sich auf allen anderen Knoten automatisch.

  Der Wert dieses Dienstes muss ein `ganeti-wconfd-configuration`-Objekt sein.

`ganeti-wconfd-configuration` [Datentyp]
: Dies ist die Konfiguration des `ganeti-wconfd`-Dienstes.

  `ganeti` (Vorgabe: `ganeti`)
  : Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

  `no-voting?` (Vorgabe: `#f`)
  : Der Daemon wird das Starten verweigern, sofern die Mehrheit der Knoten *nicht* anerkennt, dass er auf dem Master-Knoten läuft. Setzen Sie dies auf `#t`, um ihn selbst dann zu starten, wenn sich keine Mehrheit dafür findet (das ist gefährlich; hoffentlich wissen Sie, was Sie tun).

  `debug?` (Vorgabe: `#f`)
  : Steht es auf wahr, wird der Daemon zum Zweck der Fehlersuche ausführlicher protokollieren.

`ganeti-luxid-service-type` [Scheme-Variable]
: Der Daemon `ganeti-luxid` ist dazu da, Anfragen zur Konfiguration und zum aktuellen Zustand des laufenden Ganeti-Clusters zu beantworten. Des Weiteren ist es der autoritative Daemon für Ganetis Auftragsliste. Aufträge können über diesen Daemon eingereicht werden und werden von ihm geplant und gestartet.

  Er nimmt ein `ganeti-luxid-configuration`-Objekt entgegen.

`ganeti-luxid-configuration` [Datentyp]
: Dies ist die Konfiguration des `ganeti-luxid`-Dienstes.

  `ganeti` (Vorgabe: `ganeti`)
  : Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

  `no-voting?` (Vorgabe: `#f`)
  : Der Daemon verweigert das Starten, wenn er sich nicht sicher sein kann, dass die Mehrheit der Cluster-Knoten überzeugt ist, dass er auf dem Master-Knoten läuft. Setzen Sie dies auf `#t`, um solche Hinweise zu ignorieren (das kann gefährlich sein!).

  `debug?` (Vorgabe: `#f`)
  : Steht es auf wahr, wird der Daemon zum Zweck der Fehlersuche ausführlicher protokollieren.

`ganeti-rapi-service-type` [Scheme-Variable]

`ganeti-rapi` bietet eine Schnittstelle für entfernte Aufrufe (englisch *Remote API*) für Ganeti-Cluster. Sie läuft auf dem Master-Knoten und mit ihr können Aktionen auf dem Cluster programmatisch mittels eines JSON-basierten Protokolls für entfernte Prozeduraufrufe durchgeführt werden.

Die meisten Anfrageoperationen werden ohne Authentisierung zugelassen (außer wenn *require-authentication?* gesetzt ist), wohingegen Schreibzugriffe einer ausdrücklichen Authentisierung durch die Datei `/var/lib/ganeti/rapi/users` bedürfen. Siehe die Dokumentation der Ganeti Remote API (`https://docs.ganeti.org/docs/ganeti/3.0/html/rapi.html`) für mehr Informationen.

Der Wert dieses Dienstes muss ein `ganeti-rapi-configuration`-Objekt sein.

`ganeti-rapi-configuration` [Datentyp]

Dies ist die Konfiguration des `ganeti-rapi`-Dienstes.

`ganeti` (Vorgabe: `ganeti`)
: Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

`require-authentication?` (Vorgabe: `#f`)
: Ob selbst für Nur-Lese-Operationen eine Authentisierung verlangt werden soll.

`port` (Vorgabe: `5080`)
: Der TCP-Port, auf dem auf API-Anfragen gelauscht werden soll.

`address` (Vorgabe: `"0.0.0.0"`)
: An welche Netzwerkadresse sich der Dienst binden soll. Nach Vorgabe wird auf allen konfigurierten Adressen gelauscht.

`interface` (Vorgabe: `#f`)
: Wenn dies gesetzt ist, muss es eine bestimmte Netzwerkschnittstelle angeben wie z.B. `eth0`, an die sich der Daemon binden wird.

`max-clients` (Vorgabe: `20`)
: Die Maximalzahl gleichzeitiger Verbindungen, um die sich der Dienst kümmern darf. Weitere Verbindungen sind möglich, über sie wird aber erst dann geantwortet, wenn genügend viele Verbindungen wieder geschlossen wurden.

`ssl?` (Vorgabe: `#t`)
: Ob SSL-/TLS-Verschlüsselung auf dem RAPI-Port eingesetzt werden soll.

`ssl-key` (Vorgabe: `"/var/lib/ganeti/server.pem"`)
: Hiermit kann ein bestimmter Schlüssel für mit TLS verschlüsselte Kommunikation festgelegt werden.

`ssl-cert` (Vorgabe: `"/var/lib/ganeti/server.pem"`)
: Hiermit kann ein bestimmtes Zertifikat für mit TLS verschlüsselte Kommunikation festgelegt werden.

`debug?` (Vorgabe: `#f`)
: Steht dies auf wahr, führt der Daemon zum Zweck der Fehlersuche ausführlichere Protokolle. Beachten Sie, dass dadurch Details der

Verschlüsselung in die Protokolldateien fließen können. Seien Sie vorsichtig!

`ganeti-kvmd-service-type` [Scheme-Variable]

`ganeti-kvmd` ist dafür verantwortlich, festzustellen, wenn eine gegebene KVM-Instanz durch einen Administrator oder Nutzer geschlossen wurde. Normalerweise wird Ganeti Instanzen neu starten, die nicht über ihn selbst gestoppt wurden. Wenn die Cluster-Option `user_shutdown` wahr ist, beobachtet der Daemon den durch QEMU bereitgestellten `QMP`-Socket und lauscht auf Abschalteereignisse. Solche Instanzen werden von ihm als durch den Benutzer heruntergefahren (*USER_down*) markiert statt als durch Fehler beendet (*ERROR_down*), wenn sie von selbst ordnungsgemäß heruntergefahren sind.

Der Dienst benutzt ein `ganeti-kvmd-configuration`-Objekt.

`ganeti-kvmd-configuration` [Datentyp]

`ganeti` (Vorgabe: `ganeti`)
: Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

`debug?` (Vorgabe: `#f`)
: Steht es auf wahr, wird der Daemon zum Zweck der Fehlersuche ausführlicher protokollieren.

`ganeti-mond-service-type` [Scheme-Variable]

`ganeti-mond` ist ein optionaler Daemon, mit dem Ganetis Überwachungsfunktionalität gewährleistet wird. Er ist dafür verantwortlich, Datensammler auszuführen und die gesammelten Informationen über eine HTTP-Schnittstelle bereitzustellen.

Der Dienst benutzt ein `ganeti-mond-configuration`-Objekt.

`ganeti-mond-configuration` [Datentyp]

`ganeti` (Vorgabe: `ganeti`)
: Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

`port` (Vorgabe: `1815`)
: Der Port, auf dem der Daemon lauschen wird.

`address` (Vorgabe: `"0.0.0.0"`)
: Die Netzwerkadresse, an die sich der Daemon binden soll, nach Vorgabe an alle verfügbaren.

`debug?` (Vorgabe: `#f`)
: Steht es auf wahr, wird der Daemon zum Zweck der Fehlersuche ausführlicher protokollieren.

`ganeti-metad-service-type` [Scheme-Variable]

`ganeti-metad` ist ein optionaler Daemon, der benutzt werden kann, um Informationen über den Cluster an Instanzen oder an Betriebssysteminstallationsskripte weiterzugeben.

Dazu nimmt er ein `ganeti-metad-configuration`-Objekt.

`ganeti-metad-configuration` [Datentyp]

`ganeti` (Vorgabe: `ganeti`)
: Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

`port` (Vorgabe: `80`)
: Der Port, auf dem der Daemon lauschen wird.

`address` (Vorgabe: `#f`)
: Wenn es gesetzt ist, wird sich der Daemon nur an diese Adresse binden. Ist es *nicht* gesetzt, hängt das Verhalten von der Cluster-Konfiguration ab.

`debug?` (Vorgabe: `#f`)
: Steht es auf wahr, wird der Daemon zum Zweck der Fehlersuche ausführlicher protokollieren.

`ganeti-watcher-service-type` [Scheme-Variable]

`ganeti-watcher` ist ein Skript, das dafür gedacht ist, regelmäßig ausgeführt zu werden und die Verfügbarkeit des Clusters sicherzustellen. Instanzen, die ohne Ganetis Zustimmung abgebrochen sind, werden durch es automatisch neu gestartet und DRBD-Verbindungen werden repariert, wenn ein Knoten neu gestartet wurde. Außerdem werden alte Cluster-Aufträge archiviert und nicht laufende Ganeti-Daemons erneut gestartet. Wenn der Cluster-Parameter `ensure_node_health` gesetzt ist, wird der Watcher auch Instanzen und DRBD-Geräte herunterfahren, wenn der Knoten zwar läuft, aber von bekannten Kandidaten auf die Rolle des Master-Knotens als offline deklariert wurde.

Er kann mit `gnt-cluster watcher pause` auf allen Knoten pausiert werden.

Der Dienst benutzt ein `ganeti-watcher-configuration`-Objekt.

`ganeti-watcher-configuration` [Datentyp]

`ganeti` (Vorgabe: `ganeti`)
: Das `ganeti`-Paket, was für diesen Dienst benutzt werden soll.

`schedule` (Vorgabe: `'(next-second-from (next-minute (range 0 60 5)))`)
: Wie oft das Skript ausgeführt werden soll. Nach Vorgabe läuft es alle fünf Minuten.

`rapi-ip` (Vorgabe: `#f`)
: Diese Option muss nur dann angegeben werden, wenn der RAPI-Daemon so konfiguriert wurde, dass er eine bestimmte Schnittstelle oder Adresse verwenden soll. Nach Vorgabe wird die Adresse des Clusters verwendet.

`job-age` (Vorgabe: `(* 6 3600)`)
: Cluster-Aufträge archivieren, die älter als diese Anzahl Sekunden sind. Die Vorgabe beträgt 6 Stunden. Dadurch wird gewährleistet, dass man `gnt-job list` noch sinnvoll bedienen kann.

`verify-disks?` (Vorgabe: `#t`)
: Steht dies auf `#f`, wird der Watcher *nicht* versuchen, fehlgeschlagene DRBD-Verbindungen automatisch zu reparieren. Administratoren müssen stattdessen `gnt-cluster verify-disks` manuell aufrufen.

`debug?` (Vorgabe: `#f`)
: Steht dies auf `#t`, so führt das Skript zum Zweck der Fehlersuche ausführlichere Protokolle.

`ganeti-cleaner-service-type` [Scheme-Variable]
: `ganeti-cleaner` ist ein Skript, das regelmäßig ausgeführt werden soll, um alte Dateien vom Cluster zu entfernen. Dieser Diensttyp steuert zwei *cron-Aufträge* (*cron jobs*): einer, um alte Cluster-Aufträge auf dem Master-Knoten andauernd zu löschen, und einer, der auf allen Knoten ausgelaufene X509-Zertifikate, -Schlüssel sowie veraltete Informationen des `ganeti-watcher` entfernt. Wie alle Ganeti-Dienste kann man ihn ohne Probleme auch auf Nicht-Master-Knoten in die Konfiguration mit aufnehmen, denn er deaktiviert sich selbst, wenn er nicht gebraucht wird.

  Der Dienst benutzt ein `ganeti-cleaner-configuration`-Objekt.

`ganeti-cleaner-configuration` [Datentyp]
: `ganeti` (Vorgabe: `ganeti`)
  : Welches `ganeti`-Paket für den Befehl `gnt-cleaner` benutzt werden soll.

  `master-schedule` (Vorgabe: `"45 1 * * *"`)
  : Wie oft der Bereinigungsauftrag für den Master durchgeführt werden soll. Vorgegeben ist einmal täglich um 01:45:00 Uhr.

  `node-schedule` (Vorgabe: `"45 2 * * *"`)
  : Wie oft der Bereinigungsauftrag für Knoten durchgeführt werden soll. Vorgegeben ist einmal täglich um 02:45:00 Uhr.

12.9.30 Versionskontrolldienste

Das Modul `(gnu services version-control)` stellt einen Dienst zur Verfügung, der einen Fernzugriff auf lokale Git-Repositorys ermöglicht. Dafür gibt es drei Möglichkeiten: den `git-daemon-service`, der Zugang zu Repositorys über das ungesicherte, TCP-basierte `git://`-Protokoll gewährt, das Erweitern des `nginx`-Webservers, um ihn als Proxy für Anfragen an das `git-http-backend` einzusetzen, oder mit dem `cgit-service-type` eine Weboberfläche zur Verfügung zu stellen.

`git-daemon-service` [*#:config* (*git-daemon-configuration*)] [Scheme-Prozedur]
: Liefert einen Dienst, der `git daemon` ausführt. Der Befehl startet den Git-Daemon, einen einfachen TCP-Server, um Repositorys über das Git-Protokoll für anonymen Zugriff zugänglich zu machen.

  Das optionale Argument *config* sollte ein `<git-daemon-configuration>`-Objekt sein. Nach Vorgabe wird Lese-Zugriff auf exportierte[6] Repositorys in `/srv/git` gewährt.

`git-daemon-configuration` [Datentyp]
: Datentyp, der die Konfiguration für `git-daemon-service` repräsentiert.

  `package` (Vorgabe: `git`)
  : Paketobjekt des verteilten Versionskontrollsystems Git.

[6] Das geschieht, indem die magische Datei `git-daemon-export-ok` im Repository erzeugt wird.

`export-all?` (Vorgabe: `#f`)
: Ob Zugriff auf alle Git-Repositorys gewährt werden soll, selbst wenn keine `git-daemon-export-ok`-Datei in ihrem Verzeichnis gefunden wird.

`base-path` (Vorgabe: `/srv/git`)
: Ob alle Pfadanfragen behandelt werden sollen, als wären sie relativ zum angegebenen Pfad. Wenn Sie `git daemon` mit `(base-path "/srv/git")` auf '`example.com`' ausführen und später versuchen, '`git://example.com/hello.git`' zu pullen, wird der Git-Daemon den Pfad als `/srv/git/hello.git` interpretieren.

`user-path` (Vorgabe: `#f`)
: Ob die `~benutzerkonto`-Notation in Anfragen verwendet werden darf. Wird hier die leere Zeichenkette angegeben, werden Anfragen an '`git://host/~alice/foo`' als Anfragen verstanden, auf das `foo`-Repository im Persönlichen Verzeichnis des `alice`-Benutzerkontos verstanden. Wird `(user-path "pfad")` angegeben, wird dieselbe Anfrage als eine Anfrage verstanden, auf das *`pfad`*`/foo`-Repository im Persönlichen Verzeichnis des `alice`-Benutzerkontos zuzugreifen.

`listen` (Vorgabe: `'()`)
: Ob auf bestimmte IP-Adressen oder Rechnernamen („Hostnames“) gelauscht werden soll. Vorgegeben ist auf allen.

`port` (Vorgabe: `#f`)
: Ob auf einer alternativen Portnummer gelauscht werden soll. Vorgegeben ist 9418.

`whitelist` (Vorgabe: `'()`)
: Wenn dies nicht leer gelassen wird, wird nur der Zugriff auf die aufgelisteten Verzeichnisse gewährt.

`extra-options` (Vorgabe: `'()`)
: Zusätzliche Befehlszeilenoptionen, die dem `git daemon` mitgegeben werden sollen. Bitte führen Sie `man git-daemon` aus, um weitere Informationen zu erhalten.

Zugriffe über das `git://`-Protokoll werden nicht authentifiziert. Wenn Sie von einem Repository pullen, dass Sie über `git://` geholt haben, wissen Sie nicht, ob die empfangenen Daten modifiziert wurden oder auch nur vom angegebenen Rechner kommen, und Ihre Verbindung kann abgehört werden. Es ist besser, eine authentifizierte und verschlüsselte Übertragungsart zu verwenden, zum Beispiel `https`. Obwohl Git es Ihnen ermöglicht, Repositorys über schlichte dateibasierte Webserver anzubieten, gibt es ein schnelleres Protokoll, das vom `git-http-backend`-Programm implementiert wird. Dieses Programm dient als Hintergrundsystem für einen ordentlichen Git-Webdienst. Es wurde so konstruiert, dass es über einen FastCGI-Proxy abrufbar ist. Siehe Abschnitt 12.9.19 [Web-Dienste], Seite 466, für weitere Informationen, wie Sie den benötigten `fcgiwrap`-Daemon ausführen.

Guix hat einen separaten Konfigurationsdatentyp, um Git-Repositorys über HTTP anzubieten.

`git-http-configuration` [Datentyp]
Datentyp, der in Zukunft die Konfiguration eines `git-http-service-type` repräsentieren soll; zurzeit kann damit Nginx über `git-http-nginx-location-configuration` eingerichtet werden.

`package` (Vorgabe: *git*)
Paketobjekt des verteilten Versionskontrollsystems Git.

`git-root` (Vorgabe: `/srv/git`)
Das Verzeichnis, das die Git-Repositorys enthält, die der Allgemeinheit zugänglich gemacht werden sollen.

`export-all?` (Vorgabe: `#f`)
Ob alle Git-Repositorys in *git-root* zugänglich gemacht werden sollen, selbst wenn keine `git-daemon-export-ok`-Datei in ihrem Verzeichnis gefunden wird.

`uri-path` (Vorgabe: '`/git/`')
Präfix für Pfade beim Git-Zugriff. Beim vorgegebenen Präfix '`/git/`' wird '`http://`*server*`/git/`*repo*`.git`' auf `/srv/git/`*repo*`.git` abgebildet. Anfragen, deren URI-Pfade nicht mit dem Präfix beginnen, werden nicht an die Git-Instanz weitergereicht.

`fcgiwrap-socket` (Vorgabe: `127.0.0.1:9000`)
Der Socket, auf dem der `fcgiwrap`-Daemon lauscht. Siehe Abschnitt 12.9.19 [Web-Dienste], Seite 466.

Es gibt zurzeit keinen `git-http-service-type`, stattdessen können Sie eine `nginx-location-configuration` aus einer `git-http-configuration` heraus erstellen und als Location zu einem Webserver hinzufügen.

`git-http-nginx-location-configuration` [Scheme-Prozedur]
[*config*=(*git-http-configuration*)] *Eine* `nginx-location-configuration` berechnen, die der angegebenen Git-HTTP-Konfiguration entspricht. Ein Beispiel für eine nginx-Dienstdefinition, um das vorgegebene `/srv/git`-Verzeichnis über HTTPS anzubieten, könnte so aussehen:

```
(service nginx-service-type
         (nginx-configuration
          (server-blocks
           (list
            (nginx-server-configuration
             (listen '("443 ssl"))
             (server-name "git.mein-rechner.org")
             (ssl-certificate
              "/etc/letsencrypt/live/git.mein-rechner.org/fullchain.pem")
             (ssl-certificate-key
              "/etc/letsencrypt/live/git.mein-rechner.org/privkey.pem")
             (locations
              (list
               (git-http-nginx-location-configuration
```

```
(git-http-configuration (uri-path "/")))))))))))
```

Für dieses Beispiel nehmen wir an, dass Sie Ihr TLS-Zertifikat über Let's Encrypt beziehen. Siehe Abschnitt 12.9.20 [Zertifikatsdienste], Seite 487. Der vorgegebene `certbot`-Dienst leitet alle HTTP-Anfragen nach `git.mein-rechner.org` auf HTTPS um. Zu Ihren Systemdiensten werden Sie auch einen `fcgiwrap`-Proxy hinzufügen müssen. Siehe Abschnitt 12.9.19 [Web-Dienste], Seite 466.

Cgit-Dienst

Cgit (`https://git.zx2c4.com/cgit/`) ist eine in C geschriebene Weboberfläche als Vordergrundsystem für Git-Repositorys.

Im folgenden Beispiel wird der Dienst mit den vorgegebenen Werten eingerichtet. Nach Vorgabe kann auf Cgit auf Port 80 unter `http://localhost:80` zugegriffen werden.

```
(service cgit-service-type)
```

Der Typ `Dateiobjekt` bezeichnet entweder ein dateiartiges Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) oder eine Zeichenkette.

Verfügbare `cgit-configuration`-Felder sind:

`„package“ package` [`cgit-configuration`-Parameter]
Das CGIT-Paket.

`„nginx-server-configuration-list“ nginx` [`cgit-configuration`-Parameter]
NGINX-Konfiguration.

`Dateiobjekt about-filter` [`cgit-configuration`-Parameter]
Gibt einen Befehl an, der zur Formatierung des Inhalts der Übersichtsseiten aufgerufen wird (sowohl auf oberster Ebene und für jedes Repository).

Die Vorgabe ist ‘`""`’.

`Zeichenkette agefile` [`cgit-configuration`-Parameter]
Gibt einen Pfad relativ zu jedem Repository-Pfad an, unter dem eine Datei gespeichert sein kann, die Datum und Uhrzeit des jüngsten Commits im Repository angibt.

Die Vorgabe ist ‘`""`’.

`Dateiobjekt auth-filter` [`cgit-configuration`-Parameter]
Gibt einen Befehl an, der aufgerufen wird, um Benutzer zu authentifizieren.

Die Vorgabe ist ‘`""`’.

`Zeichenkette branch-sort` [`cgit-configuration`-Parameter]
Wenn diese Option auf ‘`age`’ gesetzt wurde, wird die Liste der Branch-Referenzen nach Datum sortiert, und wenn sie auf ‘`name`’ gesetzt wurde, wird nach dem Branch-Namen sortiert.

Die Vorgabe ist ‘`"name"`’.

`Zeichenkette cache-root` [`cgit-configuration`-Parameter]
Pfad, unter dem Cgit-Zwischenspeichereinträge abgelegt werden.

Die Vorgabe ist ‘`"/var/cache/cgit"`’.

`Ganze-Zahl cache-static-ttl` [`cgit-configuration`-Parameter]
Zahl, die angibt, wie viele Minuten die Zwischenspeicherungen für Repository-Seiten mit fester SHA1-Summe gültig bleiben, auf die zugegriffen wird („Time-to-live“).

Die Vorgabe ist '`-1`'.

`Ganze-Zahl cache-dynamic-ttl` [`cgit-configuration`-Parameter]
Zahl, die angibt, wie viele Minuten die Zwischenspeicherungen für Repository-Seiten mit veränderlicher SHA1-Summe gültig bleiben, auf die zugegriffen wird.<

Die Vorgabe ist '`5`'.

`Ganze-Zahl cache-repo-ttl` [`cgit-configuration`-Parameter]
Zahl, die angibt, wie viele Minuten die Zwischenspeicherungen für die Übersichtsseiten („summary“) von Repositorys gültig bleiben.

Die Vorgabe ist '`5`'.

`Ganze-Zahl cache-root-ttl` [`cgit-configuration`-Parameter]
Zahl, die angibt, wie viele Minuten die Zwischenspeicherung der Seite mit dem Repository-Index gültig bleibt.

Die Vorgabe ist '`5`'.

`Ganze-Zahl cache-scanrc-ttl` [`cgit-configuration`-Parameter]
Zahl, die angibt, wie viele Minuten die Zwischenspeicherung des Ergebnisses einer Suche in einem Pfad nach Git-Repositorys gültig bleibt.

Die Vorgabe ist '`15`'.

`Ganze-Zahl cache-about-ttl` [`cgit-configuration`-Parameter]
Zahl, die angibt, wie viele Minuten die Zwischenspeicherungen für die Beschreibungsseiten („about“) von Repositorys gültig bleiben.

Die Vorgabe ist '`15`'.

`Ganze-Zahl cache-snapshot-ttl` [`cgit-configuration`-Parameter]
Zahl, die angibt, wie viele Minuten die Zwischenspeicherungen für die Snapshots von Repositorys gültig bleiben.

Die Vorgabe ist '`5`'.

`Ganze-Zahl cache-size` [`cgit-configuration`-Parameter]
Wie viele Einträge der Cgit-Zwischenspeicher höchstens haben kann. Wird '`0`' festgelegt, wird *nicht* zwischengespeichert.

Die Vorgabe ist '`0`'.

`Boolescher-Ausdruck case-sensitive-sort?` [`cgit-configuration`-Parameter]
Ob beim Sortieren von Objekten in der Repository-Liste die Groß-/Kleinschreibung beachtet werden soll.

Die Vorgabe ist '`#t`'.

`Liste clone-prefix` [`cgit-configuration`-Parameter]
Liste gemeinsamer Präfixe, von denen ein Repository geklont werden kann. D.h. dass, wenn eines mit einer Repository-URL kombiniert wird, eine gültige URL zum Klonen des Repositorys entsteht.

Die Vorgabe ist '`()`'.

`Liste clone-url` [`cgit-configuration`-Parameter]
Liste von Schablonen, aus denen eine `clone-url` entsteht.

Die Vorgabe ist '`()`'.

`Dateiobjekt commit-filter` [`cgit-configuration`-Parameter]
Befehl, mit dem Commit-Nachrichten formatiert werden.

Die Vorgabe ist '`""`'.

`Zeichenkette commit-sort` [`cgit-configuration`-Parameter]
Wenn diese Option als '`date`' festgelegt wird, wird das Commit-Log streng nach Datum geordnet. Wenn sie auf '`topo`' gesetzt ist, wird es streng topologisch geordnet.

Die Vorgabe ist '`"git log"`'.

`Dateiobjekt css` [`cgit-configuration`-Parameter]
URL, die angibt, welches CSS-Dokument von jeder Cgit-Seite eingebunden werden soll.

Die Vorgabe ist '`"/share/cgit/cgit.css"`'.

`Dateiobjekt email-filter` [`cgit-configuration`-Parameter]
Gibt einen Befehl an, um die Namen und E-Mail-Adressen der Committer, Autoren und Tagger zu formatieren, die an verschiedenen Stellen in der Oberfläche von Cgit vorkommen.

Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck embedded?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, wird Cgit ein HTML-Fragment erzeugen, das für die Einbettung in andere HTML-Seiten geeignet ist.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-commit-graph?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, wird Cgit den Graphen der Commit-Historie links von den Commit-Nachrichten auf den Commit-Log-Seiten mit ASCII-Zeichen darstellen.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-filter-overrides?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, können alle Filtereinstellungen durch die cgitrc-Dateien für das jeweilige Repository geändert werden.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-follow-links?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, können Benutzer in der Log-Ansicht einer Datei folgen („–follow“).

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-http-clone?` [`cgit-configuration`-Parameter]
Wenn es auf '`#t`' gesetzt ist, kann Cgit als Endpunkt für eine Dumb-HTTP-Übertragung mit „git clone“ benutzt werden.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck enable-index-links?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, legt Cgit für jedes Repository zusätzlich Hyperlinks „summary“, „commit“, „tree“ im Repository-Index an.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-index-owner?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, zeigt Cgit den Besitzer für jedes Repository im Repository-Index an.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck enable-log-filecount?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, zeigt Cgit für jeden Commit auf den Repository-Log-Seiten die geänderten Dateien an.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-log-linecount?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, zeigt Cgit für jeden Commit auf den Repository-Log-Seiten die Anzahl der hinzugefügten und entfernten Zeilen an.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-remote-branches?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, zeigt Cgit unter den „summary“- und „ref“-Seiten entfernte Branches an.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-subject-links?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, zeigt Cgit für Links auf Eltern-Commits die Betreffzeile des Eltern-Commits als Linktext in der Commit-Ansicht an.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-html-serving?` [`cgit-configuration`-Parameter]
Flag which, when set to '`#t`', will make cgit use the subject of the parent commit as link text when generating links to parent commits in commit view.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck enable-tree-linenumbers?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, zeigt Cgit für jeden Blob aus reinem Text Links auf dessen Zeilennummern in der Baumansicht („tree“) an.

Die Vorgabe ist '`#t`'.

`Boolescher-Ausdruck enable-git-config?` [`cgit-configuration`-Parameter]
Flag which, when set to '`#f`', will allow cgit to use Git config to set any repo specific settings.

Vorgegeben ist '`#f`'.

`Dateiobjekt favicon` [`cgit-configuration`-Parameter]
URL, auf der ein Cgit-Symbol für die Anzeige in einem Webbrowser zu finden ist.

Die Vorgabe ist '`"/favicon.ico"`'.

`Zeichenkette footer` [`cgit-configuration`-Parameter]
Der Inhalt der für diese Option angegebenen Datei wird wortwörtlich am Ende jeder Seite eingefügt (d.h. er ersetzt die vorgegebene Mitteilung „generated by . . . “).

Die Vorgabe ist '`""`'.

`Zeichenkette head-include` [`cgit-configuration`-Parameter]
Der Inhalt der für diese Option angegebenen Datei wird wortwörtlich im HTML-HEAD-Bereich jeder Seite eingefügt.

Die Vorgabe ist '`""`'.

`Zeichenkette header` [`cgit-configuration`-Parameter]
Der Inhalt der für diese Option angegebenen Datei wird wortwörtlich am Anfang jeder Seite eingefügt.

Die Vorgabe ist '`""`'.

`Dateiobjekt include` [`cgit-configuration`-Parameter]
Der Name einer Konfigurationsdatei, deren Inhalt eingefügt werden soll, bevor die übrige hier angegebene Konfiguration eingelesen wird.

Die Vorgabe ist '`""`'.

`Zeichenkette index-header` [`cgit-configuration`-Parameter]
Der Inhalt der mit dieser Option angegebenen Datei wird wortwörtlich oberhalb des Repository-Index eingefügt.

Die Vorgabe ist '`""`'.

`Zeichenkette index-info` [`cgit-configuration`-Parameter]
Der Inhalt der mit dieser Option angegebenen Datei wird wortwörtlich unterhalb der Überschrift auf jeder Repository-Index-Seite eingefügt.

Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck local-time?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, gibt Cgit die Zeitstempel von Commits und Tags in der Zeitzone des Servers an.

Vorgegeben ist '`#f`'.

`Dateiobjekt logo` [`cgit-configuration`-Parameter]
URL, unter der ein Bild zu finden ist, das auf allen Cgit-Seiten als Logo zu sehen sein wird.

Die Vorgabe ist '`"/share/cgit/cgit.png"`'.

`Zeichenkette logo-link` [`cgit-configuration`-Parameter]
URL, die geladen wird, wenn jemand auf das Logo-Bild klickt.

Die Vorgabe ist '`""`'.

`Dateiobjekt owner-filter` [`cgit-configuration`-Parameter]
Befehl, der aufgerufen wird, um die Besitzerspalte auf der Hauptseite zu formatieren.

Die Vorgabe ist '`""`'.

`Ganze-Zahl max-atom-items` [`cgit-configuration`-Parameter]
Anzahl der Objekte, die in der Atom-Feed-Ansicht angezeigt werden sollen.

Die Vorgabe ist '`10`'.

`Ganze-Zahl max-commit-count` [`cgit-configuration`-Parameter]
Anzahl der Einträge, die in der Log-Ansicht pro Seite angezeigt werden sollen.

Die Vorgabe ist '`50`'.

`Ganze-Zahl max-message-length` [`cgit-configuration`-Parameter]
Anzahl der Zeichen, die in der Log-Ansicht von jeder Commit-Nachricht angezeigt werden sollen.

Die Vorgabe ist '`80`'.

`Ganze-Zahl max-repo-count` [`cgit-configuration`-Parameter]
Gibt an, wie viele Einträge auf jeder Seite der Repository-Index-Seiten stehen.

Die Vorgabe ist '`50`'.

`Ganze-Zahl max-repodesc-length` [`cgit-configuration`-Parameter]
Gibt die maximale Anzahl der Zeichen an, die von jeder Repository-Beschreibung auf den Repository-Index-Seiten angezeigt werden sollen.

Die Vorgabe ist '`80`'.

`Ganze-Zahl max-blob-size` [`cgit-configuration`-Parameter]
Gibt die maximale Größe eines Blobs in Kilobytes an, für den HTML angezeigt werden soll.

Die Vorgabe ist '`0`'.

`Zeichenkette max-stats` [`cgit-configuration`-Parameter]
Maximaler Zeitraum für Statistiken. Gültige Werte sind '`week`' (Woche), '`month`' (Monat), '`quarter`' (Quartal) und '`year`' (Jahr).

Die Vorgabe ist '`""`'.

`Mimetype-Assoziative-Liste mimetype` [`cgit-configuration`-Parameter]
Mimetype je für die angegebene Dateinamenserweiterung.

Die Vorgabe ist '`((gif "image/gif") (html "text/html") (jpg "image/jpeg") (jpeg "image/jpeg") (pdf "application/pdf") (png "image/png") (svg "image/svg+xml"))`'.

`Dateiobjekt mimetype-file` [`cgit-configuration`-Parameter]
Gibt an, welche Datei zur automatischen Auflösung des Mimetypes benutzt werden soll.

Die Vorgabe ist '`""`'.

`Zeichenkette module-link` [`cgit-configuration`-Parameter]
Text, der als Formatzeichenkette für einen Hyperlink benutzt wird, wenn in einer Verzeichnisauflistung ein Submodul ausgegeben wird.

Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck nocache?` [`cgit-configuration`-Parameter]
Wenn dies auf '`#t`' gesetzt ist, wird nicht zwischengespeichert.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck noplainemail?` [`cgit-configuration`-Parameter]
Wenn dies auf '`#t`' gesetzt ist, werden keine vollen E-Mail-Adressen angezeigt.

Vorgegeben ist '`#f`'.

`Boolescher-Ausdruck noheader?` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt ist, wird Cgit die Standardseitenkopf auf allen Seiten weglassen.

Vorgegeben ist '`#f`'.

`Projektliste project-list` [`cgit-configuration`-Parameter]
Eine Liste der Unterverzeichnisse innerhalb des mit `repository-directory` festgelegten Verzeichnisses, relativ dazu angegeben, die als Git-Repositorys geladen werden sollen. Eine leere Liste bedeutet, dass alle Unterverzeichnisse geladen werden.

Die Vorgabe ist '`()`'.

`Dateiobjekt readme` [`cgit-configuration`-Parameter]
Text, der als voreingestellter Wert für `cgit-repo-readme` benutzt wird.

Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck remove-suffix?` [`cgit-configuration`-Parameter]
Wenn es auf `#t` gesetzt ist und `repository-directory` aktiviert ist, wird, wenn Repositorys mit einem Suffix von `.git` gefunden werden, dieses Suffix von der URL und dem Namen weggelassen.

Vorgegeben ist '`#f`'.

`Ganze-Zahl renamelimit` [`cgit-configuration`-Parameter]
Maximale Anzahl der Dateien, die bei der Erkennung von Umbenennungen berücksichtigt werden.

Die Vorgabe ist '`-1`'.

`Zeichenkette repository-sort` [`cgit-configuration`-Parameter]
Auf welche Art Repositorys in jedem Abschnitt sortiert werden.

Die Vorgabe ist '`""`'.

`Robots-Liste robots` [`cgit-configuration`-Parameter]
Text, der als Inhalt des `robots`-Meta-Tags dienen soll.
Die Vorgabe ist '`("noindex" "nofollow")`'.

`Zeichenkette root-desc` [`cgit-configuration`-Parameter]
Welcher Text unterhalb der Überschrift auf Repository-Index-Seiten ausgegeben wird.
Die Vorgabe ist '`"a fast webinterface for the git dscm"`'.

`Zeichenkette root-readme` [`cgit-configuration`-Parameter]
Der Inhalt der mit dieser Option angegebenen Datei wird wortwörtlich unter dem „about"-Link auf der Repository-Index-Seite angezeigt.
Die Vorgabe ist '`""`'.

`Zeichenkette root-title` [`cgit-configuration`-Parameter]
Welcher Text als Überschrift auf Repository-Index-Seiten ausgegeben werden soll.
Die Vorgabe ist '`""`'.

`Boolescher-Ausdruck scan-hidden-path` [`cgit-configuration`-Parameter]
Wenn es auf '`#t`' gesetzt ist und repository-directory aktiviert ist, wird repository-directory in Verzeichnissen rekursiv schauen, deren Name mit einem Punkt beginnt. Ansonsten werden solche Verzeichnisse unter repository-directory als „versteckt" betrachtet und ignoriert. Beachten Sie, dass dies keine Wirkung auf das `.git`-Verzeichnis in *nicht* auf „bare" gestellten Repositorys hat.
Vorgegeben ist '`#f`'.

`Liste snapshots` [`cgit-configuration`-Parameter]
Dieser Text gibt an, für welche Snapshot-Formate Cgit Links erzeugt.
Die Vorgabe ist '`()`'.

`Repository-Verzeichnis` [`cgit-configuration`-Parameter]
`repository-directory`
Der Name des Verzeichnisses, in dem nach Repositorys gesucht wird (wird als `scan-path` in die Einstellungen übernommen).
Die Vorgabe ist '`"/srv/git"`'.

`Zeichenkette section` [`cgit-configuration`-Parameter]
Der Name des aktuellen Abschnitts („section") für Repositorys – alle später definierten Repositorys werden den aktuellen Abschnittsnamen erben.
Die Vorgabe ist '`""`'.

`Zeichenkette section-sort` [`cgit-configuration`-Parameter]
Wenn diese Option auf '`#t`' gesetzt wird, werden die Abschnitte in Repository-Auflistungen nach Namen sortiert.
Die Vorgabe ist '`""`'.

`Ganze-Zahl section-from-path` [`cgit-configuration`-Parameter]
Wenn diese Zahl vor „repository-directory" definiert wurde, gibt sie an, wie viele Pfadelemente jedes Repository-Pfads für den Abschnittsnamen voreingestellt verwendet werden.
Die Vorgabe ist '`0`'.

`Boolescher-Ausdruck side-by-side-diffs?` [`cgit-configuration`-Parameter]
Wenn es auf '`#t`' gesetzt ist, werden Diffs nach Voreinstellung in Nebeneinanderdarstellung („side by side“) statt als zusammengeführte „Unidiffs“ angezeigt.

Vorgegeben ist '`#f`'.

`Dateiobjekt source-filter` [`cgit-configuration`-Parameter]
Gibt einen Befehl an, der aufgerufen wird, um Klartext-Blobs in der Baumansicht („tree“) zu formatieren.

Die Vorgabe ist '`""`'.

`Ganze-Zahl summary-branches` [`cgit-configuration`-Parameter]
Gibt die Anzahl der Branches an, die in der „summary“-Ansicht eines Repositorys zu sehen sein sollen.

Die Vorgabe ist '`10`'.

`Ganze-Zahl summary-log` [`cgit-configuration`-Parameter]
Gibt die Anzahl der Log-Einträge an, die in der „summary“-Ansicht eines Repositorys zu sehen sein sollen.

Die Vorgabe ist '`10`'.

`Ganze-Zahl summary-tags` [`cgit-configuration`-Parameter]
Gibt die Anzahl in der „summary“-Ansicht eines Repositorys anzuzeigender Tags an.

Die Vorgabe ist '`10`'.

`Zeichenkette strict-export` [`cgit-configuration`-Parameter]
Wenn dieser Dateiname angegeben wird, muss eine Datei diesen Namens in einem Repository enthalten sein, damit es angezeigt wird.

Die Vorgabe ist '`""`'.

`Zeichenkette virtual-root` [`cgit-configuration`-Parameter]
Wird diese URL angegeben, wird sie als Wurzel für alle Cgit-Links verwendet.

Die Vorgabe ist '`"/"`'.

`„repository-cgit-configuration“-Liste repositories` [`cgit-configuration`-Parameter]
Eine Liste von *cgit-repo*-Verbundsobjekten, die innerhalb der Konfiguration benutzt werden sollen.

Die Vorgabe ist '`()`'.

Verfügbare `repository-cgit-configuration`-Felder sind:

`Repo-Liste snapshots` [`repository-cgit-configuration`-Parameter]
Eine Maske, die für dieses Repository auf die Snapshots gelegt wird, für die Cgit Links erzeugt. Dadurch kann die globale Einstellung `snapshots` eingeschränkt werden.

Die Vorgabe ist '`()`'.

`Repo-Dateiobjekt source-filter` [`repository-cgit-configuration`-Parameter]
: Die Voreinstellung für `source-filter` ersetzen.

  Die Vorgabe ist '`""`'.

`Repo-Zeichenkette url` [`repository-cgit-configuration`-Parameter]
: Die relative URL, mit der auf das Repository zugegriffen wird.

  Die Vorgabe ist '`""`'.

`Repo-Dateiobjekt about-filter` [`repository-cgit-configuration`-Parameter]
: Die Voreinstellung für `about-filter` ersetzen.

  Die Vorgabe ist '`""`'.

`Repo-Zeichenkette branch-sort` [`repository-cgit-configuration`-Parameter]
: Wenn diese Option auf '`age`' gesetzt wurde, wird die Liste der Branch-Referenzen nach Datum sortiert, und wenn sie auf '`name`' gesetzt wurde, wird nach dem Branch-Namen sortiert.

  Die Vorgabe ist '`""`'.

`Repo-Liste clone-url` [`repository-cgit-configuration`-Parameter]
: Eine Liste von URLs, von denen das Repository geklont werden kann.

  Die Vorgabe ist '`()`'.

`Repo-Dateiobjekt commit-filter` [`repository-cgit-configuration`-Parameter]
: Die Voreinstellung für `commit-filter` ersetzen.

  Die Vorgabe ist '`""`'.

`Repo-Dateiobjekt commit-sort` [`repository-cgit-configuration`-Parameter]
: Wenn diese Option als '`date`' festgelegt wird, wird das Commit-Log streng nach Datum geordnet. Wenn sie auf '`topo`' gesetzt ist, wird es streng topologisch geordnet.

  Die Vorgabe ist '`""`'.

`Repo-Zeichenkette defbranch` [`repository-cgit-configuration`-Parameter]
: Der Name des voreingestellten Branches dieses Repositorys. Wenn kein solcher Branch im Repository existiert, wird der erste Branchname in sortierter Reihenfolge voreingestellt. Nach Vorgabe wird der Branch voreingestellt, auf den HEAD zeigt, oder „master“, wenn es keinen passenden HEAD gibt.

  Die Vorgabe ist '`""`'.

`Repo-Zeichenkette desc` [`repository-cgit-configuration`-Parameter]
: Der Wert, der als Repository-Beschreibung angezeigt werden soll.

  Die Vorgabe ist '`""`'.

`Repo-Zeichenkette` [`repository-cgit-configuration`-Parameter]
`homepage`
Der Wert, der als Repository-Homepage angezeigt werden soll.

Die Vorgabe ist '`""`'.

`Repo-Dateiobjekt` [`repository-cgit-configuration`-Parameter]
`email-filter`
Die Voreinstellung für `email-filter` ersetzen.

Die Vorgabe ist '`""`'.

`Vielleicht-Repo-Boolescher-Ausdruck` [`repository-cgit-configuration`-Parameter]
`enable-commit-graph?`
Eine Option, mit der die globale Einstellung für `enable-commit-graph?` deaktiviert werden kann.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Repo-Boolescher-Ausdruck` [`repository-cgit-configuration`-Parameter]
`enable-log-filecount?`
Eine Option, mit der die globale Einstellung für `enable-log-filecount?` deaktiviert werden kann.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Repo-Boolescher-Ausdruck` [`repository-cgit-configuration`-Parameter]
`enable-log-linecount?`
Eine Option, mit der die globale Einstellung für `enable-log-linecount?` deaktiviert werden kann.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Repo-Boolescher-Ausdruck` [`repository-cgit-configuration`-Parameter]
`enable-remote-branches?`
Wenn diese Option auf '`#t`' gesetzt ist, zeigt Cgit unter den „summary"- und „ref"-Seiten entfernte Branches an.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Repo-Boolescher-Ausdruck` [`repository-cgit-configuration`-Parameter]
`enable-subject-links?`
Eine Option, mit der die globale Einstellung für `enable-subject-links?` deaktiviert werden kann.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Vielleicht-Repo-Boolescher-Ausdruck` [`repository-cgit-configuration`-Parameter]
`enable-html-serving?`
Eine Option, mit der die globale Einstellung für `enable-html-serving?` deaktiviert werden kann.

Der Vorgabewert ist '`disabled`' (d.h. deaktiviert).

`Repo-Boolescher-Ausdruck hide?` [`repository-cgit-configuration`-Parameter]

Wenn diese Option auf `#t` gesetzt ist, wird das Repository im Repository-Index verborgen.

Vorgegeben ist '`#f`'.

`Repo-Boolescher-Ausdruck ignore?` [`repository-cgit-configuration`-Parameter]

Wenn diese Option auf `#t` gesetzt ist, wird das Repository ignoriert.

Vorgegeben ist '`#f`'.

`Repo-Dateiobjekt logo` [`repository-cgit-configuration`-Parameter]

URL, unter der ein Bild zu finden ist, das auf allen Seiten dieses Repositorys als Logo zu sehen sein wird.

Die Vorgabe ist '`""`'.

`Repo-Zeichenkette logo-link` [`repository-cgit-configuration`-Parameter]

URL, die geladen wird, wenn jemand auf das Logo-Bild klickt.

Die Vorgabe ist '`""`'.

`Repo-Dateiobjekt owner-filter` [`repository-cgit-configuration`-Parameter]

Die Voreinstellung für `owner-filter` ersetzen.

Die Vorgabe ist '`""`'.

`Repo-Zeichenkette module-link` [`repository-cgit-configuration`-Parameter]

Text, der als Formatzeichenkette für einen Hyperlink benutzt wird, wenn in einer Verzeichnisauflistung ein Submodul ausgegeben wird. Die Argumente für diese Formatzeichenkette sind Pfad und SHA1 des Submodul-Commits.

Die Vorgabe ist '`""`'.

`module-link-Pfad module-link-path` [`repository-cgit-configuration`-Parameter]

Text, der als Formatzeichenkette für einen Hyperlink benutzt wird, wenn in einer Verzeichnisauflistung ein Submodul mit dem angegebenen Unterverzeichnispfad ausgegeben wird.

Die Vorgabe ist '`()`'.

`Repo-Zeichenkette max-stats` [`repository-cgit-configuration`-Parameter]

Die Voreinstellung für den maximalen Zeitraum für Statistiken ersetzen.

Die Vorgabe ist '`""`'.

`Repo-Zeichenkette name` [`repository-cgit-configuration`-Parameter]

Welcher Wert als Repository-Name angezeigt werden soll.

Die Vorgabe ist '`""`'.

`Repo-Zeichenkette owner` [`repository-cgit-configuration`-Parameter]
Ein Wert, um den Besitzer des Repositorys zu identifizieren.

Die Vorgabe ist ‘`""`’.

`Repo-Zeichenkette path` [`repository-cgit-configuration`-Parameter]
Ein absoluter Pfad zum Repository-Verzeichnis.

Die Vorgabe ist ‘`""`’.

`Repo-Zeichenkette readme` [`repository-cgit-configuration`-Parameter]
Ein Pfad (relativ zum Repository), der eine Datei angibt, deren Inhalt wortwörtlich auf der „About“-Seite dieses Repositorys eingefügt werden soll.

Die Vorgabe ist ‘`""`’.

`Repo-Zeichenkette section` [`repository-cgit-configuration`-Parameter]
Der Name des aktuellen Abschnitts („section“) für Repositorys – alle später definierten Repositorys werden den aktuellen Abschnittsnamen erben.

Die Vorgabe ist ‘`""`’.

`Repo-Liste extra-options` [`repository-cgit-configuration`-Parameter]
Zusätzliche Optionen werden an die cgitrc-Datei angehängt.

Die Vorgabe ist ‘`()`’.

`Liste extra-options` [`cgit-configuration`-Parameter]
Zusätzliche Optionen werden an die cgitrc-Datei angehängt.

Die Vorgabe ist ‘`()`’.

Aber es könnte auch sein, dass Sie schon eine `cgitrc` haben und zum Laufen bringen wollen. In diesem Fall können Sie eine `opaque-cgit-configuration` als Verbundsobjekt an `cgit-service-type` übergeben. Wie der Name schon sagt, bietet eine opake Konfiguration keinerlei Unterstützung für Reflexion.

Verfügbare `opaque-cgit-configuration`-Felder sind:

`„package“ cgit` [`opaque-cgit-configuration`-Parameter]
Das cgit-Paket.

`Zeichenkette string` [`opaque-cgit-configuration`-Parameter]
Der Inhalt für `cgitrc` als eine Zeichenkette.

Wenn zum Beispiel Ihre `cgitrc` nur aus der leeren Zeichenkette bestehen soll, könnten Sie einen Cgit-Dienst auf diese Weise instanziieren:

```
(service cgit-service-type
         (opaque-cgit-configuration
          (cgitrc "")))
```

Gitolite-Dienst

Gitolite (`https://gitolite.com/gitolite/`) ist ein Werkzeug, um Git-Repositorys anderen auf einem zentralen Server anzubieten.

Gitolite kann mehrere Nutzer mit mehreren Repositorys bedienen und unterstützt flexible Konfigurationsmöglichkeiten der Berechtigungen der Repository-Nutzer.

Das folgende Beispiel richtet Gitolite für den voreingestellten `git`-Benutzer und den angegebenen öffentlichen SSH-Schlüssel ein.

```
(service gitolite-service-type
         (gitolite-configuration
           (admin-pubkey (plain-file
                          "ihrname.pub"
                          "ssh-rsa AAAA... guix@example.com"))))
```

Sie konfigurieren Gitolite, indem Sie ein besonderes Admin-Repository anpassen. Sie können es zum Beispiel klonen, indem Sie, wenn Sie Gitolite auf `example.com` eingerichtet haben, den folgenden Befehl zum Klonen des Admin-Repositorys ausführen:

```
git clone git@example.com:gitolite-admin
```

Wenn der Gitolite-Dienst aktiviert wird, wird der mitgegebene `admin-pubkey` ins `keydir`-Verzeichnis vom „gitolite-admin“-Repository eingefügt. Wenn sich dadurch das Repository ändert, wird die Änderung mit der Commit-Nachricht „gitolite setup by GNU Guix“ commitet.

`gitolite-configuration` [Datentyp]

Repräsentiert die Konfiguration vom `gitolite-service-type`.

`package` (Vorgabe: *gitolite*)
: Welches Gitolite-Paket benutzt werden soll. Sie können hiermit auch optionale Abhängigkeiten von Gitolite, die *nicht* im Standard-Gitolite-Paket enthalten sind, hinzufügen, wie Redis und git-annex. Diese Funktionalitäten können Sie mit der Prozedur `make-gitolite` aus dem Modul `(gnu packages version-control)` verfügbar machen, welche eine Variante von Gitolite erzeugt, bei der die gewünschten Abhängigkeiten vorhanden sind.

  Folgender Code liefert ein Paket zurück, wo die Programme Redis und git-annex durch die Skripte von Gitolite aufrufbar sind:

```
(use-modules (gnu packages databases)
             (gnu packages haskell-apps)
             (gnu packages version-control))
(make-gitolite (list redis git-annex))
```

`user` (Vorgabe: *git*)
: Welches Benutzerkonto für Gitolite benutzt werden soll. Mit diesem Benutzer werden Sie über SSH auf Gitolite zugreifen.

`group` (Vorgabe: *git*)
: Gruppe für Gitolite.

`home-directory` (Vorgabe: "*/var/lib/gitolite*")
: Das Verzeichnis, in dem die Gitolite-Konfiguration und Repositorys gespeichert werden sollen.

`rc-file` (Vorgabe: *(gitolite-rc-file)*)
: Ein dateiartiges Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), das die Konfiguration für Gitolite repräsentiert.

`admin-pubkey` (Vorgabe: *#f*)
: Ein dateiartiges Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), mit dem Gitolite eingerichtet werden kann. Er wird in das `keydir`-Verzeichnis im „gitolite-admin“-Repository eingefügt.

  Um einen SSH-Schlüssel als Zeichenkette anzugeben, benutzen Sie die `plain-file`-Funktion.

```
(plain-file "ihrname.pub" "ssh-rsa AAAA... guix@example.com")
```

`gitolite-rc-file` [Datentyp]

Repräsentiert die Gitolie-RC-Datei.

`umask` (Vorgabe: `#o0077`)
: Dies legt fest, welche Berechtigungen Gitolite an die Repositorys und deren Inhalt vergibt.

  Für einen Wert wie `#o0027` wird die Gruppe, die Gitolite benutzt (nach Vorgabe: `git`) Lesezugriff erhalten. Das ist nötig, wenn Sie Gitolite mit Software wie Cgit oder Gitweb kombinieren.

`local-code` (Vorgabe: `"$rc{GL_ADMIN_BASE}/local"`)
: Hiermit können Sie eigene Programme zu den als „non-core“ eingestuften hinzufügen oder sogar die mitgelieferten Programme durch Ihre eigenen ersetzen.

  Bitte geben Sie für diese Variable den vollständigen Pfad an. Vorgegeben ist, dass das Verzeichnis namens "local" in Ihrem Klon des Gitolite-Repositorys benutzt wird. Dadurch haben Sie den Vorteil, dass alles versioniert wird, und können Änderungen vornehmen, ohne sich auf dem Server anmelden zu müssen.

`unsafe-pattern` (Vorgabe: `#f`)
: Optional ein Perl-kompatibler regulärer Ausdruck, um unsichere Konfigurationen in der Konfigurationsdatei zu unterbinden. Siehe die Dokumentation von Gitolite (`https://gitolite.com/gitolite/git-config.html#compensating-for-unsafe_patt`) für weitere Informationen.

  Wenn als Wert *nicht* `#f` angegeben wird, muss es eine Zeichenkette mit einem Perl-kompatiblen regulären Ausdruck sein wie ‘`"[`~#\$\&()|;<>]"`’, was der Voreinstellung von Gitolite entspricht. Damit wird unterbunden, jegliche Sonderzeichen in die Konfiguration zu schreiben, die von einer Shell interpretiert werden könnten, damit die Administration auch auf andere Leute übertragen werden kann, die sonst keinen Shell-Zugriff auf dem Server haben.

`git-config-keys` (Vorgabe: `""`)
: Mit Gitolite können Sie Werte für Git-Konfigurationen über das '`config`'-Schlüsselwort festlegen. Mit dieser Einstellung können Sie steuern, welche Konfigurationsschlüssel akzeptiert werden.

`roles` (Vorgabe: `'(("READERS" . 1) ("WRITERS" . ))`)
: Legt fest, welche Rollennamen für Nutzer möglich sind, wenn Sie den Befehl perms ausführen.

`enable` (Vorgabe: `'("help" "desc" "info" "perms" "writable" "ssh-authkeys" "git-config" "daemon" "gitweb")`)
: Diese Einstellung legt die innerhalb von Gitolite zur Verfügung gestellten Befehle fest.

Gitile-Dienst

Mit Gitile (`https://git.lepiller.eu/gitile`) kann eine eigene Git-„Forge“ aufgesetzt werden, wo sich jeder über einen Web-Browser den Inhalt von Git-Repositorys ansehen kann.

Am besten kombiniert man Gitile mit Gitolite, und Gitile bietet nach Voreinstellung die für die Öffentlichkeit bestimmten Repositorys von Gitolite an. Der Dienst sollte auf einem lokalen Port lauschen, und ein Webserver sollte eingerichtet werden, um Gitiles statische Ressourcen anzubieten. Für den Gitile-Dienst kann der Nginx-Dienst einfach zu diesem Zweck erweitert werden (siehe [NGINX], Seite 469).

Im Folgenden sehen Sie ein Beispiel, wie Gitile konfiguriert wird, um Repositorys von einem selbst ausgewählten Ort anzubieten, zusammen mit etwas Text für die Hauptseite und Fußzeilen.

```
(service gitile-service-type
         (gitile-configuration
           (repositories "/srv/git")
           (base-git-url "https://meineweb.site/git")
           (index-title "Meine Git-Repositorys")
           (intro '((p "Hier sind all meine öffentlichen Werke!")))
           (footer '((p "Nun ist Schluss")))
           (nginx-server-block
             (nginx-server-configuration
               (ssl-certificate
                 "/etc/letsencrypt/live/meineweb.site/fullchain.pem")
               (ssl-certificate-key
                 "/etc/letsencrypt/live/meineweb.site/privkey.pem")
               (listen '("443 ssl http2" "[::]:443 ssl http2"))
               (locations
                 (list
                   ;; Anonymen Abruf über HTTPS von „/git/“-URLs zulassen.
                   (git-http-nginx-location-configuration
                     (git-http-configuration
                       (uri-path "/git/")
                       (git-root "/var/lib/gitolite/repositories")))))))))
```

Neben dem Konfigurations-Verbundsobjekt sollten Sie auch bei Ihren Git-Repositorys ein paar optionale Informationen darüber eintragen. Zunächst werden Repositorys nur für die Allgemeinheit zugänglich, wenn sie die magische Datei `git-daemon-export-ok` enthalten. Gitile erkennt daran, welche Repositorys öffentlich sind und zugänglich gemacht werden sollen. Bei Gitolite heißt das zum Beispiel, Sie ändern Ihre `conf/gitolite.conf`, dass dort für die öffentlich zu machenden Repositorys Folgendes steht:

```
repo foo
    R = daemon
```

Zusätzlich kann Gitile aus der Repository-Konfiguration weitere Informationen über das Repository auslesen und anzeigen. Gitile verwendet für seine Konfiguration den gitweb-Namensraum. Zum Beispiel kann Ihre `conf/gitolite.conf` Folgendes enthalten:

```
repo foo
    R = daemon
    desc = Eine lange Beschreibung, gerne auch mit <i>HTML</i>-Tags, für die Seite mit
    config gitweb.name = Das Projekt Foo
    config gitweb.synopsis = Eine kurze Beschreibung für die Hauptseite des Projekts
```

Denken Sie daran, die Änderungen auch zu commiten und zu pushen, sobald Sie zufrieden sind. Vielleicht müssen Sie Ihre gitolite-Konfiguration anpassen, damit die genannten Konfigurationsoptionen zulässig sind. Eine Möglichkeit ist folgende Dienstdefinition:

```
(service gitolite-service-type
         (gitolite-configuration
           (admin-pubkey (local-file "key.pub"))
           (rc-file
             (gitolite-rc-file
               (umask #o0027)
               ;; Man soll jeden Konfigurationsschlüssel setzen dürfen
               (git-config-keys ".*")
               ;; Jeder Text soll ein gültiger Konfigurationswert sein
               (unsafe-patt "^$")))))
```

`gitile-configuration` [Datentyp]

Repräsentiert die Konfiguration vom `gitile-service-type`.

`package` (Vorgabe: *gitile*)
: Welches Gitile-Paket benutzt werden soll.

`host` (Vorgabe: `"localhost"`)
: Der Rechnername, auf den Gitile lauscht.

`port` (Vorgabe: `8080`)
: Der Port, auf dem Gitile lauscht.

`database` (Vorgabe: `"/var/lib/gitile/gitile-db.sql"`)
: Wo die Datenbank liegt.

`repositories` (Vorgabe: `"/var/lib/gitolite/repositories"`)
: Wo die Repositorys zu finden sind. Beachten Sie: Gitile zeigt nur öffentliche Repositorys an. Um ein Repository öffentlich zu machen, fügen Sie eine leere Datei `git-daemon-export-ok` in das Wurzelverzeichnis des Repositorys ein.

`base-git-url`
: Die Basis-Git-URL, von der angezeigt wird, dass man Befehle zum Klonen darauf ausführen kann.

`index-title` (Vorgabe: `"Index"`)
: Der Seitentitel der Index-Seite mit der Liste aller verfügbaren Repositorys.

`intro` (Vorgabe: `'()`)
: Der Inhalt der Einführung als Liste von SXML-Ausdrücken. Sie wird über der Liste der Repositorys auf der Index-Seite angezeigt.

`footer` (Vorgabe: `'()`)
: Der Inhalt der Fußzeile als Liste von SXML-Ausdrücken. Sie wird unter jeder von Gitile angebotenen Seite angezeigt.

`nginx-server-block`
: Ein Server-Block für NGinx. NGinx wird durch Gitile erweitert und als inverser Proxy („Reverse Proxy“) verwendet, um dessen Seiten anzubieten, und es wird als einfacher Web-Server verwendet, um statische Dateien und Grafiken anzubieten.

  Über diesen Block können Sie weitere eigene URLs auf Ihrer Domain anbieten lassen wie z.B. eine `/git/`-URL zum „anonymen“ Klonen ohne Anmeldung oder sonstige Dateien, die Sie anbieten möchten.

12.9.31 Spieldienste

„The Battle for Wesnoth“-Dienst

The Battle for Wesnoth (`https://wesnoth.org`) ist ein in einer Fantasy-Welt angesiedeltes, rundenbasiertes, taktisches Strategiespiel. Es verfügt über mehrere Einzelspielerkampagnen und Mehrspielerspiele (über das Netzwerk und lokal).

`Scheme-Variable` *wesnothd-service-type* [Variable]
: Diensttyp für den wesnothd-Dienst. Als Wert muss ein `wesnothd-configuration`-Objekt benutzt werden. Um wesnothd mit seiner Vorgabekonfiguration auszuführen, instanziieren Sie es als:

```
(service wesnothd-service-type)
```

`wesnothd-configuration` [Datentyp]
: Datentyp, der die Konfiguration von `wesnothd` repräsentiert.

  `package` (Vorgabe: `wesnoth-server`)
  : Das Paket, das für den Wesnoth-Server benutzt werden soll.

  `port` (Vorgabe: `15000`)
  : Der Port, an den der Server gebunden wird.

12.9.32 PAM-Einbindedienst

Das Modul `(gnu services pam-mount)` stellt einen Dienst zur Verfügung, mit dem Benutzer Datenträger beim Anmelden einbinden können. Damit sollte es möglich sein, jedes vom System unterstützte Datenträgerformat einzubinden.

`Scheme-Variable` *pam-mount-service-type* [Variable]
Diensttyp für PAM-Einbindeunterstützung.

`pam-mount-configuration` [Datentyp]
Datentyp, der die Konfiguration für PAM-Einbindungen („PAM Mount“) repräsentiert.

Sie hat folgende Parameter:

`rules` Die Konfigurationsregeln, um `/etc/security/pam_mount.conf.xml` zu erzeugen.

Die Konfigurationsregeln sind SXML-Elemente (siehe Abschnitt “SXML” in *Referenzhandbuch zu GNU Guile*) und nach Vorgabe wird für niemanden etwas beim Anmelden eingebunden:

```
`((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"))))
```

Es müssen `volume`-Elemente eingefügt werden, um Datenträger automatisch bei der Anmeldung einzubinden. Hier ist ein Beispiel, mit dem die Benutzerin `alice` ihr verschlüsseltes `HOME`-Verzeichnis einbinden kann, und der Benutzer `bob` die Partition einbinden kann, wo er seine Daten abspeichert.

```
(define pam-mount-rules
`((debug (@ (enable "0")))
            (volume (@ (user "alice")
                       (fstype "crypt")
                       (path "/dev/sda2")
                       (mountpoint "/home/alice")))
            (volume (@ (user "bob")
                       (fstype "auto")
                       (path "/dev/sdb3")
                       (mountpoint "/home/bob/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 pam-mount-rules)))
```

Die vollständige Liste möglicher Optionen finden Sie in der Handbuchseite („man page“) für pam_mount.conf (`http://pam-mount.sourceforge.net/pam_mount.conf.5.html`).

12.9.33 Guix-Dienste

Guix-Erstellungskoordinator

Der Guix-Erstellungskoordinator (`https://git.cbaines.net/guix/build-coordinator/`) („Guix Build Coordinator“) hilft dabei, Erstellungen von Ableitungen auf Maschinen zu verteilen, auf denen ein *Agent* läuft. Der Erstellungs-Daemon wird weiterhin benutzt, um die Ableitungen zu erstellen, der Guix-Erstellungskoordinator verwaltet nur die Zuteilung der Erstellungen und den Umgang mit den Ergebnissen.

Der Guix-Erstellungskoordinator setzt sich aus einem *Koordinator* und mindestens einem verbundenen *Agentenprozess* zusammen. Der Koordinatorprozess kümmert sich um Clients, die Erstellungen einreichen, und teilt Erstellungen den Agenten zu. Die Agentenprozesse kommunizieren mit einem Erstellungs-Daemon, welcher die Erstellungen eigentlich durchführt und die Ergebnisse an den Koordinator zurückgibt.

Es gibt ein Skript, um die Koordinatorkomponente des Guix-Erstellungskoordinators auszuführen, aber der Dienst für Guix benutzt stattdessen sein eigenes Guile-Skript, was in der Konfiguration benutzte G-Ausdrücke besser integriert.

`Scheme-Variable` *guix-build-coordinator-service-type* [Variable]
: Diensttyp für den Guix-Erstellungskoordinator. Sein Wert muss ein `guix-build-coordinator-configuration`-Objekt sein.

`guix-build-coordinator-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration des Guix-Erstellungskoordinators repräsentiert.

  `package` (Vorgabe: `guix-build-coordinator`)
  : Das zu verwendende Guix-Erstellungskoordinator-Paket.

  `user` (Vorgabe: `"guix-build-coordinator"`)
  : Das Systembenutzerkonto, mit dem der Dienst ausgeführt wird.

  `group` (Vorgabe: `"guix-build-coordinator"`)
  : Die Systembenutzergruppe, mit der der Dienst ausgeführt wird.

`database-uri-string` (Vorgabe: `"sqlite:///var/lib/guix-build-coordinator/guix_build_coordinator.db"`)
: Die URI für die Datenbank.

`agent-communication-uri` (Vorgabe: `"http://0.0.0.0:8745"`)
: Die URI, die beschreibt, wie auf Anfragen von Agentenprozessen gelauscht wird.

`client-communication-uri` (Vorgabe: `"http://127.0.0.1:8746"`)
: Die URI, die beschreibt, wie auf Anfragen von Clients gelauscht wird. Mit der Client-API können Erstellungen eingereicht werden und derzeit findet dabei keine Authentifizierung statt. Das sollten Sie bedenken, wenn Sie hieran Einstellungen vornehmen.

`allocation-strategy` (Vorgabe: `#~basic-build-allocation-strategy`)
: Ein G-Ausdruck für die zu nutzende Zuteilungsstrategie. Angegeben wird eine Prozedur, die den Datenspeicher als Argument nimmt und in den Zuteilungsplan in der Datenbank einfügt.

`hooks` (Vorgabe: *'()*)
: Eine assoziative Liste mit Hooks. Mit ihnen kann infolge bestimmter Ereignisse beliebiger Code ausgeführt werden, z.B. wenn ein Erstellungsergebnis verarbeitet wird.

`parallel-hooks` (Vorgabe: *'()*)
: Sie können Hooks so einstellen, dass sie parallel ausgeführt werden. Mit diesem Parameter geben Sie eine assoziative Liste an, welche der Hooks parallel ausgeführt werden sollen. Der Schlüssel ist das Symbol für den Hook und der Wert ist die Anzahl Threads, die ausgeführt werden.

`guile` (Vorgabe: `guile-3.0-latest`)
: Mit welchem Guile-Paket der Guix-Erstellungskoordinator ausgeführt wird.

`Scheme-Variable` *guix-build-coordinator-agent-service-type* [Variable]
Diensttyp für einen Guix-Erstellungskoordinator-Agenten. Als Wert muss ein `guix-build-coordinator-agent-configuration`-Objekt angegeben werden.

`guix-build-coordinator-agent-configuration` [Datentyp]
Der Datentyp, der die Konfiguration eines Guix-Erstellungskoordinator-Agenten repräsentiert.

`package` (Vorgabe: `guix-build-coordinator/agent-only`)
: Das zu verwendende Guix-Erstellungskoordinator-Paket.

`user` (Vorgabe: `"guix-build-coordinator-agent"`)
: Das Systembenutzerkonto, mit dem der Dienst ausgeführt wird.

`coordinator` (Vorgabe: `"http://localhost:8745"`)
: Das Passwort, das zum Herstellen einer Verbindung mit dem Koordinator verwendet werden soll.

`authentication`
: Ein Verbundsobjekt, das beschreibt, wie sich dieser Agent beim Koordinator authentisiert. Mögliche Verbundsfelder werden im Folgenden beschrieben.

`systems` (Vorgabe: `#f`)
: Die Systeme, für die dieser Agent Erstellungen übernehmen soll. Voreingestellt verwendet der Agentenprozess den Systemtyp, auf dem er aktuell läuft.

`max-parallel-builds` (Vorgabe: `1`)
: Die Anzahl der Erstellungen, die parallel ausgeführt werden sollen.

`max-allocated-builds` (Vorgabe: `#f`)
: Wie viele Erstellungen diesem Agenten höchstens zugewiesen werden können.

`max-1min-load-average` (Vorgabe: `#f`)
: Die durchschnittliche Last (Load Average), die bestimmt, ob der Koordinatoragent mit neuen Erstellungen beginnt. Wenn die Load Average über 1 Minute den angegebenen Wert übersteigt, wartet der Agent und beginnt keine zusätzlichen Erstellungen.

  Wenn als Wert `#f` angegeben wird, wird die Einstellung nicht festgelegt und der Agent verwendet die Anzahl der Kerne, die das System meldet, als maximaler Load Average über 1 Minute.

`derivation-substitute-urls` (Vorgabe: `#f`)
: URLs, von denen versucht werden soll, Substitute für Ableitungen herunterzuladen, wenn die Ableitungen noch nicht vorliegen.

`non-derivation-substitute-urls` (Vorgabe: `#f`)
: URLs, von denen versucht werden soll, Substitute für Erstellungseingaben herunterzuladen, wenn die Eingabe-Store-Objekte noch nicht vorliegen.

`guix-build-coordinator-agent-password-auth` [Datentyp]
: Der Datentyp, der einen Agenten repräsentiert, der sich bei einem Koordinator über eine UUID und ein Passwort authentisiert.

  `uuid` Die UUID des Agenten. Sie sollte durch den Koordinatorprozess erzeugt worden sein, in der Datenbank des Koordinators eingetragen sein und von dem Agenten benutzt werden, für den sie gedacht ist.

  `password` Das Passwort, das zum Herstellen einer Verbindung mit dem Koordinator verwendet werden soll.

`guix-build-coordinator-agent-password-file-auth` [Datentyp]
: Der Datentyp, der einen Agenten repräsentiert, der sich bei einem Koordinator über eine UUID und ein aus einer Datei gelesenes Passwort authentisiert.

  `uuid` Die UUID des Agenten. Sie sollte durch den Koordinatorprozess erzeugt worden sein, in der Datenbank des Koordinators eingetragen sein und von dem Agenten benutzt werden, für den sie gedacht ist.

`password-file`
: Eine Datei mit dem Passwort, um sich mit dem Koordinator zu verbinden.

`guix-build-coordinator-agent-dynamic-auth` [Datentyp]
: Der Datentyp, der einen Agenten repräsentiert, der sich bei einem Koordinator über einen dynamischen Authentisierungs-Token und den Agentennamen authentisiert.

`agent-name`
: Der Name des Agenten. Er wird benutzt, um den passenden Eintrag in der Datenbank für ihn zu finden. Gibt es noch keinen Eintrag, wird automatisch einer hinzugefügt.

`token`
: Dynamischer Authentisierungs-Token. Er wird in der Datenbank des Koordinators erzeugt und vom Agenten zur Authentisierung benutzt.

`guix-build-coordinator-agent-dynamic-auth-with-file` [Datentyp]
: Der Datentyp, der einen Agenten repräsentiert, der sich bei einem Koordinator über einen aus einer Datei gelesenen dynamischen Authentisierungs-Token und den Agentennamen authentisiert.

`agent-name`
: Der Name des Agenten. Er wird benutzt, um den passenden Eintrag in der Datenbank für ihn zu finden. Gibt es noch keinen Eintrag, wird automatisch einer hinzugefügt.

`token-file`
: Eine Datei, in der der dynamische Authentisierungs-Token enthalten ist. Er wird in der Datenbank des Koordinators erzeugt und vom Agenten zur Authentisierung benutzt.

Das Paket des Guix-Erstellungskoordinators enthält ein Skript, mit dem eine Instanz des Guix-Datendienstes („Guix Data Service“) nach zu erstellenden Ableitungen angefragt werden kann, woraufhin Erstellungen für selbige Ableitungen beim Koordinator eingereicht werden. Der folgende Diensttyp hilft dabei, dieses Skript einzusetzen. Es ist ein weiteres Werkzeug, das beim Erstellen von Ableitungen von Nutzen sein kann, die in einer Instanz des Guix-Datendienstes vorhanden sind.

`Scheme-Variable` *guix-build-coordinator-queue-builds-service-type* [Variable]
: Der Diensttyp für das Skript guix-build-coordinator-queue-builds-from-guix-data-service. Als Wert muss ein `guix-build-coordinator-queue-builds-configuration`-Objekt benutzt werden.

`guix-build-coordinator-queue-builds-configuration` [Datentyp]
: Der Datentyp, der die Optionen an das Skript guix-build-coordinator-queue-builds-from-guix-data-service repräsentiert.

`package` (Vorgabe: `guix-build-coordinator`)
: Das zu verwendende Guix-Erstellungskoordinator-Paket.

`user` (Vorgabe: `"guix-build-coordinator-queue-builds"`)
: Das Systembenutzerkonto, mit dem der Dienst ausgeführt wird.

`coordinator` (Vorgabe: `"http://localhost:8746"`)
: Das Passwort, das zum Herstellen einer Verbindung mit dem Koordinator verwendet werden soll.

`systems` (Vorgabe: `#f`)
: Für welche Systeme zu erstellende Ableitungen übernommen werden sollen.

`systems-and-targets` (Vorgabe: `#f`)
: Eine assoziative Liste aus Paaren aus System und Zielplattform, für die zu erstellende Ableitungen übernommen werden sollen.

`guix-data-service` (Vorgabe: `"https://data.guix.gnu.org"`)
: Die Instanz des Guix-Datendienstes, die nach zu erstellenden Ableitungen angefragt werden soll.

`guix-data-service-build-server-id` (Vorgabe: `#f`)
: Die Server-ID des Erstellungsservers des Guix-Datendienstes, wo die Erstellungen stattfinden. So wird das Einreichen von Erstellungen beschleunigt, weil für bereits erstellte Ableitungen dann keine Erstellungsanfrage an den Koordinator mehr nötig ist.

`processed-commits-file` (Vorgabe: `"/var/cache/guix-build-coordinator-queue-builds/processed-commits"`)
: Eine Datei, in die gespeichert wird, welche Commits bereits verarbeitet wurden, um sie nicht unnötig ein weiteres Mal zu verarbeiten, wenn der Dienst neu gestartet wird.

Guix-Datendienst

Der Guix-Datendienst (`http://data.guix.gnu.org`) („Guix Data Service“) verarbeitet und speichert Daten über GNU Guix und stellt diese zur Verfügung. Dazu gehören Informationen über Pakete, Ableitungen sowie durch Linting erkannte Paketfehler.

Die Daten werden in einer PostgreSQL-Datenbank gespeichert und stehen über eine Weboberfläche zur Verfügung.

`Scheme-Variable` *guix-data-service-type* [Variable]
: Diensttyp für den Guix-Datendienst. Sein Wert muss ein `guix-data-service-configuration`-Objekt sein. Der Dienst kann optional den getmail-Dienst erweitern und die guix-commits-Mailing-Liste benutzen, um bei Änderungen am Guix-Git-Repository auf dem Laufenden zu bleiben.

`guix-data-service-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration des Guix-Datendienstes repräsentiert.

  `package` (Vorgabe: `guix-data-service`)
  : Das zu verwendende Guix-Datendienst-Paket.

  `user` (Vorgabe: `"guix-data-service"`)
  : Das Systembenutzerkonto, mit dem der Dienst ausgeführt wird.

  `group` (Vorgabe: `"guix-data-service"`)
  : Die Systembenutzergruppe, mit der der Dienst ausgeführt wird.

`port` (Vorgabe: `8765`)
: Der Port, an den der Webdienst gebunden wird.

`host` (Vorgabe: `"127.0.0.1"`)
: Rechnername oder Netzwerkschnittstelle, an die der Webdienst gebunden wird.

`getmail-idle-mailboxes` (Vorgabe: `#f`)
: Wenn es festgelegt ist, wird es als Liste der Postfächer („Mailboxes“) eingerichtet, die der getmail-Dienst beobachtet.

`commits-getmail-retriever-configuration` (Vorgabe: `#f`)
: Wenn es festgelegt ist, bezeichnet dies das `getmail-retriever-configuration`-Objekt, mit dem getmail eingerichtet wird, um E-Mails von der „guix-commits“-Mailing-Liste zu beziehen.

`extra-options` (Vorgabe: *'()*)
: Zusätzliche Befehlszeilenoptionen für `guix-data-service`.

`extra-process-jobs-options` (Vorgabe: *'()*)
: Zusätzliche Befehlszeilenoptionen für `guix-data-service-process-jobs`.

Nar Herder

Der Nar Herder (`https://git.cbaines.net/guix/nar-herder/about/`) ist ein Werkzeug zum Umgang mit einer Menge von Nars.

`Scheme-Variable` *nar-herder-type* [Variable]
: Ein Diensttyp für den Guix-Datendienst. Sein Wert muss ein `nar-herder-configuration`-Objekt sein. Der Dienst kann optional den getmail-Dienst erweitern und die guix-commits-Mailing-Liste benutzen, um bei Änderungen am Guix-Git-Repository auf dem Laufenden zu bleiben.

`nar-herder-configuration` [Datentyp]
: Der Datentyp, der die Konfiguration des Guix-Datendienstes repräsentiert.

`package` (Vorgabe: `nar-herder`)
: Das zu verwendende Nar-Herder-Paket.

`user` (Vorgabe: `"nar-herder"`)
: Das Systembenutzerkonto, mit dem der Dienst ausgeführt wird.

`group` (Vorgabe: `"nar-herder"`)
: Die Systembenutzergruppe, mit der der Dienst ausgeführt wird.

`port` (Vorgabe: `8734`)
: Der Port, an den der Server gebunden wird.

`host` (Vorgabe: `"127.0.0.1"`)
: Rechnername oder Netzwerkschnittstelle, an die der Server gebunden wird.

`mirror` (Vorgabe: `#f`)
: Optional die URL der anderen Nar-Herder-Instanz, wenn diese gespiegelt werden soll. Das bedeutet, die hiesige Nar-Herder-Instanz wird die Datenbank von dort herunterladen und auf dem aktuellen Stand halten.

`database` (Vorgabe: `"/var/lib/nar-herder/nar_herder.db"`)
: Wo die Datenbank gespeichert wird. Wenn die hiesige Nar-Herder-Instanz eine von anderswo spiegelt, wird die Datenbank von dort heruntergeladen, wenn noch keine existiert. Wenn keine andere Nar-Herder-Instanz gespiegelt wird, wird eine leere Datenbank angelegt.

`database-dump` (Vorgabe: `"/var/lib/nar-herder/nar_herder_dump.db"`)
: Der Ort für das Datenbank-Dump. Es wird erzeugt und regelmäßig aktualisiert, indem eine Kopie der Datenbank gemacht wird. Es handelt sich um die Version der Datenbank, die zum Herunterladen angeboten wird.

`storage` (Vorgabe: `#f`)
: Optional der Ort, wo Nars gespeichert werden.

`storage-limit` (Vorgabe: `"none"`)
: Wie viele Bytes die am Speicherort hinterlegten Nars höchstens einnehmen. Wenn „none“ angegeben wird, gibt es keine Begrenzung.

  Wenn am Speicherort mehr hinterlegt ist als hier angegeben ist, werden Nars gemäß `storage-nar-removal-criteria` gelöscht.

`storage-nar-removal-criteria` (Vorgabe: `'()`)
: Nach welchen Kriterien Nars aus dem Speicherort gelöscht werden. Sie werden in Verbindung mit `storage-limit` angewandt.

  Wenn die Größe am Speicherort `storage-limit` überschreitet, werden Nars gemäß der hier angegebenen Löschkriterien überprüft und, wenn irgendein Kriterium erfüllt wird, gelöscht. Das geht so weiter, bis die vom Speicherort eingenommene Größe unterhalb von `storage-limit` liegt.

  Jedes Kriterium wird in Form einer Zeichenkette, dann einem Gleichheitszeichen, dann noch einer Zeichenkette festgelegt. Derzeit wird nur ein Kriterium unterstützt, nämlich ob ein Nar schon auf einer anderen Nar-Herder-Instanz vorliegt.

`ttl` (Vorgabe: `#f`)
: `Cache-Control`-HTTP-Kopfzeilen erzeugen, die eine Time-to-live (TTL) von *ttl* signalisieren. Für *ttl* muss eine Dauer (mit dem Anfangsbuchstaben der Maßeinheit der Dauer im Englischen) angegeben werden: `5d` bedeutet 5 Tage, `1m` bedeutet 1 Monat und so weiter.

  Das ermöglicht es Guix, Substitutinformationen *ttl* lang zwischenzuspeichern.

`negative-ttl` (Vorgabe: `#f`)
: Eben solche `Cache-Control`-HTTP-Kopfzeilen für erfolglose (negative) Suchen erzeugen, um eine Time-to-live (TTL) zu signalisieren, wenn Store-Objekte fehlen und mit dem HTTP-Status-Code 404 geantwortet wird. Nach Vorgabe wird für negative Antworten *keine* TTL signalisiert.

`log-level` (Vorgabe: `'DEBUG`)
: Die Protokollstufe, etwa `'INFO`, um *nicht* einzelne Anfragen zu protokollieren.

12.9.34 Linux-Dienste

Early-OOM-Dienst

Early OOM (`https://github.com/rfjakob/earlyoom`), auch bekannt als Earlyoom, ist ein minimalistischer Out-Of-Memory-Daemon (OOM), um auf Anwendungsebene („User Space“) Programme abzuwürgen, wenn einem der freie Arbeitsspeicher ausgeht (ein „OOM-Killer“). Er stellt eine Alternative zum im Kernel eingebauten OOM-Killer dar, mit der das System in einem solchen Fall besser weiterhin auf Benutzereingaben reagieren kann und die konfigurierbarer ist.

`earlyoom-service-type` [Scheme-Variable]
: Der Diensttyp, um `earlyoom`, den Early-OOM-Daemon, auszuführen. Als Wert muss ein `earlyoom-configuration`-Objekt angegeben werden, wie unten beschrieben. So kann der Dienst mit seiner Vorgabekonfiguration instanziieret werden:

```
(service earlyoom-service-type)
```

`earlyoom-configuration` [Datentyp]
: Dies ist das Verbundsobjekt mit der Konfiguration des `earlyoom-service-type`.

`earlyoom` (Vorgabe: *earlyoom*)
: Das Earlyoom-Paket, das benutzt werden soll.

`minimum-available-memory` (Vorgabe: `10`)
: Der Schwellwert, wie viel Arbeitsspeicher mindestens *verfügbar* bleiben muss, in Prozent.

`minimum-free-swap` (Vorgabe: `10`)
: Der Schwellwert, wie viel Swap-Speicher mindestens frei bleiben muss, in Prozent.

`prefer-regexp` (Vorgabe: `#f`)
: Ein regulärer Ausdruck (als eine Zeichenkette), der auf die Namen jener Prozesse passt, die als Erste abgewürgt werden sollen.

`avoid-regexp` (Vorgabe: `#f`)
: Ein regulärer Ausdruck (als eine Zeichenkette), der auf die Namen jener Prozesse passt, die *nicht* abgewürgt werden sollen.<

`memory-report-interval` (Vorgabe: `0`)
: Das Intervall in Sekunden, in dem ein Bericht über den Speicher ausgegeben werden soll. Nach Vorgabe ist es deaktiviert.

`ignore-positive-oom-score-adj?` (Vorgabe: `#f`)
: Ein boolescher Wert, der angibt, ob die in `/proc/*/oom_score_adj` festgelegten Anpassungen nach oben ignoriert werden sollen.

`show-debug-messages?` (Vorgabe: `#f`)
: Ein boolescher Ausdruck, der angibt, ob Nachrichten zur Fehlersuche ausgegeben werden sollen. Die Protokolle werden unter `/var/log/earlyoom.log` gespeichert.

`send-notification-command` (Vorgabe: `#f`)
: Hiermit kann ein eigener Befehl eingestellt werden, um Benachrichtigungen zu senden.

Kernelmodul-Ladedienst

Mit dem Kernelmodul-Ladedienst („Kernel Module Loader Service“) können Sie veranlassen, dass hinzuladbare Kernelmodule beim Systemstart geladen werden. Das bietet sich besonders für Module an, die nicht automatisch geladen werden („Autoload“), sondern manuell geladen werden müssen, wie es z.B. bei `ddcci` der Fall ist.

`kernel-module-loader-service-type` [Scheme-Variable]
: Der Diensttyp, um hinzuladbare Kernelmodule beim Systemstart über `modprobe` zu laden. Als Wert muss eine Liste von Zeichenketten angegeben werden, die den Modulnamen entsprechen. Um zum Beispiel die durch `ddcci-driver-linux` zur Verfügung gestellten Treiber zu laden und dabei durch Übergabe bestimmter Parameter den Modus zur Fehlersuche zu aktivieren, können Sie Folgendes benutzen:

```
(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)))
```

Rasdaemon-Dienst

Mit dem Rasdaemon-Dienst steht Ihnen ein Daemon zur Verfügung, mit dem Trace-Ereignisse des Linux-Kernels bezüglich der Zuverlässigkeit (Reliability), Verfügbarkeit (Availability) und Wartbarkeit (Serviceability) der Plattform beobachtet werden. Sie werden in syslogd protokolliert.

Reliability, Availability, Serviceability ist ein Konzept auf Servern, um deren Robustheit zu messen.

Relability (Zuverlässigkeit) ist die Wahrscheinlichkeit, mit der ein System korrekte Ausgaben liefert:

- Im Allgemeinen wird sie durch die Mittlere Betriebsdauer zwischen Ausfällen (Mean Time Between Failures, MTBF) beziffert.
- Sie wird besser, wenn Hardware-Fehler vermieden, erkannt oder repariert werden können.

Availability (Verfügbarkeit) ist die Wahrscheinlichkeit, dass ein System zu einem bestimmten Zeitpunkt betriebsbereit ist:

- Im Allgemeinen wird sie als Anteil der Ausfallzeit an der Gesamtzeit gemessen.
- Oft kommen hier Mechanismen zum Einsatz, um Hardware-Fehler im laufenden Betrieb zu erkennen und zu beheben.

Serviceability (Wartbarkeit) bezeichnet, wie einfach und schnell ein System repariert und gepflegt werden kann:

- Im Allgemeinen wird sie durch den Mittleren Abstand zwischen Reparaturen (Mean Time Between Repair, MTBR) gemessen.

Zu den beobachtbaren Messwerten gehören für gewöhnlich:

- Prozessor – Fehler erkennen bei der Befehlsausführung (Instruction Execution) und an den Prozessor-Caches (L1/L2/L3),
- Arbeitsspeicher – Fehlerbehebungslogik wie ECC einbauen, um Fehler zu erkennen und zu beheben,
- Ein-/Ausgabe – CRC-Prüfsummen für übertragene Daten einbauen,
- Massenspeicher – RAID, Journaling in Dateisystemen, Prüfsummen, Selbstüberwachung, Technologien zur Analyse und Berichterstattung (SMART).

Durch das Beobachten, wie oft Fehler erkannt werden, kann bestimmt werden, ob die Wahrscheinlichkeit von Hardware-Fehlern zunimmt, und in diesem Fall kann ein vorsorglicher Austausch der schwächelnden Komponente vorgenommen werden, solange die Fehler leicht zu beheben sind.

Genaue Informationen über die Arten der gesammelten Fehlerereignisse und was sie bedeuten finden Sie in der Anleitung für Kernel-Administratoren unter `https://www.kernel.org/doc/html/latest/admin-guide/ras.html`.

`rasdaemon-service-type` [Scheme-Variable]

Diensttyp für den `rasdaemon`-Dienst. Als Wert nimmt er ein `rasdaemon-configuration`-Objekt. Sie instanziieren ihn so:

```
(service rasdaemon-service-type)
```

Dadurch wird die Vorgabekonfiguration geladen, mit der alle Ereignisse beobachtet und mit syslogd protokolliert werden.

`rasdaemon-configuration` [Datentyp]

Repräsentiert die Konfiguration von `rasdaemon`.

`record?` (Vorgabe: `#f`)

Ein Boolescher Ausdruck, der anzeigt, ob die Ereignisse in eine SQLite-Datenbank geschrieben werden. Dadurch wird ein strukturierterer Zugang

zu den Informationen aus der Protokolldatei ermöglicht. Der Ort für die Datenbank ist fest einprogrammiert als `/var/lib/rasdaemon/ras-mc_event.db`.

Dienst für Zram-Geräte

Der Dienst für Zram-Geräte macht ein komprimiertes Swap-Gerät im Arbeitsspeicher verfügbar. Mehr Informationen finden Sie in der Linux-Kernel-Dokumentation über Zram-Geräte (`https://www.kernel.org/doc/html/latest/admin-guide/blockdev/zram.html`).

`zram-device-service-type` [Scheme-Variable]

Dieser Dienst erzeugt das Zram-Blockgerät, formatiert es als Swap-Speicher und aktiviert es als ein Swap-Gerät. Der Wert des Dienstes ist ein `zram-device-configuration`-Verbundsobjekt.

`zram-device-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des zram-device-Dienstes.

`size` (Vorgabe: `"1G"`)

Wie viel Speicher Sie für das Zram-Gerät verfügbar machen möchten. Angegeben werden kann eine Zeichenkette oder eine Zahl mit der Anzahl Bytes oder mit einem Suffix, z.B. `"512M"` oder `1024000`.

`compression-algorithm` (Vorgabe: `'lzo`)

Welchen Kompressionsalgorithmus Sie verwenden möchten. Alle Konfigurationsmöglichkeiten lassen sich hier nicht aufzählen, aber der Linux-Libre-Kernel von Guix unterstützt unter anderem die häufig benutzten `'lzo`, `'lz4` und `'zstd`.

`memory-limit` (Vorgabe: `0`)

Wie viel Arbeitsspeicher dem Zram-Gerät höchstens zur Verfügung steht. Wenn es auf 0 steht, entfällt die Beschränkung. Obwohl im Allgemeinen von einem Kompressionsverhältnis von 2:1 ausgegangen wird, kann es passieren, dass *nicht* komprimierbare Daten den Swap-Speicher füllen. Mit diesem Feld kann das begrenzt werden. Es nimmt eine Zeichenkette oder eine Anzahl Bytes; ein Suffix kann angegeben werden, z.B. `"2G"`.

`priority` (Vorgabe: `#f`)

Welche Priorität dem Swap-Gerät gegeben wird, das aus dem Zram-Gerät entsteht. Siehe Abschnitt 12.5 [Swap-Speicher], Seite 271, für eine Beschreibung der Swap-Prioritäten. Es kann wichtig sein, für das Zram-Gerät eine bestimmte Priorität zuzuweisen, sonst bleibt es am Ende aus den dort beschriebenen Gründen praktisch ungenutzt.

12.9.35 Hurd-Dienste

`hurd-console-service-type` [Scheme-Variable]

Diensttyp, mit dem der fantastische `VGA`-Konsolen-Client auf Hurd ausgeführt wird.

Der Wert des Dienstes ist ein `hurd-console-configuration`-Verbundsobjekt.

`hurd-console-configuration` [Datentyp]
Dieser Datentyp repräsentiert die Konfiguration des hurd-console-Dienstes.

`hurd` (Vorgabe: *hurd*)
Das zu verwendende Hurd-Paket.

`hurd-getty-service-type` [Scheme-Variable]
Der Diensttyp, um ein TTY über Hurds `getty`-Programm zu starten.

Der Wert des Dienstes ist ein `hurd-getty-configuration`-Verbundsobjekt.

`hurd-getty-configuration` [Datentyp]
Dieser Datentyp repräsentiert die Konfiguration des hurd-getty-Dienstes.

`hurd` (Vorgabe: *hurd*)
Das zu verwendende Hurd-Paket.

`tty` Der Name der Konsole, auf der dieses Getty läuft – z.B. `"tty1"`.

`baud-rate` (Vorgabe: `38400`)
Eine ganze Zahl, die die Baud-Rate des TTYs angibt.

12.9.36 Verschiedene Dienste

Fingerabdrucklese-Dienst

Das Modul `(gnu services authentication)` stellt einen DBus-Dienst zur Verfügung, mit dem Fingerabdrücke mit Hilfe eines Fingerabdrucksensors gelesen und identifiziert werden können.

`fprintd-service-type` [Scheme-Variable]
Der Diensttyp für `fprintd`, mit dem Fingerabdrücke gelesen werden können.

```
(service fprintd-service-type)
```

Systemsteuerungsdienst

Das Modul `(gnu services sysctl)` stellt einen Dienst zur Verfügung, um Kernelparameter zur Boot-Zeit einzustellen.

`sysctl-service-type` [Scheme-Variable]
Der Diensttyp für `sysctl`, das Kernel-Parameter unter `/proc/sys/` anpasst. Um IPv4-Weiterleitung („Forwarding“) zu aktivieren, kann er auf diese Weise instanziiert werden:

```
(service sysctl-service-type
         (sysctl-configuration
           (settings '(("net.ipv4.ip_forward" . "1")))))
```

Weil `sysctl-service-type` in den Listen vorgegebener Dienste vorkommt, sowohl in `%base-services` als auch in `%desktop-services`, können Sie `modify-services` benutzen, um seine Konfiguration zu ändern und die Kernel-Parameter einzutragen, die Sie möchten (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639).

```
(modify-services %base-services
```

```
(sysctl-service-type config =>
                     (sysctl-configuration
                       (settings (append '(("net.ipv4.ip_forward" . "1"))
                                         %default-sysctl-settings)))))
```

`sysctl-configuration` [Datentyp]

Der Datentyp, der die Konfiguration von `sysctl` repräsentiert.

`sysctl` (Vorgabe: `(file-append procps "/sbin/sysctl")`)
: Die ausführbare Datei für `sysctl`, die benutzt werden soll.

`settings` (Vorgabe: `%default-sysctl-settings`)
: Eine assoziative Liste, die Kernel-Parameter und ihre Werte festlegt.

`%default-sysctl-settings` [Scheme-Variable]

Eine assoziative Liste, die die vorgegebenen `sysctl`-Parameter auf Guix System enthält.

PC/SC-Smart-Card-Daemon-Dienst

Das Modul `(gnu services security-token)` stellt den folgenden Dienst zur Verfügung, um `pcscd` auszuführen, den PC/SC-Smart-Card-Daemon. `pcscd` ist das Daemonprogramm für die Rahmensysteme pcsc-lite und MuscleCard. Es handelt sich um einen Ressourcenverwaltungsdienst, der die Kommunikation mit Smart-Card-Lesegeräten, Smart Cards und kryptographischen Tokens steuert, die mit dem System verbunden sind.

`pcscd-service-type` [Scheme-Variable]

Diensttyp für den `pcscd`-Dienst. Als Wert muss ein `pcscd-configuration`-Objekt angegeben werden. Um pcscd mit seiner Vorgabekonfiguration auszuführen, instanzieren Sie ihn als:

```
(service pcscd-service-type)
```

`pcscd-configuration` [Datentyp]

Repräsentiert die Konfiguration von `pcscd`.

`pcsc-lite` (Vorgabe: `pcsc-lite`)
: Das „pcsc-lite“-Paket, das pcscd zur Verfügung stellt.

`usb-drivers` (Vorgabe: `(list ccid)`)
: Die Liste der Pakete, die USB-Treiber für pcscd zur Verfügung stellen. Es wird erwartet, dass sich Treiber unter `pcsc/drivers` innerhalb des Store-Verzeichnisses des Pakets befinden.

Lirc-Dienst

Das Modul `(gnu services lirc)` stellt den folgenden Dienst zur Verfügung.

`lirc-service` [*#:lirc lirc*] [*#:device #f*] [*#:driver #f*] [*#:config-file #f*] [*#:extra-options '()*] [Scheme-Prozedur]

Liefert einen Dienst, der LIRC (`http://www.lirc.org`) ausführt, einen Dienst zum Dekodieren von Infrarot-Signalen aus Fernbedienungen.

Optional können *device* (Gerät), *driver* (Treiber) und *config-file* (Name der Konfigurationsdatei) festgelegt werden. Siehe das Handbuch von `lircd` für Details.

Schließlich enthält *extra-options* eine Liste zusätzlicher Befehlszeilenoptionen, die an `lircd` übergeben werden.

Spice-Dienst

Das Modul `(gnu services spice)` stellt den folgenden Dienst bereit.

`spice-vdagent-service` [*#:spice-vdagent*] [Scheme-Prozedur]
Liefert einen Dienst, der VDAGENT (`https://www.spice-space.org`) ausführt, einen Daemon, um die Zwischenablage mit einer virtuellen Maschine zu teilen und die Auflösung des Anzeigegeräts des Gastsystems umzustellen, wenn sich die Größe des grafischen Konsolenfensters ändert.

inputattach-Dienst

Der inputattach-Dienst (`https://linuxwacom.github.io/`) macht es Ihnen möglich, Eingabegeräte wie Wacom-Tabletts, Tastbildschirme („Touchscreens“) oder Joysticks mit dem Xorg-Anzeigeserver zu benutzen.

`inputattach-service-type` [Scheme-Variable]
Der Diensttyp für den Dienst, der `inputattach` auf einem Gerät ausführt und Ereignisse davon weiterleitet.

`inputattach-configuration` [Datentyp]

`device-type` (Vorgabe: `"wacom"`)
Der Typ des Geräts, mit dem eine Verbindung hergestellt werden soll. Führen Sie `inputattach --help` aus dem `inputattach`-Paket aus, um eine Liste unterstützter Gerätetypen zu sehen.

`device` (Vorgabe: `"/dev/ttyS0"`)
Die Gerätedatei, um sich mit dem Gerät zu verbinden.

`baud-rate` (Vorgabe: `#f`)
Welche Baudrate für die serielle Verbindung benutzt werden soll. Es sollte eine Zahl oder `#f` angegeben werden.

`log-file` (Vorgabe: `#f`)
Wenn es wahr ist, muss es der Name einer Datei sein, in die Protokollnachrichten geschrieben werden sollen.

Wörterbuchdienst

Das Modul `(gnu services dict)` stellt den folgenden Dienst zur Verfügung:

`dicod-service-type` [Scheme-Variable]
Dies ist der Diensttyp für einen Dienst, der den `dicod`-Daemon ausführt. Dabei handelt es sich um eine Implementierung eines DICT-Servers (siehe das Abschnitt “Dicod” in *Handbuch von GNU Dico*).

`dicod-service` [*#:config* (*dicod-configuration*)] [Scheme-Prozedur]
Liefert einen Dienst, der den `dicod`-Daemon ausführt. Dabei handelt es sich um eine Implementierung eines DICT-Servers (siehe das Abschnitt “Dicod” in *Handbuch von GNU Dico*).

Das optionale Argument *config* gibt die Konfiguration für `dicod` an, welche ein `<dicod-configuration>`-Objekt sein sollte. Nach Vorgabe wird als Wörterbuch das „GNU Collaborative International Dictionary of English“ angeboten.

Sie können in Ihre `~/.dico`-Datei `open localhost` eintragen, damit `localhost` zum voreingestellten Server des `dico`-Clients wird (siehe das Abschnitt “Initialization File” in *Handbuch von GNU Dico*).

`dicod-configuration` [Datentyp]

Der Datentyp, der die Konfiguration von dicod repräsentiert.

`dico` (Vorgabe: *dico*)
: Paketobjekt des GNU-Dico-Wörterbuchservers.

`interfaces` (Vorgabe: *'("localhost")*)
: Hierfür muss die Liste der IP-Adressen, Ports und möglicherweise auch Socket-Dateinamen angegeben werden, auf die gelauscht werden soll (siehe Abschnitt “Server Settings” in *Handbuch von GNU Dico*).

`handlers` (Vorgabe: *'()*)
: Liste der `<dicod-handler>`-Objekte, die Handler (Modulinstanzen) bezeichnen.

`databases` (Vorgabe: *(list %dicod-database:gcide)*)
: Liste der `<dicod-database>`-Objekte, die anzubietende Wörterbücher bezeichnen.

`dicod-handler` [Datentyp]

Der Datentyp, der einen Wörterbuch-Handler (eine Modulinstanz) repräsentiert.

`name` Der Name des Handlers (der Modulinstanz).

`module` (Vorgabe: *#f*)
: Der Name des dicod-Moduls (der Instanz) des Handlers. Wenn er `#f` ist, heißt das, das Modul hat denselben Namen wie der Handler (siehe Abschnitt “Modules” in *Handbuch von GNU Dico*).

`options` Liste der Zeichenketten oder G-Ausdrücke, die die Argumente für den Modul-Handler repräsentieren.

`dicod-database` [Datentyp]

Datentyp, der eine Wörterbuchdatenbank repräsentiert.

`name` Der Name der Datenbank, der in DICT-Befehlen benutzt wird.

`handler` Der Name des dicod-Handlers (der Modulinstanz), die von dieser Datenbank benutzt wird (siehe Abschnitt “Handlers” in *Handbuch von GNU Dico*).

`complex?` (Vorgabe: *#f*)
: Ob die Datenbankkonfiguration komplex ist. In diesem Fall muss für die komplexe Konfiguration auch ein entsprechendes `<dicod-handler>`-Objekt existieren, ansonsten nicht.

`options` Liste der Zeichenketten oder G-Ausdrücke, die die Argumente für die Datenbank repräsentiert (siehe Abschnitt "Databases" in *Handbuch von GNU Dico*).

`%dicod-database:gcide` [Scheme-Variable]
Ein `<dicod-database>`-Objekt, um das „GNU Collaborative International Dictionary of English" anzubieten. Dazu wird das `gcide`-Paket benutzt.

Im Folgenden sehen Sie eine Beispielkonfiguration für einen `dicod-service`.

```
(dicod-service #:config
  (dicod-configuration
   (handlers (list (dicod-handler
                    (name "wordnet")
                    (module "dictorg")
                    (options
                     (list #~(string-append "dbdir=" #$wordnet))))))
   (databases (list (dicod-database
                     (name "wordnet")
                     (complex? #t)
                     (handler "wordnet")
                     (options '("database=wn")))
                    %dicod-database:gcide))))
```

Docker-Dienst

Das Modul `(gnu services docker)` stellt die folgenden Dienste zur Verfügung.

`docker-service-type` [Scheme-Variable]
Dies ist der Diensttyp des Dienstes, um Docker (`https://www.docker.com`) auszuführen, einen Daemon, der Anwendungsbündel in „Containern", d.h. isolierten Umgebungen, ausführen kann.

`docker-configuration` [Datentyp]
Dies ist der Datentyp, der die Konfiguration von Docker und Containerd repräsentiert.

`docker` (Vorgabe: `docker`)
Das Docker-Daemon-Paket, was benutzt werden soll.

`docker-cli` (Vorgabe: `docker-cli`)
Das Docker-Client-Paket, was benutzt werden soll.

`containerd` (Vorgabe: *containerd*)
Das Containerd-Paket, was benutzt werden soll.

`proxy` (Vorgabe: *docker-libnetwork-cmd-proxy*)
Das Paket des mit Benutzerrechten (im „User-Land") ausgeführten Docker-Netzwerkproxys, das verwendet werden soll.

`enable-proxy?` (Vorgabe: `#t`)
Den mit Benutzerrechten (im „User-Land") ausgeführten Docker-Netzwerkproxy an- oder abschalten.

`debug?` (Vorgabe: `#f`)
: Ausgaben zur Fehlersuche an- oder abschalten.

`enable-iptables?` (Vorgabe: `#t`)
: Das Hinzufügen von iptables-Regeln durch Docker an- oder abschalten.

`environment-variables` (Vorgabe: `()`)
: Liste Umgebungsvariabler, mit denen `dockerd` gestartet wird.

  Hier muss eine Liste von Zeichenketten angegeben werden, die jeweils der Form '*`Schlüssel=Wert`*' genügen wie in diesem Beispiel:

```
(list "LANGUAGE=eo:ca:eu"
      "TMPDIR=/tmp/dockerd")
```

`singularity-service-type` [Scheme-Variable]
: Dies ist der Diensttyp für den Dienst, mit dem Sie Singularity (`https://www.sylabs.io/singularity/`) ausführen können, ein Docker-ähnliches Werkzeug, um Anwendungsbündel (auch bekannt als „Container“) auszuführen. Der Wert für diesen Dienst ist das Singularity-Paket, das benutzt werden soll.

  Der Dienst installiert keinen Daemon, sondern er installiert Hilfsprogramme als setuid-root (siehe Abschnitt 12.10 [Setuid-Programme], Seite 604), damit auch „unprivilegierte“ Nutzer ohne besondere Berechtigungen `singularity run` und ähnliche Befehle ausführen können.

Auditd-Dienst

Das Modul `(gnu services auditd)` stellt den folgenden Dienst zur Verfügung.

`auditd-service-type` [Scheme-Variable]
: Dies ist der Diensttyp des Dienstes, mit dem auditd (`https://people.redhat.com/sgrubb/audit/`) ausgeführt wird, ein Daemon, der sicherheitsrelevante Informationen auf Ihrem System sammelt.

  Beispiele für Dinge, über die Informationen gesammelt werden sollen:

  1. Dateizugriffe
  2. Betriebssystemaufrufe („System Calls“)
  3. Aufgerufene Befehle
  4. Fehlgeschlagene Anmeldeversuche
  5. Filterung durch die Firewall
  6. Netzwerkzugriff

  `auditctl` aus dem `audit`-Paket kann benutzt werden, um zu überwachende Ereignisse (bis zum nächsten Neustart) hinzuzufügen oder zu entfernen. Um über Ereignisse dauerhaft Informationen sammeln zu lassen, schreiben Sie die Befehlszeilenargumente für auditctl in eine Datei namens `audit.rules` im Verzeichnis für Konfigurationen (siehe unten). `aureport` aus dem `audit`-Paket kann benutzt werden, um einen Bericht über alle aufgezeichneten Ereignisse anzuzeigen. Nach Vorgabe speichert der Audit-Daemon Protokolle in die Datei `/var/log/audit.log`.

`auditd-configuration` [Datentyp]
Dies ist der Datentyp, der die Konfiguration von auditd repräsentiert.

`audit` (Vorgabe: `audit`)
Das zu verwendende audit-Paket.

`configuration-directory` (Vorgabe: `%default-auditd-configuration-directory`)
Das Verzeichnis mit der Konfigurationsdatei für das audit-Paket. Sie muss den Namen `auditd.conf` tragen und optional kann sie ein paar Audit-Regeln enthalten, die beim Start instanziiert werden sollen.

R-Shiny-Dienst

Das Modul `(gnu services science)` stellt den folgenden Dienst bereit.

`rshiny-service-type` [Scheme-Variable]
Dies ist der Diensttyp eines Dienstes, um eine mit `r-shiny` erzeugte Web-Anwendung („Webapp“) auszuführen. Dieser Dienst legt die Umgebungsvariable `R_LIBS_USER` fest und führt das eingestellte Skript aus, um `runApp` aufzurufen.

`rshiny-configuration` [Datentyp]
Dies ist der Datentyp, der die Konfiguration von rshiny repräsentiert.

`package` (Vorgabe: `r-shiny`)
Das zu benutzende Paket.

`binary` (Vorgabe: `"rshiny"`)
Der Name der Binärdatei oder des Shell-Skripts, das sich in `Paket/bin/` befindet und beim Starten des Dienstes ausgeführt werden soll.

Die übliche Art, diese Datei erzeugen zu lassen, ist folgende:

```
...
(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))))
```

Nix-Dienst

Das Modul `(gnu services nix)` stellt den folgenden Dienst zur Verfügung:

`nix-service-type` [Scheme-Variable]

Dies ist der Diensttyp für den Dienst, der den Erstellungs-Daemon der Nix-Paketverwaltung (`https://nixos.org/nix/`) ausführt. Hier ist ein Beispiel, wie man ihn benutzt:

```
(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)))
```

Nach `guix system reconfigure` können Sie Nix für Ihr Benutzerkonto konfigurieren:

- Fügen Sie einen Nix-Kanal ein und aktualisieren Sie ihn. Siehe Nix Package Manager Guide (`https://nixos.org/nix/manual/`).
- Erzeugen Sie eine symbolische Verknüpfung zu Ihrem Profil und aktivieren Sie das Nix-Profil:

```
$ ln -s "/nix/var/nix/profiles/per-user/$USER/profile" ~/.nix-profile
$ source /run/current-system/profile/etc/profile.d/nix.sh
```

`nix-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration des Nix-Daemons.

`nix` (Vorgabe: `nix`)
: Das zu verwendende Nix-Paket.

`sandbox` (Vorgabe: `#t`)
: Gibt an, ob Erstellungen nach Voreinstellung in einer isolierten Umgebung („Sandbox“) durchgeführt werden sollen.

`build-sandbox-items` (Vorgabe: `'()`)
: Dies ist eine Liste von Zeichenketten oder Objekten, die an das `build-sandbox-items`-Feld der Konfigurationsdatei angehängt werden.

`extra-config` (Vorgabe: `'()`)
: Dies ist eine Liste von Zeichenketten oder Objekten, die an die Konfigurationsdatei angehängt werden. Mit ihnen wird zusätzlicher Text wortwörtlich zur Konfigurationsdatei hinzugefügt.

`extra-options` (Vorgabe: `'()`)
: Zusätzliche Befehlszeilenoptionen für `nix-service-type`.

Fail2Ban-Dienst

Durch `fail2ban` (`http://www.fail2ban.org/`) werden Protokolldateien durchgesehen (wie `/var/log/apache/error_log`) und diejenigen IP-Adressen ausgesperrt, die bösartig erscheinen – also mehrfach ein falsches Passwort probieren, nach Sicherheitslücken suchen und Ähnliches.

Der Diensttyp `fail2ban-service-type` wird durch das Modul `(gnu services security)` verfügbar gemacht.

Mit dem Diensttyp wird der `fail2ban`-Daemon ausgeführt. Sie können ihn auf unterschiedliche Weise konfigurieren, nämlich:

Grundlegende Konfiguration
: Die Grundeinstellungen für den Fail2Ban-Dienst können Sie über seine `fail2ban`-Konfiguration bestimmen. Ihre Dokumentation finden Sie unten.

Vom Nutzer festgelegte Jail-Erweiterungen
: Mit der Funktion `fail2ban-jail-service` können neue Fail2Ban-Jails hinzugefügt werden.

Erweiterungen für Shepherd-Dienste
: Entwickler von Diensten können den Dienst mit dem Typ `fail2ban-service-type` über den üblichen Mechanismus zur Diensterweiterung erweitern.

`fail2ban-service-type` [Scheme-Variable]

Dies ist der Diensttyp für einen Dienst, der den `fail2ban`-Daemon ausführt. Hier folgt ein Beispiel für eine grundlegende, explizite Konfiguration.

```
(append
 (list
  (service fail2ban-service-type
           (fail2ban-configuration
            (extra-jails
             (list
              (fail2ban-jail-configuration
               (name "sshd")
               (enabled? #t))))))
  ;; Es besteht keine implizite Abhängigkeit von einem wirklichen
  ;; SSH-Dienst, also müssen Sie ihn zusätzlich hinschreiben.
  (service openssh-service-type))
 %base-services)
```

`fail2ban-jail-service` *Diensttyp Jail* [Scheme-Prozedur]

Den *Diensttyp*, ein `<service-type>`-Objekt, mit *Jail* ausstatten, einem Objekt vom Typ `fail2ban-jail-configuration`.

Zum Beispiel:

```
(append
 (list
  (service
   ;; Durch die Prozedur 'fail2ban-jail-service' kann jeglicher Diensttyp
   ;; mit einem fail2ban-Jail versehen werden. So müssen die Dienste
   ;; nicht explizit im fail2ban-service-type aufgeführt werden.
   (fail2ban-jail-service
    openssh-service-type
    (fail2ban-jail-configuration
     (name "sshd")
```

```
          (enabled? #t)))
        (openssh-configuration ...))))
```

Nun folgt die Referenz der Verbundstypen zur Konfiguration des `jail-service-type`.

`fail2ban-configuration` [Datentyp]

Verfügbare `fail2ban-configuration`-Felder sind:

`fail2ban` (Vorgabe: `fail2ban`) (Typ: „package“)
: Welches `fail2ban`-Paket benutzt werden soll. Es stellt sowohl die Binärdateien als auch die Voreinstellungen für die Konfiguration bereit, die mit `<fail2ban-jail-configuration>`-Objekten erweitert werden soll.

`run-directory` (Vorgabe: `"/var/run/fail2ban"`) (Typ: Zeichenkette)
: Das Zustandsverzeichnis für den `fail2ban`-Daemon.

`jails` (Vorgabe: `()`) (Typ: Liste-von-„fail2ban-jail-configuration“)
: `<fail2ban-jail-configuration>`-Instanzen, die sich aus Diensterweiterungen ergeben.

`extra-jails` (Vorgabe: `()`) (Typ: Liste-von-„fail2ban-jail-configuration“)
: `<fail2ban-jail-configuration>`-Instanzen, die explizit angegeben werden.

`extra-content` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Zusätzlicher „roher Inhalt“, der ans Ende der Datei `jail.local` angefügt wird. Geben Sie ihn als Liste von dateiartigen Objekten an.

`fail2ban-ignore-cache-configuration` [Datentyp]

Verfügbare `fail2ban-ignore-cache-configuration`-Felder sind:

`key` (Typ: Zeichenkette)
: Schlüssel, für den zwischengespeichert wird.

`max-count` (Typ: Ganze-Zahl)
: Für welche Größe zwischengespeichert wird.

`max-time` (Typ: Zeichenkette)
: Wie lange die Zwischenspeicherung anhält.

`fail2ban-jail-action-configuration` [Datentyp]

Verfügbare `fail2ban-jail-action-configuration`-Felder sind:

`name` (Typ: Zeichenkette)
: Name der Aktion.

`arguments` (Vorgabe: `()`) (Typ: Liste-von-Argumenten)
: Argumente der Aktion.

`fail2ban-jail-configuration` [Datentyp]

Verfügbare `fail2ban-jail-configuration`-Felder sind:

`name` (Typ: Zeichenkette)
: Verpflichtend der Name dieser Jail-Konfiguration.

`enabled?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Gibt an, ob dieses Jail aktiviert ist.

`backend` (Typ: Vielleicht-Symbol)
: Mit welchem Hintergrundprogramm Änderungen am `log-path` erkannt werden sollen. Voreingestellt ist 'auto. Um die Voreinstellungen zu der Jail-Konfiguration einzusehen, siehe die Datei `/etc/fail2ban/jail.conf` des `fail2ban`-Pakets.

`max-retry` (Typ: Vielleicht-Ganze-Zahl)
: Wie oft Fehler erkannt werden müssen, bevor ein Rechner gesperrt wird (etwa `(max-retry 5)`).

`max-matches` (Typ: Vielleicht-Ganze-Zahl)
: Wie viele Treffer je Ticket höchstens gespeichert werden sollen (in Aktionen kann man dies mit `<matches>` verwenden).

`find-time` (Typ: Vielleicht-Zeichenkette)
: In welchem Zeitfenster die Höchstzahl an Neuversuchen festgestellt werden muss, damit eine IP-Adresse gesperrt wird. Ein Rechner wird gesperrt, wenn er `max-retry` während der letzten `find-time` Sekunden erreicht hat (z.B. `(find-time "10m")`). Es kann in Sekunden angegeben werden oder im Zeitkurzformat von Fail2Ban (das „time abbreviation format“, das in `man 5 jail.conf` beschrieben ist).

`ban-time` (Typ: Vielleicht-Zeichenkette)
: Die Dauer einer Sperre, in Sekunden oder besagtem Zeitkurzformat. (Etwa `(ban-time "10m")`.)

`ban-time-increment?` (Typ: Vielleicht-Boolescher-Ausdruck)
: Ob vorherige Sperren einen Einfluss auf berechnete Steigerungen der Sperrzeit einer bestimmten IP-Adresse haben sollen.

`ban-time-factor` (Typ: Vielleicht-Zeichenkette)
: Der auf die Sperrzeit angerechnete Koeffizient für eine exponentiell zunehmende Sperrzeit.

`ban-time-formula` (Typ: Vielleicht-Zeichenkette)
: Mit dieser Formel wird der nächste Wert einer Sperrzeit berechnet.

`ban-time-multipliers` (Typ: Vielleicht-Zeichenkette)
: Hiermit wird der nächste Wert einer Sperrzeit berechnet und die Formel ignoriert.

`ban-time-max-time` (Typ: Vielleicht-Zeichenkette)
: Wie viele Sekunden jemand längstens gesperrt wird.

`ban-time-rnd-time` (Typ: Vielleicht-Zeichenkette)
: Wie viele Sekunden höchstens zufällig auf die Sperrzeit angerechnet werden. So können ausgefuchste Botnetze *nicht* genau ausrechnen, wann eine IP-Adresse wieder entsperrt sein wird.

`ban-time-overall-jails?` (Typ: Vielleicht-Boolescher-Ausdruck)
: Wenn dies wahr ist, wird festgelegt, dass nach einer IP-Adresse in der Datenbank aller Jails gesucht wird. Andernfalls wird nur das aktuelle Jail beachtet.

`ignore-self?` (Typ: Vielleicht-Boolescher-Ausdruck)
: Niemals die eigene IP-Adresse der lokalen Maschine bannen.

`ignore-ip` (Vorgabe: `()`) (Typ: Liste-von-Zeichenketten)
: Eine Liste von IP-Adressen, CIDR-Masken oder DNS-Rechnernamen, die ignoriert werden. `fail2ban` wird keinen Rechner bannen, der zu einer Adresse auf dieser Liste gehört.

`ignore-cache` (Typ: Vielleicht-„fail2ban-ignore-cache-configuration“)
: Machen Sie Angaben zum Zwischenspeicher, mit dem mehrfache Auffälligkeiten ignoriert werden können.

`filter` (Typ: Vielleicht-„fail2ban-jail-filter-configuration“)
: Der Filter, der für das Jail gilt, als `<fail2ban-jail-filter-configuration>`-Objekt. Nach Voreinstellung entsprechen die Namen der Jails ihren Filternamen.

`log-time-zone` (Typ: Vielleicht-Zeichenkette)
: Die Voreinstellung für die Zeitzone, wenn eine Zeile im Protokoll keine nennt.

`log-encoding` (Typ: Vielleicht-Symbol)
: In welcher Kodierung die Protokolldateien abgelegt sind, um die sich das Jail kümmert. Mögliche Werte sind `'ascii`, `'utf-8` und `'auto`.

`log-path` (Vorgabe: `()`) (Typ: Liste-von-Zeichenketten)
: Die Dateinamen der Protokolldateien, die beobachtet werden.

`action` (Vorgabe: `()`) (Typ: Liste-von-„fail2ban-jail-action“)
: Eine Liste von `<fail2ban-jail-action-configuration>`.

`extra-content` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Zusätzlicher Inhalt für die Jail-Konfiguration. Geben Sie ihn als Liste von dateiartigen Objekten an.

`fail2ban-jail-filter-configuration` [Datentyp]

Verfügbare `fail2ban-jail-filter-configuration`-Felder sind:

`name` (Typ: Zeichenkette)
: Filter, der benutzt werden soll.

`mode` (Typ: Vielleicht-Zeichenkette)
: In welchem Modus der Filter stehen soll.

12.10 Setuid-Programme

Manche Programme müssen mit erhöhten Berechtigungen ausgeführt werden, selbst wenn Nutzer ohne besondere Berechtigungen sie starten. Ein bekanntes Beispiel ist das Programm `passwd`, womit Nutzer ihr Passwort ändern können, wozu das Programm auf

die Dateien `/etc/passwd` und `/etc/shadow` zugreifen muss – was normalerweise nur der „root“-Nutzer darf, aus offensichtlichen Gründen der Informationssicherheit. Deswegen sollte `passwd` *setuid-root* sein, d.h. es läuft immer mit den Administratorrechten des root-Nutzers, egal wer sie startet (siehe Abschnitt “How Change Persona” in *Referenzhandbuch der GNU-C-Bibliothek* für mehr Informationen über den setuid-Mechanismus).

Der Store selbst kann *keine* setuid-Programme enthalten: Das wäre eine Sicherheitslücke, weil dann jeder Nutzer auf dem System Ableitungen schreiben könnte, die in den Store solche Dateien einfügen würden (siehe Abschnitt 9.9 [Der Store], Seite 165). Wir benutzen also einen anderen Mechanismus: Statt auf den ausführbaren Dateien im Store selbst deren setuid-Bit oder setgid-Bit zu setzen, lassen wir den Systemadministrator *deklarieren*, welchen Programmen diese zusätzlichen Berechtigungen gewährt werden.

Das Feld `setuid-programs` einer `operating-system`-Deklaration enthält eine Liste von `<setuid-program>`-Objekten, die die Namen der Programme angeben, deren setuid- oder setgid-Bits gesetzt werden sollen (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250). Zum Beispiel kann das Programm `mount.nfs`, was Teil des Pakets nfs-utils ist, so setuid-root werden:

```
(setuid-program
  (program (file-append nfs-utils "/sbin/mount.nfs")))
```

Um `mount.nfs` also mit setuid auszuführen, tragen Sie das vorige Beispiel in Ihre Betriebssystemdeklaration ein, indem Sie es an `%setuid-programs` anhängen wie hier:

```
(operating-system
  ;; Davor stehen andere Felder ...
  (setuid-programs
    (append (list (setuid-program
                    (program (file-append nfs-utils "/sbin/mount.nfs"))))
            %setuid-programs)))
```

`setuid-program` [Datentyp]

Dieser Datentyp steht für ein Programm, bei dem das setuid- oder setgid-Bit gesetzt werden soll.

`program` Ein dateiartiges Objekt, dessen setuid- und/oder setgid-Bit gesetzt werden soll.

`setuid?` (Vorgabe: `#t`)
Ob das setuid-Bit für den Benutzer gesetzt werden soll.

`setgid?` (Vorgabe: `#f`)
Ob das setgid-Bit für die Benutzergruppe gesetzt werden soll.

`user` (Vorgabe: `0`)
Benutzeridentifikator (UID, als ganze Zahl) oder Benutzername (als Zeichenkette) des Benutzers, dem das Programm gehören soll, nach Vorgabe der Administratornutzer root.

`group` (Vorgabe: `0`)
GID (als ganze Zahl) oder Gruppenname (als Zeichenkette) der Benutzergruppe, der das Programm gehört, nach Vorgabe die Benutzergruppe root.

Eine vorgegebene Menge von setuid-Programmen wird durch die Variable `%setuid-programs` aus dem Modul `(gnu system)` definiert.

`%setuid-programs` [Scheme-Variable]

Eine Liste von `<setuid-program>`-Objekten, die übliche Programme angeben, die setuid-root sein müssen.

Die Liste enthält Befehle wie `passwd`, `ping`, `su` und `sudo`.

Intern erzeugt Guix die eigentlichen setuid-Programme im Verzeichnis `/run/setuid-programs`, wenn das System aktiviert wird. Die Dateien in diesem Verzeichnis verweisen auf die „echten“ Binärdateien im Store.

12.11 X.509-Zertifikate

Über HTTPS verfügbare Webserver (also HTTP mit gesicherter Transportschicht, englisch „Transport-Layer Security“, kurz TLS) senden Client-Programmen ein *X.509-Zertifikat*, mit dem der Client den Server dann *authentifizieren* kann. Dazu verifiziert der Client, dass das Zertifikat des Servers von einer sogenannten Zertifizierungsstelle signiert wurde (*Certificate Authority*, kurz CA). Damit er aber die Signatur der Zertifizierungsstelle verifizieren kann, muss jeder Client das Zertifikat der Zertifizierungsstelle besitzen.

Web-Browser wie GNU IceCat liefern ihre eigenen CA-Zertifikate mit, damit sie von Haus aus Zertifikate verifizieren können.

Den meisten anderen Programmen, die HTTPS sprechen können – `wget`, `git`, `w3m` etc. – muss allerdings erst mitgeteilt werden, wo die CA-Zertifikate installiert sind.

In Guix müssen Sie dazu ein Paket, das Zertifikate enthält, in das `packages`-Feld der `operating-system`-Deklaration des Betriebssystems hinzufügen (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258). Guix liefert ein solches Paket mit, `nss-certs`, was als Teil von Mozillas „Network Security Services“ angeboten wird.

Beachten Sie, dass es *nicht* zu den `%base-packages` gehört, Sie es also ausdrücklich hinzufügen müssen. Das Verzeichnis `/etc/ssl/certs`, wo die meisten Anwendungen und Bibliotheken ihren Voreinstellungen entsprechend nach Zertifikaten suchen, verweist auf die global installierten Zertifikate.

Unprivilegierte Benutzer, wie die, die Guix auf einer Fremddistribution benutzen, können sich auch lokal ihre eigenen Pakete mit Zertifikaten in ihr Profil installieren. Eine Reihe von Umgebungsvariablen muss dazu definiert werden, damit Anwendungen und Bibliotheken wissen, wo diese Zertifikate zu finden sind. Und zwar folgt die OpenSSL-Bibliothek den Umgebungsvariablen `SSL_CERT_DIR` und `SSL_CERT_FILE`, manche Anwendungen benutzen stattdessen aber ihre eigenen Umgebungsvariablen. Das Versionskontrollsystem Git liest den Ort zum Beispiel aus der Umgebungsvariablen `GIT_SSL_CAINFO` aus. Sie würden typischerweise also so etwas ausführen:

```
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"
```

Ein weiteres Beispiel ist R, was voraussetzt, dass die Umgebungsvariable `CURL_CA_BUNDLE` auf ein Zertifikatsbündel verweist, weshalb Sie etwas wie hier ausführen müssten:

```
guix install nss-certs
```

```
export CURL_CA_BUNDLE="$HOME/.guix-profile/etc/ssl/certs/ca-certificates.crt"
```

Für andere Anwendungen möchten Sie die Namen der benötigten Umgebungsvariablen vielleicht in deren Dokumentation nachschlagen.

12.12 Name Service Switch

Das Modul `(gnu system nss)` enthält Anbindungen für die Konfiguration des *Name Service Switch* (NSS) der libc (siehe Abschnitt "NSS Configuration File" in *Referenzhandbuch der GNU-C-Bibliothek*). Kurz gesagt ist der NSS ein Mechanismus, mit dem die libc um neue „Namens"-Auflösungsmethoden für Systemdatenbanken erweitert werden kann; dazu gehören Rechnernamen (auch bekannt als „Host"-Namen), Dienstnamen, Benutzerkonten und mehr (siehe Abschnitt "Name Service Switch" in *Referenzhandbuch der GNU-C-Bibliothek*).

Die NSS-Konfiguration legt für jede Systemdatenbank fest, mit welcher Methode der Name nachgeschlagen („aufgelöst") werden kann und welche Methoden zusammenhängen – z.B. unter welchen Umständen der NSS es mit der nächsten Methode auf seiner Liste versuchen sollte. Die NSS-Konfiguration wird im Feld `name-service-switch` von `operating-system`-Deklarationen angegeben (siehe Abschnitt 12.2 [„operating-system"-Referenz], Seite 258).

Zum Beispiel konfigurieren die folgenden Deklarationen den NSS so, dass er das `nss-mdns`-Backend (`https://0pointer.de/lennart/projects/nss-mdns/`) benutzt, wodurch er auf `.local` endende Rechnernamen über Multicast-DNS (mDNS) auflöst:

```
(name-service-switch
   (hosts (list %files    ;zuerst in /etc/hosts nachschlagen

                ;; Wenn das keinen Erfolg hatte, es
                ;; mit 'mdns_minimal' versuchen.
                (name-service
                  (name "mdns_minimal")

                  ;; 'mdns_minimal' ist die Autorität für
                  ;; '.local'. Gibt es not-found ("nicht
                  ;; gefunden") zurück, müssen wir die
                  ;; nächsten Methoden gar nicht erst
                  ;; versuchen.
                  (reaction (lookup-specification
                             (not-found => return))))

                ;; Ansonsten benutzen wir DNS.
                (name-service
                  (name "dns"))

                ;; Ein letzter Versuch mit dem
                ;; "vollständigen" 'mdns'.
                (name-service
                  (name "mdns")))))
```

Keine Sorge: Die Variable `%mdns-host-lookup-nss` (siehe unten) enthält diese Konfiguration bereits. Statt das alles selst einzutippen, können Sie sie benutzen, wenn alles, was Sie möchten, eine funktionierende Namensauflösung für `.local`-Rechner ist.

Beachten Sie dabei, dass es zusätzlich zum Festlegen des `name-service-switch` in der `operating-system`-Deklaration auch erforderlich ist, den `avahi-service-type` zu benutzen (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314). Es genügt auch, wenn Sie die `%desktop-services` benutzen, weil er darin enthalten ist (siehe Abschnitt 12.9.9 [Desktop-Dienste], Seite 366). Dadurch wird `nss-mdns` für den Name Service Cache Daemon nutzbar (siehe Abschnitt 12.9.1 [Basisdienste], Seite 281).

Um sich eine lange Konfiguration zu ersparen, können Sie auch einfach die folgenden Variablen für typische NSS-Konfigurationen benutzen.

`%default-nss` [Scheme-Variable]
: Die vorgegebene Konfiguration des Name Service Switch als ein `name-service-switch`-Objekt.

`%mdns-host-lookup-nss` [Scheme-Variable]
: Die Name-Service-Switch-Konfiguration mit Unterstützung für Rechnernamensauflösung über „Multicast DNS“ (mDNS) für auf `.local` endende Rechnernamen.

Im Folgenden finden Sie eine Referenz, wie eine Name-Service-Switch-Konfiguration aussehen muss. Sie hat eine direkte Entsprechung zum Konfigurationsdateiformat der C-Bibliothek, lesen Sie weitere Informationen also bitte im Handbuch der C-Bibliothek nach (siehe Abschnitt “NSS Configuration File” in *Referenzhandbuch der GNU-C-Bibliothek*). Gegenüber dem Konfigurationsdateiformat des libc-NSS bekommen Sie mit unserer Syntax nicht nur ein warm umklammerndes Gefühl, sondern auch eine statische Analyse: Wenn Sie Syntax- und Schreibfehler machen, werden Sie darüber benachrichtigt, sobald Sie `guix system` aufrufen.

`name-service-switch` [Datentyp]
: Der Datentyp, der die Konfiguration des Name Service Switch (NSS) der libc repräsentiert. Jedes im Folgenden aufgeführte Feld repräsentiert eine der unterstützten Systemdatenbanken.

`aliases`
`ethers`
`group`
`gshadow`
`hosts`
`initgroups`
`netgroup`
`networks`
`password`
`public-key`
`rpc`
`services`
`shadow` Das sind die Systemdatenbanken, um die sich NSS kümmern kann. Jedes dieser Felder muss eine Liste aus `<name-service>`-Objekten sein (siehe unten).

`name-service` [Datentyp]
Der einen eigentlichen Namensdienst repräsentierende Datentyp zusammen mit der zugehörigen Auflösungsaktion.

`name` Eine Zeichenkette, die den Namensdienst bezeichnet (siehe Abschnitt "Services in the NSS configuration" in *Referenzhandbuch der GNU-C-Bibliothek*).

Beachten Sie, dass hier aufgeführte Namensdienste für den nscd sichtbar sein müssen. Dazu übergeben Sie im Argument `#:name-services` des `nscd-service` die Liste der Pakete, die die entsprechenden Namensdienste anbieten (siehe Abschnitt 12.9.1 [Basisdienste], Seite 281).

`reaction` Eine mit Hilfe des Makros `lookup-specification` angegebene Aktion (siehe Abschnitt "Actions in the NSS configuration" in *Referenzhandbuch der GNU-C-Bibliothek*). Zum Beispiel:

```
(lookup-specification (unavailable => continue)
                      (success => return))
```

12.13 Initiale RAM-Disk

Um ihn zu initialisieren (zu „bootstrappen"), wird für den Kernel Linux-Libre eine *initiale RAM-Disk* angegeben (kurz *initrd*). Eine initrd enthält ein temporäres Wurzeldateisystem sowie ein Skript zur Initialisierung. Letzteres ist dafür zuständig, das echte Wurzeldateisystem einzubinden und alle Kernel-Module zu laden, die dafür nötig sein könnten.

Mit dem Feld `initrd-modules` einer `operating-system`-Deklaration können Sie angeben, welche Kernel-Module für Linux-libre in der initrd verfügbar sein müssen. Insbesondere müssen hier die Module aufgeführt werden, um die Festplatte zu betreiben, auf der sich Ihre Wurzelpartition befindet – allerdings sollte der vorgegebene Wert der `initrd-modules` in dem meisten Fällen genügen. Wenn Sie aber zum Beispiel das Kernel-Modul `megaraid_sas` zusätzlich zu den vorgegebenen Modulen brauchen, um auf Ihr Wurzeldateisystem zugreifen zu können, würden Sie das so schreiben:

```
(operating-system
```

```
;; ...
(initrd-modules (cons "megaraid_sas" %base-initrd-modules)))
```

`%base-initrd-modules` [Scheme-Variable]

Der Vorgabewert für die Liste der Kernel-Module, die in der initrd enthalten sein sollen.

Wenn Sie noch systemnähere Anpassungen durchführen wollen, können Sie im Feld `initrd` einer `operating-system`-Deklaration angeben, was für eine Art von initrd Sie benutzen möchten. Das Modul `(gnu system linux-initrd)` enthält drei Arten, eine initrd zu erstellen: die abstrakte Prozedur `base-initrd` und die systemnahen Prozeduren `raw-initrd` und `expression->initrd`.

Mit der Prozedur `base-initrd` sollten Sie die häufigsten Anwendungszwecke abdecken können. Wenn Sie zum Beispiel ein paar Kernel-Module zur Boot-Zeit laden lassen möchten, können Sie das `initrd`-Feld auf diese Art definieren:

```
(initrd (lambda (file-systems . rest)
          ;; Eine gewöhnliche initrd, aber das Netzwerk wird
          ;; mit den Parametern initialisiert, die QEMU
          ;; standardmäßig erwartet.
          (apply base-initrd file-systems
                 #:qemu-networking? #t
                 rest)))
```

Die Prozedur `base-initrd` kann auch mit üblichen Anwendungszwecken umgehen, um das System als QEMU-Gastsystem zu betreiben oder als ein „Live“-System ohne ein dauerhaft gespeichertes Wurzeldateisystem.

Die Prozedur `base-initrd` baut auf der Prozedur `raw-initrd` auf. Anders als `base-initrd` hat `raw-initrd` keinerlei Zusatzfunktionalitäten: Es wird kein Versuch unternommen, für die initrd notwendige Kernel-Module und Pakete automatisch hinzuzunehmen. `raw-initrd` kann zum Beispiel benutzt werden, wenn ein Nutzer eine eigene Konfiguration des Linux-Kernels verwendet und die Standard-Kernel-Module, die mit `base-initrd` hinzugenommen würden, nicht verfügbar sind.

Die initiale RAM-Disk, wie sie von `base-initrd` oder `raw-initrd` erzeugt wird, richtet sich nach verschiedenen Optionen, die auf der Kernel-Befehlszeile übergeben werden (also über GRUBs `linux`-Befehl oder die `-append`-Befehlszeilenoption von QEMU). Erwähnt werden sollten:

`gnu.load=`*boot*

Die initiale RAM-Disk eine Datei *boot*, in der ein Scheme-Programm steht, laden lassen, nachdem das Wurzeldateisystem eingebunden wurde.

Guix übergibt mit dieser Befehlszeilenoption die Kontrolle an ein Boot-Programm, das die Dienstaktivierungsprogramme ausführt und anschließend den GNU Shepherd startet, das Initialisierungssystem („init“-System) von Guix System.

`root=`*Wurzel*

Das mit *Wurzel* bezeichnete Dateisystem als Wurzeldateisystem einbinden. *Wurzel* kann ein Geratename wie `/dev/sda1`, eine Dateisystembezeichnung

(d.h. ein Dateisystem-„Label“) oder eine Dateisystem-UUID sein. Wird nichts angegeben, wird der Gerätename aus dem Wurzeldateisystem der Betriebssystemdeklaration benutzt.

`rootfstype=`*`Typ`*
: Den Dateisystemtyp für das Wurzeldateisystem festlegen. Der angegebene Typ hat Vorrang vor dem `type`-Feld, das für das Wurzeldateisystem in der `operating-system`-Deklaration angegeben wurde, falls vorhanden.

`rootflags=`*`Optionen`*
: Die Einbinde-Optionen („mount options“) für das Wurzeldateisystem festlegen. Diese haben Vorrang vor dem *options*-Feld, das für das Wurzeldateisystem in der `operating-system`-Deklaration angegeben wurde, falls vorhanden.

`fsck.mode=`*`Modus`*
: Ob das mit *Wurzel* bezeichnete Dateisystem vor dem Einbinden auf Fehler geprüft werden soll. Als *Modus* geben Sie entweder `skip` (nie prüfen), `force` (immer prüfen) oder `auto` an. Bei `auto` wird die `check?`-Einstellung des Dateisystemobjekts für *Wurzel* verwendet (siehe Abschnitt 12.3 [Dateisysteme], Seite 263) und eine Dateisystemüberprüfung nur durchgeführt, wenn das Dateisystem nicht ordnungsgemäß heruntergefahren wurde.

  Die Voreinstellung ist `auto`, wenn diese Option nicht angegeben wird oder *Modus* keinem der genannten Werte entspricht.

`fsck.repair=`*`Stufe`*
: Die Stufe gibt an, wie erkannte Fehler im Wurzeldateisystem *Wurzel* automatisch repariert werden sollen. *Stufe* darf `no` sein (nichts an *Wurzel* ändern, wenn möglich), `yes` (so viele Fehler wie möglich beheben) oder `preen`. Letzteres repariert solche Probleme, wo die automatische Reparatur als unbedenklich eingeschätzt wird.

  Wenn Sie diese Option weglassen oder als *Stufe* keine der genannten angeben, wird als Voreinstellung so verfahren, als hätten Sie `preen` angegeben.

`gnu.system=`*`System`*
: `/run/booted-system` und `/run/current-system` auf das *System* zeigen lassen.

`modprobe.blacklist=`*`Module...`*
: Die initiale RAM-Disk sowie den Befehl `modprobe` (aus dem kmod-Paket) anweisen, das Laden der angegebenen *Module* zu verweigern. Als *Module* muss eine kommagetrennte Liste von Kernel-Modul-Namen angegeben werden – z.B. `usbkbd,9pnet`.

`gnu.repl` Eine Lese-Auswerten-Schreiben-Schleife (englisch „Read-Eval-Print Loop“, kurz REPL) von der initialen RAM-Disk starten, bevor diese die Kernel-Module zu laden versucht und das Wurzeldateisystem einbindet. Unsere Marketingabteilung nennt das *boot-to-Guile*. Der Schemer in Ihnen wird das lieben. Siehe Abschnitt “Using Guile Interactively” in *Referenzhandbuch zu GNU Guile* für mehr Informationen über die REPL von Guile.

Jetzt wo Sie wissen, was für Funktionalitäten eine durch `base-initrd` und `raw-initrd` erzeugte initiale RAM-Disk so haben kann, möchten Sie vielleicht auch wissen, wie man sie benutzt und weiter anpasst:

`raw-initrd` *Dateisysteme* [*#:linux-modules '()*] [Scheme-Prozedur]
[*#:pre-mount #t*] [*#:mapped-devices '()*] [*#:keyboard-layout #f*]
[*#:helper-packages '()*] [*#:qemu-networking? #f*]
[#:volatile-root? #f] Liefert eine Ableitung, die eine rohe („raw“) initrd erstellt. *Dateisysteme* bezeichnet eine Liste von durch die initrd einzubindenden Dateisystemen, unter Umständen zusätzlich zum auf der Kernel-Befehlszeile mit `root` angegebenen Wurzeldateisystem. *linux-modules* ist eine Liste von Kernel-Modulen, die zur Boot-Zeit geladen werden sollen. *mapped-devices* ist eine Liste von Gerätezuordnungen, die hergestellt sein müssen, bevor die unter *file-systems* aufgeführten Dateisysteme eingebunden werden (siehe Abschnitt 12.4 [Zugeordnete Geräte], Seite 269). *pre-mount* ist ein G-Ausdruck, der vor Inkrafttreten der *mapped-devices* ausgewertet wird. *helper-packages* ist eine Liste von Paketen, die in die initrd kopiert werden. Darunter kann `e2fsck/static` oder andere Pakete aufgeführt werden, mit denen durch die initrd das Wurzeldateisystem auf Fehler hin geprüft werden kann.

Ist es auf einen wahren Wert gesetzt, dann muss *keyboard-layout* eine Tastaturbelegung als `<keyboard-layout>`-Verbundsobjekt angeben, die die gewünschte Tastaturbelegung für die Konsole bezeichnet. Sie wird verwendet, noch bevor die Gerätezuordnungen in *mapped-devices* hergestellt werden und bevor die Dateisysteme in *file-systems* eingebunden werden, damit der Anwender dabei die gewollte Tastaturbelegung beim Eingeben einer Passphrase und bei der Nutzung einer REPL verwenden kann.

Wenn *qemu-networking?* wahr ist, wird eine Netzwerkverbindung mit den Standard-QEMU-Parametern hergestellt. Wenn *virtio?* wahr ist, werden zusätzliche Kernel-Module geladen, damit die initrd als ein QEMU-Gast paravirtualisierte Ein-/Ausgabetreiber benutzen kann.

Wenn *volatile-root?* wahr ist, ist Schreiben auf das Wurzeldateisystem möglich, aber Änderungen daran bleiben nicht erhalten.

`base-initrd` *Dateisysteme* [*#:mapped-devices '()*] [Scheme-Prozedur]
[*#:keyboard-layout #f*] [*#:qemu-networking? #f*]
[#:volatile-root? #f] [#:linux-modules '()] Liefert eine allgemein anwendbare, generische initrd als dateiartiges Objekt mit den Kernel-Modulen aus *linux*. Die *file-systems* sind eine Liste von durch die initrd einzubindenden Dateisystemen, unter Umständen zusätzlich zum Wurzeldateisystem, das auf der Kernel-Befehlszeile mit `root` angegeben wurde. Die *mapped-devices* sind eine Liste von Gerätezuordnungen, die hergestellt sein müssen, bevor die *file-systems* eingebunden werden.

Ist es auf einen wahren Wert gesetzt, dann muss *keyboard-layout* eine Tastaturbelegung als `<keyboard-layout>`-Verbundsobjekt angeben, die die gewünschte Tastaturbelegung für die Konsole bezeichnet. Sie wird verwendet, noch bevor die Gerätezuordnungen in *mapped-devices* hergestellt werden und bevor die Dateisysteme in *file-systems* eingebunden werden, damit der Anwender dabei die gewollte Tastaturbelegung beim Eingeben einer Passphrase und bei der Nutzung einer REPL verwenden kann.

qemu-networking? und *volatile-root?* verhalten sich wie bei `raw-initrd`.

In die initrd werden automatisch alle Kernel-Module eingefügt, die für die unter *file-systems* angegebenen Dateisysteme und die angegebenen Optionen nötig sind.

Zusätzliche Kernel-Module können unter den *linux-modules* aufgeführt werden. Diese werden zur initrd hinzugefügt und zur Boot-Zeit in der Reihenfolge geladen, in der sie angegeben wurden.

Selbstverständlich betten die hier erzeugten und benutzten initrds ein statisch gebundenes Guile ein und das Initialisierungsprogramm ist ein Guile-Programm. Dadurch haben wir viel Flexibilität. Die Prozedur `expression->initrd` erstellt eine solche initrd für ein an sie übergebenes Programm.

`expression->initrd` *G-Ausdruck* [*#:guile %guile-static-stripped*] [*#:name* "*guile-initrd*"] *Liefert eine* [Scheme-Prozedur]
Linux-initrd (d.h. ein gzip-komprimiertes cpio-Archiv) als dateiartiges Objekt, in dem *guile* enthalten ist, womit der *G-Ausdruck* nach dem Booten ausgewertet wird. Alle vom *G-Ausdruck* referenzierten Ableitungen werden automatisch in die initrd kopiert.

12.14 Bootloader-Konfiguration

Das Betriebssystem unterstützt mehrere Bootloader. Der gewünschte Bootloader wird mit der `bootloader-configuration`-Deklaration konfiguriert. Alle Felder dieser Struktur sind für alle Bootloader gleich außer dem einen Feld `bootloader`, das angibt, welcher Bootloader konfiguriert und installiert werden soll.

Manche der Bootloader setzen nicht alle Felder einer `bootloader-configuration` um. Zum Beispiel ignoriert der extlinux-Bootloader das `theme`-Feld, weil er keine eigenen Themen unterstützt.

`bootloader-configuration` [Datentyp]
Der Typ der Deklaration einer Bootloader-Konfiguration.

`bootloader`
The bootloader to use, as a `bootloader` object. For now `grub-bootloader`, `grub-efi-bootloader`, `grub-efi-netboot-bootloader`, `grub-efi-removable-bootloader`, `extlinux-bootloader` and `u-boot-bootloader` are supported.

Verfügbare Bootloader werden in den Modulen `(gnu bootloader ...)` beschrieben. Insbesondere enthält `(gnu bootloader u-boot)` Definitionen für eine Vielzahl von ARM- und AArch64-Systemen, die den U-Boot-Bootloader (`https://www.denx.de/wiki/U-Boot/`) benutzen.

`grub-efi-bootloader` macht es möglich, auf modernen Systemen mit *Unified Extensible Firmware Interface* (UEFI) zu booten. Sie sollten das hier benutzen, wenn im Installationsabbild ein Verzeichnis `/sys/firmware/efi` vorhanden ist, wenn Sie davon auf Ihrem System booten.

Mit `grub-bootloader` können Sie vor allem auf Intel-basierten Maschinen im alten „Legacy“-BIOS-Modus booten.

Mit `grub-efi-netboot-bootloader` können Sie Ihr System *via* TFTP über das Netzwerk booten. Zusammen mit einem über NFS eingebundenen Wurzeldateisystem können Sie damit ein Guix-System ohne Plattenlaufwerk einrichten.

The installation of the `grub-efi-netboot-bootloader` generates the content of the TFTP root directory at `targets` (siehe Abschnitt 12.14 [Bootloader-Konfiguration], Seite 613), 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.

Wenn Sie außerdem vorhaben, ein NFS-Wurzeldateisystem zu benutzen (eigentlich auch, wenn Sie bloß den Store von einer NFS-Freigabe laden möchten), dann muss der TFTP-Server auch die Datei `/boot/grub/grub.cfg` und die anderen Dateien vom Store zur Verfügung stellen, etwa GRUBs Hintergrundbild, den Kernel (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258) und auch die initrd (siehe Abschnitt 12.2 [„operating-system“-Referenz], Seite 258). Auf all diese Store-Dateien greift GRUB *via* TFTP über ihren normalen Store-Pfad zu, z.B. über `tftp://tftp-server/gnu/store/...-initrd/initrd.cpio.gz`.

Um das möglich zu machen, erzeugt Guix zwei symbolische Verknüpfungen. Für jedes Ziel im Feld `targets` ist die erste Verknüpfung ‘`Ziel`’`/efi/Guix/boot/grub/grub.cfg`, die auf `../../../boot/grub/grub.cfg` zeigt, wobei das ‘`Ziel`’ dem Pfad `/boot` entsprechen kann. In diesem Fall verlässt die Verknüpfung das zugänglich gemachte TFTP-Wurzelverzeichnis *nicht*, in den anderen Fällen schon. Die zweite Verknüpfung ist ‘`Ziel`’`/gnu/store` und zeigt auf `../gnu/store`. Diese Verknüpfung verlässt das zugänglich gemachte TFTP-Wurzelverzeichnis.

Die Annahme hinter all dem ist, dass Sie einen NFS-Server haben, der das Wurzelverzeichnis für Ihr Guix-System exportiert, und außerdem einen TFTP-Server haben, der die als `targets` angegebenen Verzeichnisse liefert – normalerweise ist das ein einzelnes Verzeichnis `/boot` –, was in demselben Wurzelverzeichnis Ihres Guix-Systems gespeichert vorliegt. In dieser Konstellation werden die symbolischen Verknüpfungen funktionieren.

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 to 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.

Abgesehen vom `grub-efi-netboot-bootloader` und den bereits erwähnten TFTP- und NFS-Servern brauchen Sie auch einen passend eingerichteten DHCP-Server, der das Booten über das Netzwerk möglich macht. Derzeit können wir Ihnen bei all dem nur empfehlen, Anleitungen über die PXE (Preboot eXecution Environment) ausfindig zu machen.

Mit `grub-efi-removable-bootloader` lässt sich Ihr System von Wechseldatenträgern aus starten. Er platziert die GRUB-Datei an dem Standardort, der in der UEFI-Spezifikation für solche Fälle vorgesehen ist, nämlich in `/EFI/BOOT/BOOTX64.efi` innerhalb des Boot-Verzeichnisses, was meistens `/boot/efi` ist. `grub-efi-removable-bootloader` ist außerdem geeignet, wenn Sie eine „vergessliche“ UEFI-Firmware haben, die es *nicht* schafft, ihre Konfiguration im dafür gedachten nicht flüchtigen Speicher zu erhalten. Genau wie bei `grub-efi-bootloader` können Sie `grub-efi-removable-bootloader` nur benutzen, wenn das Verzeichnis `/sys/firmware/efi` verfügbar ist.

> **Anmerkung:** *Für jedes* andere Betriebssystem, das seine GRUB-Datei auch an diesem Standardort aus der UEFI-Spezifikation hat, wird diese hierbei überschrieben und das alte System kann *nicht* mehr gebootet werden.

`targets` Eine Liste von Zeichenketten, die angibt, auf welche Ziele der Bootloader installiert werden soll.

Was `targets` bedeutet, hängt vom jeweiligen Bootloader ab. Für `grub-bootloader` sollten hier zum Beispiel Gerätenamen angegeben werden, die vom `installer`-Befehl des Bootloaders verstanden werden, etwa `/dev/sda` oder `(hd0)` (siehe Abschnitt “Invoking grub-install” in *GNU GRUB Manual*). Für `grub-efi-bootloader` und `grub-efi-removable-bootloader` sollten die Einhängepunkte des EFI-Dateisystems angegeben werden, in der Regel `/boot/efi`. Für `grub-efi-netboot-bootloader` sollten `targets` der oder die Einhängepunkte sein, unter denen das TFTP-Wurzelverzeichnis Ihres TFTP-Servers erreichbar ist.

`menu-entries` (Vorgabe: `()`)
Eine möglicherweise leere Liste von `menu-entry`-Objekten (siehe unten), die für Menüeinträge stehen, die im Bootloader-Menü auftauchen sollen, zusätzlich zum aktuellen Systemeintrag und dem auf vorherige Systemgenerationen verweisenden Eintrag.

`default-entry` (Vorgabe: `0`)
Die Position des standardmäßig ausgewählten Bootmenü-Eintrags. An Position 0 steht der Eintrag der aktuellen Systemgeneration.

`timeout` (Vorgabe: `5`)
Wie viele Sekunden lang im Menü auf eine Tastatureingabe gewartet wird, bevor gebootet wird. 0 steht für sofortiges Booten, für -1 wird ohne Zeitbeschränkung gewartet.

`keyboard-layout` (Vorgabe: `#f`)
Wenn dies auf `#f` gesetzt ist, verwendet das Menü des Bootloaders (falls vorhanden) die Vorgabe-Tastaturbelegung, normalerweise US English („qwerty“).

Andernfalls muss es ein `keyboard-layout`-Objekt sein (siehe Abschnitt 12.7 [Tastaturbelegung], Seite 276).

Anmerkung: Dieses Feld wird derzeit von Bootloadern außer `grub` und `grub-efi` ignoriert.

`theme` (Vorgabe: *#f*)
: Ein Objekt für das im Bootloader anzuzeigende Thema. Wird kein Thema angegeben, benutzen manche Bootloader vielleicht ein voreingestelltes Thema; GRUB zumindest macht es so.

`terminal-outputs` (Vorgabe: `'(gfxterm)`)
: Die Ausgabeterminals, die für das Boot-Menü des Bootloaders benutzt werden, als eine Liste von Symbolen. GRUB akzeptiert hier diese Werte: `console`, `serial`, `serial_{0--3}`, `gfxterm`, `vga_text`, `mda_text`, `morse` und `pkmodem`. Dieses Feld entspricht der GRUB-Variablen `GRUB_TERMINAL_OUTPUT` (siehe Abschnitt "Simple configuration" in *Handbuch von GNU GRUB*).

`terminal-inputs` (Vorgabe: `'()`)
: Die Eingabeterminals, die für das Boot-Menü des Bootloaders benutzt werden, als eine Liste von Symbolen. GRUB verwendet hier das zur Laufzeit bestimmte Standardterminal. GRUB akzeptiert sonst diese Werte: `console`, `serial`, `serial_{0-3}`, `at_keyboard` und `usb_keyboard`. Dieses Feld entspricht der GRUB-Variablen `GRUB_TERMINAL_INPUT` (siehe Abschnitt "Simple configuration" in *Handbuch von GNU GRUB*).

`serial-unit` (Vorgabe: `#f`)
: Die serielle Einheit, die der Bootloader benutzt, als eine ganze Zahl zwischen 0 und 3, einschließlich. Für GRUB wird sie automatisch zur Laufzeit ausgewählt; derzeit wählt GRUB die 0 aus, die COM1 entspricht (siehe Abschnitt "Serial terminal" in *Handbuch von GNU GRUB*).

`serial-speed` (Vorgabe: `#f`)
: Die Geschwindigkeit der seriellen Schnittstelle als eine ganze Zahl. GRUB bestimmt den Wert standardmäßig zur Laufzeit; derzeit wählt GRUB 9600 bps (siehe Abschnitt "Serial terminal" in *Handbuch von GNU GRUB*).

`device-tree-support?` (Vorgabe: `#t`)
: Ob das Laden von Device-Tree-Dateien (`https://en.wikipedia.org/wiki/Devicetree`) durch Linux stattfinden soll.

  Diese Option ist standardmäßig aktiviert. In manchen Fällen, z.B. wenn durch den `u-boot`-Bootloader schon der Device Tree in den Arbeitsspeicher geladen wird, kann es gewünscht sein, diese Option hier abzuschalten, indem Sie sie auf `#f` setzen.

Sollten Sie zusätzliche Bootmenü-Einträge über das oben beschriebene `menu-entries`-Feld hinzufügen möchten, müssen Sie diese mit der `menu-entry`-Form erzeugen. Stellen Sie sich zum Beispiel vor, Sie wollten noch eine andere Distribution booten können (schwer vorstellbar!), dann könnten Sie einen Menüeintrag wie den Folgenden definieren:

```
(menu-entry
  (label "Die _andere_ Distribution")
```

```
  (linux "/boot/old/vmlinux-2.6.32")
  (linux-arguments '("root=/dev/sda2"))
  (initrd "/boot/old/initrd"))
```

Details finden Sie unten.

`menu-entry` [Datentyp]

Der Typ eines Eintrags im Bootloadermenü.

`label` Die Beschriftung, die im Menü gezeigt werden soll – z.B. `"GNU"`.

`linux` (Vorgabe: `#f`)
: Das Linux-Kernel-Abbild, was gebootet werden soll, zum Beispiel:

```
(file-append linux-libre "/bzImage")
```

Für GRUB kann hier auch ein Gerät ausdrücklich zum Dateipfad angegeben werden, unter Verwendung von GRUBs Konventionen zur Gerätebenennung (siehe Abschnitt “Naming convention” in *Handbuch von GNU GRUB*), zum Beispiel:

```
"(hd0,msdos1)/boot/vmlinuz"
```

Wenn das Gerät auf diese Weise ausdrücklich angegeben wird, wird das `device`-Feld gänzlich ignoriert.

`linux-arguments` (Vorgabe: `()`)
: Die Liste zusätzlicher Linux-Kernel-Befehlszeilenargumente – z.B. `("console=ttyS0")`.

`initrd` (Vorgabe: `#f`)
: Ein G-Ausdruck oder eine Zeichenkette, die den Dateinamen der initialen RAM-Disk angibt, die benutzt werden soll (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

`device` (Vorgabe: `#f`)
: Das Gerät, auf dem Kernel und initrd zu finden sind – d.h. bei GRUB die Wurzel (*root*) dieses Menüeintrags (siehe Abschnitt “root” in *Handbuch von GNU GRUB*).

Dies kann eine Dateisystembezeichnung (als Zeichenkette), eine Dateisystem-UUID (als Bytevektor, siehe Abschnitt 12.3 [Dateisysteme], Seite 263) oder `#f` sein, im letzten Fall wird der Bootloader auf dem Gerät suchen, das die vom `linux`-Feld benannte Datei enthält (siehe Abschnitt “search” in *Handbuch von GNU GRUB*). Ein vom Betriebssystem vergebener Gerätename wie `/dev/sda1` ist aber *nicht* erlaubt.

`multiboot-kernel` (Vorgabe: `#f`)
: Der Kernel, der im Multiboot-Modus gebootet werden soll (siehe Abschnitt “multiboot” in *GNU GRUB manual*). Wenn dieses Feld gesetzt ist, wird ein Multiboot-Menüeintrag erzeugt. Zum Beispiel:

```
(file-append mach "/boot/gnumach")
```

`multiboot-arguments` (Vorgabe: `()`)
: Liste zusätzlicher Befehlszeilenoptionen für den Multiboot-Kernel.

`multiboot-modules` (Vorgabe: `()`)
: Die Liste der Befehle zum Laden von Multiboot-Modulen. Zum Beispiel:

```
(list (list (file-append hurd "/hurd/ext2fs.static") "ext2fs"
            ...)
      (list (file-append libc "/lib/ld.so.1") "exec"
            ...))
```

`chain-loader` (Vorgabe: `#f`)
: Eine Zeichenkette, die von GRUBs `chainloader`-Direktive akzeptiert wird. Sie hat keine Auswirkungen, wenn auch die Felder `linux` oder `multiboot-kernel` angegeben werden. Im folgenden Beispiel wird ein anderes GNU/Linux-System per Chainloading gebootet.

```
(bootloader
 (bootloader-configuration
  ;; ...
  (menu-entries
   (list
    (menu-entry
     (label "GNU/Linux")
     (device (uuid "1C31-A17C" 'fat))
     (chain-loader "/EFI/GNULinux/grubx64.efi"))))))
```

Zurzeit lässt nur GRUB sein Aussehen durch Themen anpassen. GRUB-Themen werden mit der `grub-theme`-Form erzeugt, die hier noch nicht vollständig dokumentiert ist.

`grub-theme` [Datentyp]
: Der Datentyp, der die Konfiguration des GRUB-Themas repräsentiert.

  `gfxmode` (Vorgabe: `'("auto")`)
  : Welcher `gfxmode` für GRUB eingestellt werden soll (als eine Liste von Zeichenketten mit Bildschirmauflösungen, siehe Abschnitt “gfxmode” in *Handbuch von GNU GRUB*).

`grub-theme` [Scheme-Prozedur]
: Liefert das vorgegebene GRUB-Thema, das vom Betriebssystem benutzt wird, wenn kein `theme`-Feld im `bootloader-configuration`-Verbundsobjekt angegeben wurde.

  Es wird von einem feschen Hintergrundbild begleitet, das die Logos von GNU und Guix zeigt.

Um zum Beispiel eine andere Auflösung als vorgegeben zu verwenden, würden Sie so etwas schreiben:

```
(bootloader
 (bootloader-configuration
 ;; ...
 (theme (grub-theme
         (inherit (grub-theme))
         (gfxmode '("1024x786x32" "auto"))))))
```

12.15 `guix system` aufrufen

Sobald Sie eine Betriebssystemdeklaration geschrieben haben, wie wir sie in den vorangehenden Abschnitten gesehen haben, kann diese *instanziiert* werden, indem Sie den Befehl `guix system` aufrufen. Zusammengefasst:

```
guix system Optionen... Aktion Datei
```

Datei muss der Name einer Datei sein, in der eine Betriebssystemdeklaration als `operating-system`-Objekt steht. *Aktion* gibt an, wie das Betriebssystem instanziiert wird. Derzeit werden folgende Werte dafür unterstützt:

`search` Verfügbare Diensttypendefinitionen anzeigen, die zum angegebenen regulären Ausdruck passen, sortiert nach Relevanz:

```
$ 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

...
```

Wie auch bei `guix package --search` wird das Ergebnis im `recutils`-Format geliefert, so dass es leicht ist, die Ausgabe zu filtern (siehe *Handbuch von GNU recutils*).

`edit` Die Definition des angegebenen Dienstes bearbeiten oder anzeigen.

Beispielsweise öffnet folgender Befehl Ihren Editor, der mit der Umgebungsvariablen `EDITOR` gewählt werden kann, an der Stelle, wo der `openssh`-Diensttyp definiert ist:

```
guix system edit openssh
```

`reconfigure`

Das in der *Datei* beschriebene Betriebssystem erstellen, aktivieren und zu ihm wechseln[7].

> **Anmerkung:** Es ist sehr zu empfehlen, `guix pull` einmal auszuführen, bevor Sie `guix system reconfigure` zum ersten Mal aufrufen (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65). Wenn Sie das nicht tun, könnten Sie nach dem Abschluss von `reconfigure` eine ältere Version von Guix vorfinden, als Sie vorher hatten.

Dieser Befehl setzt die in der *Datei* festgelegte Konfiguration vollständig um: Benutzerkonten, Systemdienste, die Liste globaler Pakete, setuid-Programme und so weiter. Der Befehl startet die in der *Datei* angegebenen Systemdienste, die aktuell nicht laufen; bei aktuell laufenden Diensten wird sichergestellt, dass sie aktualisiert werden, sobald sie das nächste Mal angehalten wurden (z.B. durch `herd stop X` oder `herd restart X`).

Dieser Befehl erzeugt eine neue Generation, deren Nummer (wie `guix system list-generations` sie anzeigt) um eins größer als die der aktuellen Generation ist. Wenn die so nummerierte Generation bereits existiert, wird sie überschrieben. Dieses Verhalten entspricht dem von `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

Des Weiteren wird für den Bootloader ein Menüeintrag für die neue Betriebssystemkonfiguration hinzugefügt, außer die Befehlszeilenoption `--no-bootloader` wurde übergeben. Bei GRUB werden Einträge für ältere Konfigurationen in ein Untermenü verschoben, so dass Sie auch eine ältere Systemgeneration beim Booten noch hochfahren können, falls es notwendig wird.

Nach Abschluss wird das neue System unter `/run/current-system` verfügbar gemacht. Das Verzeichnis enthält *Provenienz-Metadaten*: Dazu gehören die Liste der Kanäle, die benutzt wurden (siehe Kapitel 7 [Kanäle], Seite 77) und die *Datei* selbst, wenn sie verfügbar ist. Sie können sie auf diese Weise ansehen:

```
guix system describe
```

Diese Informationen sind nützlich, falls Sie später inspizieren möchten, wie diese spezielle Generation erstellt wurde. Falls die *Datei* eigenständig ist, also keine anderen Dateien zum Funktionieren braucht, dann können Sie tatsächlich die Generation *n* Ihres Betriebssystems später erneut erstellen mit:

```
guix time-machine \
  -C /var/guix/profiles/system-n-link/channels.scm -- \
  system reconfigure \
```

[7] Diese Aktion (und die dazu ähnlichen Aktionen `switch-generation` und `roll-back`) sind nur auf Systemen nutzbar, auf denen „Guix System“ bereits läuft.

```
/var/guix/profiles/system-n-link/configuration.scm
```

Sie können sich das als eine Art eingebaute Versionskontrolle vorstellen! Ihr System ist nicht nur ein binäres Erzeugnis: *Es enthält seinen eigenen Quellcode.* Siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639, für mehr Informationen zur Provenienzverfolgung.

Nach Voreinstellung können Sie durch `reconfigure` Ihr System *nicht auf eine ältere Version aktualisieren* und somit herabstufen, denn dadurch könnten Sicherheitslücken eingeführt und Probleme mit „zustandsbehafteten“ Diensten wie z.B. Datenbankverwaltungssystemen ausgelöst werden. Sie können das Verhalten ändern, indem Sie `--allow-downgrades` übergeben.

`switch-generation`

Zu einer bestehenden Systemgeneration wechseln. Diese Aktion wechselt das Systemprofil atomar auf die angegebene Systemgeneration. Hiermit werden auch die bestehenden Menüeinträge des Bootloaders umgeordnet. Der Menüeintrag für die angegebene Systemgeneration wird voreingestellt und die Einträge der anderen Generationen werden in ein Untermenü verschoben, sofern der verwendete Bootloader dies unterstützt. Das nächste Mal, wenn das System gestartet wird, wird die hier angegebene Systemgeneration hochgefahren.

Der Bootloader selbst wird durch diesen Befehl *nicht* neu installiert. Es wird also lediglich der bereits installierte Bootloader mit einer neuen Konfigurationsdatei benutzt werden.

Die Zielgeneration kann ausdrücklich über ihre Generationsnummer angegeben werden. Zum Beispiel würde folgender Aufruf einen Wechsel zur Systemgeneration 7 bewirken:

```
guix system switch-generation 7
```

Die Zielgeneration kann auch relativ zur aktuellen Generation angegeben werden, in der Form `+N` oder `-N`, wobei `+3` zum Beispiel „3 Generationen weiter als die aktuelle Generation“ bedeuten würde und `-1` „1 Generation vor der aktuellen Generation“ hieße. Wenn Sie einen negativen Wert wie `-1` angeben, müssen Sie `--` der Befehlszeilenoption voranstellen, damit die negative Zahl nicht selbst als Befehlszeilenoption aufgefasst wird. Zum Beispiel:

```
guix system switch-generation -- -1
```

Zurzeit bewirkt ein Aufruf dieser Aktion *nur* einen Wechsel des Systemprofils auf eine bereits existierende Generation und ein Umordnen der Bootloader-Menüeinträge. Um die Ziel-Systemgeneration aber tatsächlich zu benutzen, müssen Sie Ihr System neu hochfahren, nachdem Sie diese Aktion ausgeführt haben. In einer zukünftigen Version von Guix wird diese Aktion einmal dieselben Dinge tun, wie `reconfigure`, also etwa Dienste aktivieren und deaktivieren.

Diese Aktion schlägt fehl, wenn die angegebene Generation nicht existiert.

`roll-back`

Zur vorhergehenden Systemgeneration wechseln. Wenn das System das nächste Mal hochgefahren wird, wird es die vorhergehende Systemgeneration benutzen. Dies ist die Umkehrung von `reconfigure` und tut genau dasselbe wie `switch-generation` mit dem Argument `-1` aufzurufen.

Wie auch bei `switch-generation` müssen Sie derzeit, nachdem Sie diese Aktion aufgerufen haben, Ihr System neu starten, um die vorhergehende Systemgeneration auch tatsächlich zu benutzen.

`delete-generations`
: Systemgenerationen löschen, wodurch diese zu Kandidaten für den Müllsammler werden (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61, für Informationen, wie Sie den „Müllsammler“ laufen lassen).

  Es funktioniert auf die gleiche Weise wie ‘`guix package --delete-generations`’ (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Wenn keine Argumente angegeben werden, werden alle Systemgenerationen außer der aktuellen gelöscht:

  ```
  guix system delete-generations
  ```

  Sie können auch eine Auswahl treffen, welche Generationen Sie löschen möchten. Das folgende Beispiel hat die Löschung aller Systemgenerationen zur Folge, die älter als zwei Monate sind:

  ```
  guix system delete-generations 2m
  ```

  Wenn Sie diesen Befehl ausführen, wird automatisch der Bootloader mit einer aktualisierten Liste von Menüeinträgen neu erstellt – z.B. werden im Untermenü für die „alten Generationen“ in GRUB die gelöschten Generationen nicht mehr aufgeführt.

`build`
: Die Ableitung des Betriebssystems erstellen, einschließlich aller Konfigurationsdateien und Programme, die zum Booten und Starten benötigt werden. Diese Aktion installiert jedoch nichts davon.

`init`
: In das angegebene Verzeichnis alle Dateien einfügen, um das in der *Datei* angegebene Betriebssystem starten zu können. Dies ist nützlich bei erstmaligen Installationen von „Guix System“. Zum Beispiel:

  ```
  guix system init my-os-config.scm /mnt
  ```

  Hiermit werden alle Store-Objekte nach `/mnt` kopiert, die von der in `my-os-config.scm` angegebenen Konfiguration vorausgesetzt werden. Dazu gehören Konfigurationsdateien, Pakete und so weiter. Auch andere essenzielle Dateien, die auf dem System vorhanden sein müssen, damit es richtig funktioniert, werden erzeugt – z.B. die Verzeichnisse `/etc`, `/var` und `/run` und die Datei `/bin/sh`.

  Dieser Befehl installiert auch den Bootloader auf jedem in `my-os-config` mit `targets` angegebenen Ziel, außer die Befehlszeilenoption `--no-bootloader` wurde übergeben.

`vm`
: Eine virtuelle Maschine (VM) erstellen, die das in der *Datei* deklarierte Betriebssystem enthält, und ein Skript liefern, das diese virtuelle Maschine startet.

  > **Anmerkung:** Die Aktion `vm` sowie solche, die weiter unten genannt werden, können KVM-Unterstützung im Kernel Linux-libre ausnutzen. Insbesondere sollte, wenn die Maschine Hardware-Virtualisierung unterstützt, das entsprechende KVM-Kernelmodul geladen sein und das Gerät `/dev/kvm` muss dann existieren und

dem Benutzer und den Erstellungsbenutzern des Daemons müssen Berechtigungen zum Lesen und Schreiben darauf gegeben werden (siehe Abschnitt 2.4.1 [Einrichten der Erstellungsumgebung], Seite 11).

An das Skript übergebene Argumente werden an QEMU weitergereicht, wie Sie am folgenden Beispiel sehen können. Damit würde eine Netzwerkverbindung aktiviert und 1 GiB an RAM für die emulierte Maschine angefragt:

```
$ /gnu/store/...-run-vm.sh -m 1024 -smp 2 -nic user,model=virtio-net-pci
```

Beide Schritte können zu einem kombiniert werden:

```
$ $(guix system vm my-config.scm) -m 1024 -smp 2 -nic user,model=virtio-net-
```

Die virtuelle Maschine verwendet denselben Store wie das Wirtssystem.

Vorgegeben ist, für die VM das Wurzeldateisystem als flüchtig („volatile“) einzubinden. Mit der Befehlszeilenoption `--persistent` werden Änderungen stattdessen dauerhaft. In diesem Fall wird das Disk-Image der VM aus dem Store in das Verzeichnis `TMPDIR` kopiert, damit Schreibzugriff darauf besteht.

Mit den Befehlszeilenoptionen `--share` und `--expose` können weitere Dateisysteme zwischen dem Wirtssystem und der VM geteilt werden: Der erste Befehl gibt ein mit Schreibzugriff zu teilendes Verzeichnis an, während der letzte Befehl nur Lesezugriff auf das gemeinsame Verzeichnis gestattet.

Im folgenden Beispiel wird eine virtuelle Maschine erzeugt, die auf das Persönliche Verzeichnis des Benutzers nur Lesezugriff hat, wo das Verzeichnis `/austausch` aber mit Lese- und Schreibzugriff dem Verzeichnis `$HOME/tmp` auf dem Wirtssystem zugeordnet wurde:

```
guix system vm my-config.scm \
   --expose=$HOME --share=$HOME/tmp=/austausch
```

Für GNU/Linux ist das vorgegebene Verhalten, direkt in den Kernel zu booten, wodurch nur ein sehr winziges „Disk-Image“ (eine Datei mit einem Abbild des Plattenspeichers der virtuellen Maschine) für das Wurzeldateisystem nötig wird, weil der Store des Wirtssystems davon eingebunden werden kann.

Mit der Befehlszeilenoption `--full-boot` wird erzwungen, einen vollständigen Bootvorgang durchzuführen, angefangen mit dem Bootloader. Dadurch wird mehr Plattenplatz verbraucht, weil dazu ein Disk-Image mindestens mit dem Kernel, initrd und Bootloader-Datendateien erzeugt werden muss.

Mit der Befehlszeilenoption `--image-size` kann die Größe des Disk-Images angegeben werden.

Mit der Befehlszeilenoption `--no-graphic` wird `guix system` angewiesen, eine VM ohne Oberfläche anzulegen, bei der die Konsole für Ein-/Ausgabe benutzt wird, über die die VM gestartet wurde. Unter anderem wird damit eine Zwischenablage zum Kopieren und Einfügen und ein Zeilenpuffer (Scrollback) angeboten. Wenn Sie *`strg`*`-a` zuvor drücken, können Sie QEMU-Befehle angeben, z.B. zeigt *`strg`*`-a` *`h`* eine Hilfe an und *`strg`*`-a` *`x`* beendet die VM; mit *`strg`*`-a` *`c`* wechseln Sie zwischen der QEMU-Anzeige und der VM.

`image` Mit dem Befehl `image` können mehrere Abbildtypen erzeugt werden. Der Abbildtyp kann durch die Befehlszeilenoption `--image-type` ausgewählt werden.

Vorgegeben ist `efi-raw`. Für den Wert `iso9660` kann mit der Befehlszeilenoption `--label` eine Datenträgerkennung („Volume ID“) angegeben werden. Nach Vorgabe wird das Wurzeldateisystem eines Abbilds als nicht-flüchtig eingebunden. Wenn die Befehlszeilenoption `--volatile` angegeben wird, dann sind Änderungen daran bloß flüchtig. Bei der Verwendung von `image` wird auf dem erzeugten Abbild derjenige Bootloader installiert, der in der `operating-system`-Definition angegeben wurde. Das folgende Beispiel zeigt, wie Sie ein Abbild erzeugen, das als Bootloader `grub-efi-bootloader` benutzt, und es mit QEMU starten:

```
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)/share/firmware/ovmf_x64.bin
```

Wenn Sie den Abbildtyp `efi-raw` anfordern, wird ein rohes Disk-Image hergestellt; es kann zum Beispiel auf einen USB-Stick kopiert werden. Angenommen `/dev/sdc` ist das dem USB-Stick entsprechende Gerät, dann kann das Disk-Image mit dem folgenden Befehls darauf kopiert werden:

```
# dd if=$(guix system image my-os.scm) of=/dev/sdc status=progress
```

Durch die Befehlszeilenoption `--list-image-types` werden alle verfügbaren Abbildtypen aufgelistet.

Wenn Sie den Abbildtypen `qcow2` benutzen, wird ein Abbild im qcow2-Format geliefert, das vom QEMU-Emulator effizient benutzt werden kann. Siehe Abschnitt 12.17 [Guix in einer VM starten], Seite 633, für weitere Informationen, wie Sie das Abbild in einer virtuellen Maschine ausführen. Als Bootloader wird immer `grub-bootloader` benutzt, egal was in der als Argument übergebenen `operating-system`-Datei deklariert wurde. Der Grund dafür ist, dass dieser besser mit QEMU funktioniert, das als BIOS nach Voreinstellung SeaBIOS benutzt, wo erwartet wird, dass ein Bootloader in den Master Boot Record (MBR) installiert wurde.

Wenn Sie den Abbildtyp `docker` anfordern, wird ein Abbild für Docker hergestellt. Guix erstellt das Abbild von Grund auf und *nicht* aus einem vorerstellten Docker-Basisabbild heraus, daher enthält es *exakt* das, was Sie in der Konfigurationsdatei für das Betriebssystem angegeben haben. Sie können das Abbild dann wie folgt laden und einen Docker-Container damit erzeugen:

```
image_id="$(docker load < guix-system-docker-image.tar.gz)"
container_id="$(docker create $image_id)"
docker start $container_id
```

Dieser Befehl startet einen neuen Docker-Container aus dem angegebenen Abbild. Damit wird das Guix-System auf die normale Weise hochgefahren, d.h. zunächst werden alle Dienste gestartet, die Sie in der Konfiguration des Betriebssystems angegeben haben. Sie können eine interaktive Shell in dieser isolierten Umgebung bekommen, indem Sie `docker exec` benutzen:

```
docker exec -ti $container_id /run/current-system/profile/bin/bash --login
```

Je nachdem, was Sie im Docker-Container ausführen, kann es nötig sein, dass Sie ihn mit weitergehenden Berechtigungen ausstatten. Wenn Sie zum Beispiel Software mit Guix innerhalb des Docker-Containers erstellen wollen, müssen Sie an `docker create` die Befehlszeilenoption `--privileged` übergeben.

Zuletzt bewirkt die Befehlszeilenoption `--network`, dass durch `guix system docker-image` ein Abbild erzeugt wird, was sein Netzwerk mit dem Wirtssystem teilt und daher auf Dienste wie nscd oder NetworkManager verzichten sollte.

`container`

Liefert ein Skript, um das in der *Datei* deklarierte Betriebssystem in einem Container auszuführen. Mit Container wird hier eine Reihe ressourcenschonender Isolierungsmechanismen im Kernel Linux-libre bezeichnet. Container beanspruchen wesentlich weniger Ressourcen als vollumfängliche virtuelle Maschinen, weil der Kernel, Bibliotheken in gemeinsam nutzbaren Objektdateien („Shared Objects“) sowie andere Ressourcen mit dem Wirtssystem geteilt werden können. Damit ist also eine „dünnere“ Isolierung möglich.

Zurzeit muss das Skript als Administratornutzer „root“ ausgeführt werden, damit darin mehr als nur ein einzelner Benutzer und eine Benutzergruppe unterstützt wird. Der Container teilt seinen Store mit dem Wirtssystem.

Wie bei der Aktion `vm` (siehe [guix system vm], Seite 622) können zusätzlich weitere Dateisysteme zwischen Wirt und Container geteilt werden, indem man die Befehlszeilenoptionen `--share` und `--expose` verwendet:

```
guix system container my-config.scm \
   --expose=$HOME --share=$HOME/tmp=/austausch
```

Die Befehlszeilenoptionen `--share` und `--expose` können Sie auch an das erzeugte Skript übergeben, um weitere Verzeichnisse im Container als Verzeichniseinbindung („bind mount“) verfügbar zu machen.

> **Anmerkung:** Diese Befehlszeilenoption funktioniert nur mit Linux-libre 3.19 oder neuer.

Unter den *Optionen* können beliebige gemeinsame Erstellungsoptionen aufgeführt werden (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189). Des Weiteren kann als *Optionen* Folgendes angegeben werden:

`--expression=`*Ausdruck*
`-e` *Ausdruck*

Als Konfiguration des Betriebssystems das `operating-system`-Objekt betrachten, zu dem der *Ausdruck* ausgewertet wird. Dies ist eine Alternative dazu, die Konfiguration in einer Datei festzulegen. Hiermit wird auch das Installationsabbild des Guix-Systems erstellt, siehe Abschnitt 3.9 [Ein Abbild zur Installation erstellen], Seite 38).

`--system=`*System*
`-s` *System* Versuchen, für das angegebene *System* statt für denselben Systemtyp wie auf dem Wirtssystem zu erstellen. Dies funktioniert wie bei `guix build` (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189).

`--target=`*Tripel*
: Lässt für das angegebene *Tripel* cross-erstellen, dieses muss ein gültiges GNU-Tripel wie z.B. `"aarch64-linux-gnu"` sein (siehe Abschnitt “Specifying target triplets” in *Autoconf*).

`--derivation`
`-d`
: Liefert den Namen der Ableitungsdatei für das angegebene Betriebssystem, ohne dazu etwas zu erstellen.

`--save-provenance`
: Wie zuvor erläutert, speichern `guix system init` und `guix system reconfigure` Provenienzinformationen immer über einen dedizierten Dienst (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639). Andere Befehle tun das nach Voreinstellung jedoch *nicht*. Wenn Sie zum Beispiel ein Abbild für eine virtuelle Maschine mitsamt Provenienzinformationen erzeugen möchten, können Sie dies ausführen:

```
guix system image -t qcow2 --save-provenance config.scm
```

Auf diese Weise wird im erzeugten Abbild im Prinzip „sein eigener Quellcode eingebettet“, in Form von Metadaten in `/run/current-system`. Mit diesen Informationen kann man das Abbild neu erzeugen, um sicherzugehen, dass es tatsächlich das enthält, was davon behauptet wird. Man könnte damit auch eine Abwandlung des Abbilds erzeugen.

`--image-type=`*Typ*
`-t` *Typ*
: Für die Aktion `image` wird ein Abbild des angegebenen *Typ*s erzeugt.

Wird diese Befehlszeilenoption nicht angegeben, so benutzt `guix system` den Abbildtyp `efi-raw`.

`--image-type=iso9660` erzeugt ein Abbild im Format ISO-9660, was für das Brennen auf CDs und DVDs geeignet ist.

`--image-size=`*Größe*
: Für die Aktion `image` wird hiermit festgelegt, dass ein Abbild der angegebenen *Größe* erstellt werden soll. Die *Größe* kann als Zahl die Anzahl Bytes angeben oder mit einer Einheit als Suffix versehen werden (siehe Abschnitt “Block size” in *GNU Coreutils*).

Wird keine solche Befehlszeilenoption angegeben, berechnet `guix system` eine Schätzung der Abbildgröße anhand der Größe des in der *Datei* deklarierten Systems.

`--network`
`-N`
: Für die Aktion `container` dürfen isolierte Umgebungen (auch bekannt als „Container“) auf das Wirtsnetzwerk zugreifen, d.h. es wird kein Netzwerknamensraum für sie erzeugt.

`--root=`*Datei*
`-r` *Datei*
: Die *Datei* zu einer symbolischen Verknüpfung auf das Ergebnis machen und als Müllsammlerwurzel registrieren.

`--skip-checks`
: Die Konfiguration *nicht* vor der Installation zur Sicherheit auf Fehler prüfen.

Das vorgegebene Verhalten von `guix system init` und `guix system reconfigure` sieht vor, die Konfiguration zur Sicherheit auf Fehler hin zu überprüfen, die ihr Autor übersehen haben könnte: Es wird sichergestellt, dass die in der `operating-system`-Deklaration erwähnten Dateisysteme tatsächlich existieren (siehe Abschnitt 12.3 [Dateisysteme], Seite 263) und dass alle Linux-Kernelmodule, die beim Booten benötigt werden könnten, auch im `initrd-modules`-Feld aufgeführt sind (siehe Abschnitt 12.13 [Initiale RAM-Disk], Seite 609). Mit dieser Befehlszeilenoption werden diese Tests allesamt übersprungen.

`--allow-downgrades`
An `guix system reconfigure` die Anweisung erteilen, Systemherabstufungen zuzulassen.

Nach Voreinstellung können Sie mit `reconfigure` Ihr System nicht auf eine ältere Version aktualisieren und somit herabstufen. Dazu werden die Provenienzinformationen Ihres Systems herangezogen (wie sie auch von `guix system describe` angezeigt werden) und mit denen aus Ihrem `guix`-Befehl verglichen (wie sie `guix describe` anzeigt). Wenn die Commits von `guix` nicht Nachfolger derer Ihres Systems sind, dann bricht `guix system reconfigure` mit einer Fehlermeldung ab. Wenn Sie `--allow-downgrades` übergeben, können Sie diese Sicherheitsüberprüfungen umgehen.

> **Anmerkung:** Sie sollten verstehen, was es für die Sicherheit Ihres Rechners bedeutet, ehe Sie `--allow-downgrades` benutzen.

`--on-error=`*Strategie*
Beim Auftreten eines Fehlers beim Einlesen der *Datei* die angegebene *Strategie* verfolgen. Als *Strategie* dient eine der Folgenden:

`nothing-special`
Nichts besonderes; der Fehler wird kurz gemeldet und der Vorgang abgebrochen. Dies ist die vorgegebene Strategie.

`backtrace`
Ebenso, aber zusätzlich wird eine Rückverfolgung des Fehlers (ein „Backtrace“) angezeigt.

`debug` Nach dem Melden des Fehlers wird der Debugger von Guile zur Fehlersuche gestartet. Von dort können Sie Befehle ausführen, zum Beispiel können Sie sich mit `,bt` eine Rückverfolgung („Backtrace“) anzeigen lassen und mit `,locals` die Werte lokaler Variabler anzeigen lassen. Im Allgemeinen können Sie mit Befehlen den Zustand des Programms inspizieren. Siehe Abschnitt “Debug Commands” in *Referenzhandbuch zu GNU Guile* für eine Liste verfügbarer Befehle zur Fehlersuche.

Sobald Sie Ihre Guix-Installation erstellt, konfiguriert, neu konfiguriert und nochmals neu konfiguriert haben, finden Sie es vielleicht hilfreich, sich die auf der Platte verfügbaren – und im Bootmenü des Bootloaders auswählbaren – Systemgenerationen auflisten zu lassen:

`describe` Die laufende Systemgeneration beschreiben: ihren Dateinamen, den Kernel und den benutzten Bootloader etc. sowie Provenienzinformationen, falls verfügbar.

Die Befehlszeilenoption `--list-installed` hat dieselbe Syntax wie bei `guix package --list-installed` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Wenn die Befehlszeilenoption angegeben wird, wird in der Beschreibung eine Liste der Pakete angezeigt, die aktuell im Systemprofil installiert sind. Optional kann die Liste mit einem regulären Ausdruck gefiltert werden.

Anmerkung: Mit der *laufenden* Systemgeneration – die, auf die `/run/current-system` verweist – ist *nicht* unbedingt die *aktuelle* Systemgeneration gemeint, auf die `/var/guix/profiles/system` verweist: Sie unterscheiden sich, wenn Sie zum Beispiel im Bootloader-Menü eine ältere Generation starten lassen.

Auch die *gebootete* Systemgeneration – auf die `/run/booted-system` verweist – ist vielleicht eine andere, etwa wenn Sie das System in der Zwischenzeit rekonfiguriert haben.

`list-generations`

Eine für Menschen verständliche Zusammenfassung jeder auf der Platte verfügbaren Generation des Betriebssystems ausgeben. Dies ähnelt der Befehlszeilenoption `--list-generations` von `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

Optional kann ein Muster angegeben werden, was dieselbe Syntax wie `guix package --list-generations` benutzt, um damit die Liste anzuzeigender Generationen einzuschränken. Zum Beispiel zeigt der folgende Befehl Generationen an, die bis zu 10 Tage alt sind:

```
$ guix system list-generations 10d
```

Sie können als Befehlszeilenoption `--list-installed` angeben mit derselben Syntax wie in `guix package --list-installed`. Das kann nützlich sein, wenn Sie herausfinden möchten, wann ein bestimmtes Paket zum System hinzukam.

Der Befehl `guix system` hat sogar noch mehr zu bieten! Mit folgenden Unterbefehlen wird Ihnen visualisiert, wie Ihre Systemdienste voneinander abhängen:

`extension-graph`

Auf die Standardausgabe den *Diensterweiterungsgraphen* des in der *Datei* definierten Betriebssystems ausgeben (siehe Abschnitt 12.18.1 [Dienstkompositionen], Seite 635, für mehr Informationen zu Diensterweiterungen). Vorgegeben ist, ihn im Dot-/Graphviz-Format auszugeben, aber Sie können ein anderes Format mit `--graph-backend` auswählen, genau wie bei `guix graph` (siehe Abschnitt 10.10 [Aufruf von guix graph], Seite 228):

Der Befehl:

```
$ guix system extension-graph Datei | xdot -
```

zeigt die Erweiterungsrelation unter Diensten an.

Anmerkung: Sie finden das Programm `dot` im Paket `graphviz`.

`shepherd-graph`

Auf die Standardausgabe den *Abhängigkeitsgraphen* der Shepherd-Dienste des in der *Datei* definierten Betriebssystems ausgeben. Siehe Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644, für mehr Informationen sowie einen Beispielgraphen.

Auch hier wird nach Vorgabe die Ausgabe im Dot-/Graphviz-Format sein, aber Sie können ein anderes Format mit `--graph-backend` auswählen.

12.16 `guix deploy` aufrufen

Wir haben bereits gesehen, wie die Konfiguration einer Maschine mit `operating-system`-Deklarationen lokal verwaltet werden kann. Doch was ist, wenn Sie mehrere Maschinen konfigurieren möchten? Vielleicht kümmern Sie sich um einen Dienst im Web, der über mehrere Server verteilt ist. Mit `guix deploy` können Sie ebensolche `operating-system`-Deklarationen verwenden, um mehrere entfernte Rechner („Hosts“) gleichzeitig in einer logischen Bereitstellung (einem „Deployment“) zu verwalten.

> **Anmerkung:** Die in diesem Abschnitt beschriebenen Funktionalitäten befinden sich noch in der Entwicklung und können sich ändern. Kontaktieren Sie uns auf `guix-devel@gnu.org`!

`guix deploy` *Datei*

Mit einem solchen Aufruf werden diejenigen „Maschinen“ bereitgestellt, zu denen der Code in der *Datei* ausgewertet wird. Zum Beispiel könnte die *Datei* eine solche Definition enthalten:

```
;; Dies ist eine Guix-Bereitstellung einer minimalen Installation ohne
;; X11-Anzeigeserver auf eine Maschine, auf der ein SSH-Daemon auf
;; localhost:2222 lauscht. So eine Konfiguration kann für virtuelle
;; Maschinen geeignet sein, deren Ports an die Loopback-Schnittstelle
;; ihres Wirtssystems weitergeleitet werden.

(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 %system)
       (environment managed-host-environment-type)
       (configuration (machine-ssh-configuration
                       (host-name "localhost")
                       (system "x86_64-linux")
                       (user "alice")
                       (identity "./id_rsa")
                       (port 2222)))))
```

Die Datei sollte zu einer Liste von *machine*-Objekten ausgewertet werden. Wenn dieses Beispiel aufgespielt wird, befindet sich danach eine neue Generation auf dem entfernten System, die der `operating-system`-Deklaration `%system` entspricht. Mit `environment` und `configuration` wird die Methode zur Belieferung der Maschine („Provisioning“) angegeben, d.h. wie sie angesteuert werden soll, um dort Rechenressourcen anzulegen und zu verwalten. Im obigen Beispiel werden keine Ressourcen angelegt, weil `'managed-host` für eine Maschine mit bereits laufendem Guix-System steht, auf die über das Netzwerk zugegriffen werden kann. Das ist ein besonders einfacher Fall; zu einer komplexeren Bereitstellung könnte das Starten virtueller Maschinen über einen Anbieter für „Virtual Private Servers“ (VPS) gehören. In einem solchen Fall würde eine andere Art von Umgebung als *environment* angegeben werden.

Beachten Sie, dass Sie zunächst ein Schlüsselpaar auf der Koordinatormaschine erzeugen lassen müssen, damit der Daemon signierte Archive mit Dateien aus dem Store exportieren kann (siehe Abschnitt 6.10 [Aufruf von guix archive], Seite 74). Auf Guix System geschieht dies automatisch.

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

Jede Zielmaschine muss den Schlüssel der Hauptmaschine autorisieren, damit diese Store-Objekte von der Koordinatormaschine empfangen kann:

```
# guix archive --authorize < öffentlicher-schlüssel-koordinatormaschine.txt
```

In dem Beispiel wird unter `user` der Name des Benutzerkontos angegeben, mit dem Sie sich zum Aufspielen auf der Maschine anmelden. Der Vorgabewert ist `root`, jedoch wird eine Anmeldung als root über SSH manchmal nicht zugelassen. Als Ausweg kann `guix deploy` Sie erst mit einem „unprivilegierten“ Benutzerkonto ohne besondere Berechtigungen anmelden, um danach automatisch mit `sudo` die Berechtigungen auszuweiten. Damit das funktioniert, muss `sudo` auf der entfernten Maschine installiert und durch das `user`-Konto ohne Nutzerinteraktion aufrufbar sein; mit anderen Worten muss die Zeile in `sudoers`, die das `user`-Benutzerkonto zum Aufruf von `sudo` berechtigt, mit dem `NOPASSWD`-Tag versehen sein. Dazu kann das folgende Schnipsel in die Betriebssystemkonfiguration eingefügt werden:

```
(use-modules ...
             (gnu system))   ;macht %sudoers-specification verfügbar

(define %user "benutzername")

(operating-system
```

```
...
(sudoers-file
   (plain-file "sudoers"
               (string-append (plain-file-content %sudoers-specification)
                              (format #f "~a ALL = NOPASSWD: ALL~%"
                                      %user)))))
```

Weitere Informationen über das Format der `sudoers`-Datei erhalten Sie, wenn Sie `man sudoers` ausführen.

Wenn Sie ein System auf einige Maschinen aufgespielt haben, möchten Sie vielleicht alle genannten Maschinen einen Befehl ausführen lassen. Mit der Befehlszeilenoption `--execute` oder `-x` können Sie das bewirken. Folgendes Beispiel zeigt, wie Sie `uname -a` auf allen in der Bereitstellungsdatei aufgelisteten Maschinen ausführen:

```
guix deploy Datei -x -- uname -a
```

Eine Sache, die häufig nach dem Aufspielen zu tun ist, ist, bestimmte Dienste auf allen Maschinen neu zu starten. Das geht so:

```
guix deploy Datei -x -- herd restart Dienst
```

Der Befehl `guix deploy -x` liefert genau dann null zurück, wenn sein Befehl auf allen Maschinen erfolgreich war.

Hier sehen Sie die Datentypen, die Sie kennen müssen, wenn Sie eine Bereitstellungsdatei schreiben.

`machine` [Datentyp]

Dieser Datentyp steht für eine einzelne Maschine beim Bereitstellen auf mehrere verschiedene Maschinen.

`operating-system`
: Das Objekt mit der aufzuspielenden Betriebssystemkonfiguration.

`environment`
: Auf welcher Art von Umgebung die Maschine läuft. Der hier angegebene `environment-type` steht dafür, wie die Maschine beliefert wird („Provisioning“).

`configuration` (Vorgabe: `#f`)
: Dieses Objekt gibt die bestimmte Konfiguration der Umgebung (`environment`) der Maschine an. Falls es für diese Umgebung eine Vorgabekonfiguration gibt, kann auch `#f` benutzt werden. Wenn `#f` für eine Umgebung ohne Vorgabekonfiguration benutzt wird, wird ein Fehler ausgelöst.

`machine-ssh-configuration` [Datentyp]

Dieser Datentyp repräsentiert die SSH-Client-Parameter einer Maschine, deren Umgebung (`environment`) vom Typ `managed-host-environment-type` ist.

`host-name`

`build-locally?` (Vorgabe: `#t`)
: Wenn es falsch ist, werden Ableitungen für das System auf der Maschine erstellt, auf die es aufgespielt wird.

`system` Der Systemtyp, der die Architektur der Maschine angibt, auf die aufgespielt wird – z.B. `"x86_64-linux"`.

`authorize?` (Vorgabe: `#t`)
Wenn es wahr ist, wird der Signierschlüssel des Koordinators zum ACL-Schlüsselbund (der Access Control List, deutsch Zugriffssteuerungsliste) der entfernten Maschine hinzugefügt.

`port` (Vorgabe: `22`)
`user` (Vorgabe: `"root"`)
`identity` (Vorgabe: `#f`)
Wenn dies angegeben wird, ist es der Pfad zum privaten SSH-Schlüssel, um sich beim entfernten Rechner zu authentisieren.

`host-key` (Vorgabe: `#f`)
Hierfür sollte der SSH-Rechnerschlüssel der Maschine angegeben werden. Er sieht so aus:

```
ssh-ed25519 AAAAC3Nz... root@example.org
```

Wenn `#f` als `host-key` angegeben wird, wird der Server gegen die Datei `~/.ssh/known_hosts` geprüft, genau wie es der `ssh`-Client von OpenSSH tut.

`allow-downgrades?` (Vorgabe: `#f`)
Ob mögliche Herabstufungen zugelassen werden sollen.

Genau wie `guix system reconfigure` vergleicht auch `guix deploy` die Commits auf den momentan eingespielten Kanälen am entfernten Rechner (wie sie von `guix system describe` gemeldet werden) mit denen, die zurzeit verwendet werden (wie sie `guix describe` anzeigt), um festzustellen, ob aktuell verwendete Commits Nachfolger der aufgespielten sind. Ist das *nicht* der Fall und steht `allow-downgrades?` auf falsch, wird ein Fehler gemeldet. Dadurch kann gewährleistet werden, dass Sie nicht aus Versehen die entfernte Maschine auf eine alte Version herabstufen.

`safety-checks?` (Vorgabe: `#t`)
Ob „Sicherheitsüberprüfungen“ vor dem Aufspielen durchgeführt werden. Dazu gehört, zu prüfen, ob die Geräte und Dateisysteme in der Betriebssystemkonfiguration tatsächlich auf der Zielmaschine vorhanden sind und die Linux-Module, die gebrauchy werden, um auf Speichergeräte beim Booten zuzugreifen, auch im Feld `initrd-modules` des `operating-system` aufgelistet sind.

Durch die Sicherheitsüberprüfungen wird gewährleistet, dass Sie nicht aus Versehen ein System bereitstellen, dass gar nicht booten kann. Überlegen Sie sich gut, wenn Sie sie ausschalten!

`digital-ocean-configuration` [Datentyp]
Dieser Datentyp beschreibt das Droplet, das für eine Maschine erzeugt werden soll, deren Umgebung (`environment`) vom Typ `digital-ocean-environment-type` ist.

`ssh-key` Der Pfad zum privaten SSH-Schlüssel, um sich beim entfernten Rechner zu authentisieren. In Zukunft wird es dieses Feld vielleicht nicht mehr geben.

`tags` Eine Liste von „Tags“ als Zeichenketten, die die Maschine eindeutig identifizieren. Sie müssen angegeben werden, damit keine zwei Maschinen in der Bereitstellung dieselbe Menge an Tags haben.

`region` Ein Digital-Ocean-„Region Slug“ (Regionskürzel), zum Beispiel `"nyc3"`.

`Größe` Ein Digital-Ocean-„Size Slug“ (Größenkürzel), zum Beispiel `"s-1vcpu-1gb"`

`enable-ipv6?`
Ob das Droplet mit IPv6-Netzanbindung erzeugt werden soll.

12.17 Guix in einer virtuellen Maschine betreiben

Um Guix in einer virtuellen Maschine (VM) auszuführen, können Sie das vorerstellte Guix-VM-Abbild benutzen, das auf `https://ftp.gnu.org/gnu/guix/guix-system-vm-image-1.4.0.x86_64-linux.qcow2` angeboten wird. Das Abbild ist ein komprimiertes Abbild im QCOW-Format. Sie können es an einen Emulator wie QEMU übergeben (siehe unten für Details).

Dieses Abbild startet eine grafische Xfce-Umgebung und enthält einige oft genutzte Werkzeuge. Sie können im Abbild mehr Software installieren, indem Sie `guix package` in einem Terminal ausführen (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Sie können das System im Abbild auch rekonfigurieren, basierend auf seiner anfänglichen Konfigurationsdatei, die als `/run/current-system/configuration.scm` verfügbar ist (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250).

Statt dieses vorerstellte Abbild zu benutzen, können Sie auch Ihr eigenes Abbild erstellen, indem Sie `guix system image` benutzen (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

Wenn Sie Ihr eigenes Abbild erstellen haben lassen, müssen Sie es aus dem Store herauskopieren (siehe Abschnitt 9.9 [Der Store], Seite 165) und sich darauf Schreibberechtigung geben, um die Kopie benutzen zu können. Wenn Sie QEMU aufrufen, müssen Sie einen Systememulator angeben, der für Ihre Hardware-Plattform passend ist. Hier ist ein minimaler QEMU-Aufruf, der das Ergebnis von `guix system image -t qcow2` auf x86_64-Hardware bootet:

```
$ 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-1.4.0.x86_64-linux.qcow2,id=myhd
```

Die Bedeutung jeder dieser Befehlszeilenoptionen ist folgende:

`qemu-system-x86_64`
Hiermit wird die zu emulierende Hardware-Plattform angegeben. Sie sollte zum Wirtsrechner passen.

`-nic user,model=virtio-net-pci`
Den als Nutzer ausgeführten Netzwerkstapel („User-Mode Network Stack“) ohne besondere Berechtigungen benutzen. Mit dieser Art von Netzwerkanbindung kann das Gast-Betriebssystem eine Verbindung zum Wirt aufbauen, aber nicht

andersherum. Es ist die einfachste Art, das Gast-Betriebssystem mit dem Internet zu verbinden. Das `model` gibt das Modell eines zu emulierenden Netzwerkgeräts an: `virtio-net-pci` ist ein besonderes Gerät, das für virtualisierte Betriebssysteme gedacht ist und für die meisten Anwendungsfälle empfohlen wird. Falls Ihre Hardware-Plattform x86_64 ist, können Sie eine Liste verfügbarer Modelle von Netzwerkkarten (englisch „Network Interface Card“, kurz NIC) einsehen, indem Sie `qemu-system-x86_64 -net nic,model=help` ausführen.

`-enable-kvm`
: Wenn Ihr System über Erweiterungen zur Hardware-Virtualisierung verfügt, beschleunigt es die Dinge, wenn Sie die Virtualisierungsunterstützung „KVM“ des Linux-Kernels benutzen lassen.

`-m 2048` Die Menge an Arbeitsspeicher (RAM), die dem Gastbetriebssystem zur Verfügung stehen soll, in Mebibytes. Vorgegeben wären 128 MiB, was für einige Operationen zu wenig sein könnte.

`-device virtio-blk,drive=myhd`
: Ein `virtio-blk`-Laufwerk namens „myhd“ erzeugen. `virtio-blk` ist ein Mechanismus zur „Paravirtualisierung“ von Blockgeräten, wodurch QEMU diese effizienter benutzen kann, als wenn es ein Laufwerk vollständig emulieren würde. Siehe die Dokumentation von QEMU und KVM für mehr Informationen.

`-drive if=none,file=/tmp/qemu-image,id=myhd`
: Unser QCOW-Abbild in der Datei `/tmp/qemu-image` soll als Inhalt des „myhd“-Laufwerks herhalten.

Das voreingestellte `run-vm.sh`-Skript, das durch einen Aufruf von `guix system vm` erzeugt wird, fügt keine Befehlszeilenoption `-nic user` an. Um innerhalb der virtuellen Maschine Netzwerkzugang zu haben, fügen Sie den `(dhcp-client-service)` zu Ihrer Systemdefinition hinzu und starten Sie die VM mit `$(guix system vm config.scm) -nic user`. Erwähnt werden sollte der Nachteil, dass bei Verwendung von `-nic user` zur Netzanbindung der `ping`-Befehl *nicht* funktionieren wird, weil dieser das ICMP-Protokoll braucht. Sie werden also einen anderen Befehl benutzen müssen, um auszuprobieren, ob Sie mit dem Netzwerk verbunden sind, zum Beispiel `guix download`.

12.17.1 Verbinden über SSH

Um SSH in der virtuellen Maschine zu aktivieren, müssen Sie einen SSH-Server wie `openssh-service-type` zu ihr hinzufügen (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314). Des Weiteren müssen Sie den SSH-Port für das Wirtssystem freigeben (standardmäßig hat er die Portnummer 22). Das geht zum Beispiel so:

```
$(guix system vm config.scm) -nic user,model=virtio-net-pci,hostfwd=tcp::10022-:22
```

Um sich mit der virtuellen Maschine zu verbinden, benutzen Sie diesen Befehl:

```
ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -p 10022 localhost
```

Mit `-p` wird `ssh` der Port mitgeteilt, über den eine Verbindung hergestellt werden soll. `-o UserKnownHostsFile=/dev/null` verhindert, dass `ssh` sich bei jeder Modifikation Ihrer `config.scm`-Datei beschwert, ein anderer bekannter Rechner sei erwartet worden, und `-o StrictHostKeyChecking=no` verhindert, dass Sie die Verbindung zu unbekannten Rechnern jedes Mal bestätigen müssen, wenn Sie sich verbinden.

> **Anmerkung:** Wenn dieses Beispiel zu '`hostfwd`' bei Ihnen nicht funktioniert (weil z.B. sich Ihr SSH-Client aufhängt, wenn Sie versuchen, sich mit dem zugeordneten Port auf Ihrer VM zu verbinden), dann überprüfen Sie nochmal, ob Ihre Guix-System-VM auch mit Netzwerkunterstützung konfiguriert wurde, also etwas wie der Diensttyp `dhcp-client-service-type` angegeben wurde.

12.17.2 `virt-viewer` mit Spice benutzen

Eine Alternative zur grafischen Schnittstelle des standardmäßigen `qemu` ist, sich mit Hilfe des `remote-viewer` aus dem Paket `virt-viewer` zu verbinden. Um eine Verbindung herzustellen, übergeben Sie die Befehlszeilenoption `-spice port=5930,disable-ticketing` an `qemu`. Siehe den vorherigen Abschnitt für weitere Informationen, wie Sie das übergeben.

Spice macht es auch möglich, ein paar nette Hilfestellungen zu benutzen, zum Beispiel können Sie Ihren Zwischenspeicher zum Kopieren und Einfügen (Ihr „Clipboard“) mit Ihrer virtuellen Maschine teilen. Um das zu aktivieren, werden Sie die folgenden Befehlszeilennoptionen zusätzlich an `qemu` übergeben müssen:

```
-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
```

Sie werden auch den `(spice-vdagent-service)` zu Ihrer Systemdefinition hinzufügen müssen (siehe Abschnitt 12.9.36 [Verschiedene Dienste], Seite 593).

12.18 Dienste definieren

Der vorhergehende Abschnitt präsentiert die verfügbaren Dienste und wie man sie in einer `operating-system`-Deklaration kombiniert. Aber wie definieren wir solche Dienste eigentlich? Und was ist überhaupt ein Dienst?

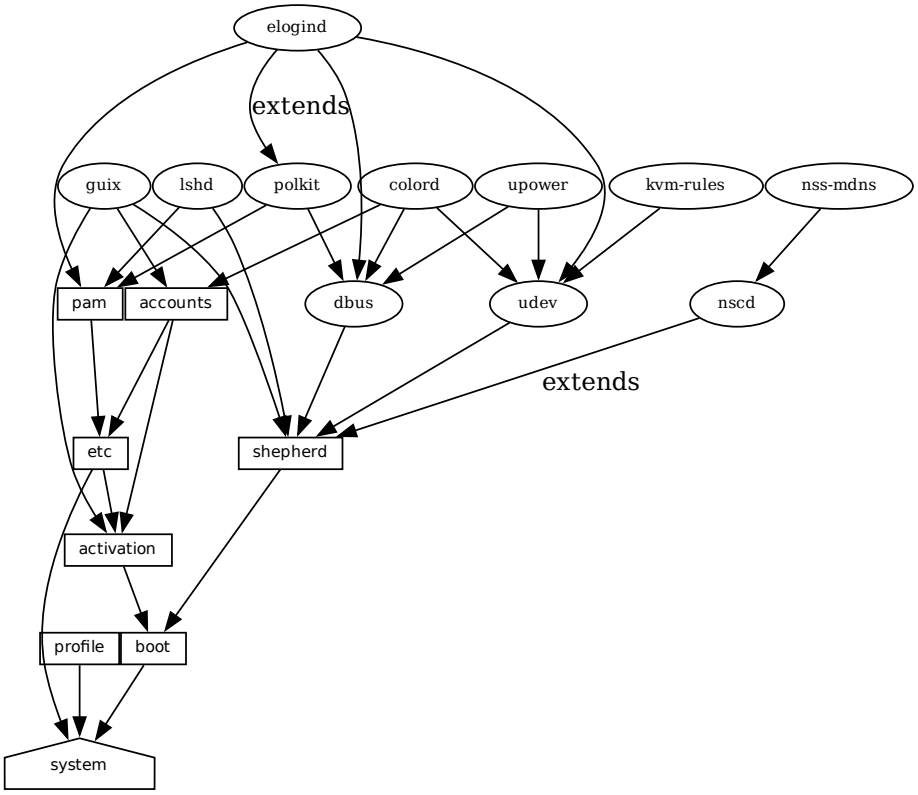
12.18.1 Dienstkompositionen

Wir definieren hier einen *Dienst* (englisch „Service“) als, grob gesagt, etwas, das die Funktionalität des Betriebssystems erweitert. Oft ist ein Dienst ein Prozess – ein sogenannter *Daemon* –, der beim Hochfahren des Systems gestartet wird: ein Secure-Shell-Server, ein Web-Server, der Guix-Erstellungsdaemon usw. Manchmal ist ein Dienst ein Daemon, dessen Ausführung von einem anderen Daemon ausgelöst wird – zum Beispiel wird ein FTP-Server von `inetd` gestartet oder ein D-Bus-Dienst durch `dbus-daemon` aktiviert. Manchmal entspricht ein Dienst aber auch keinem Daemon. Zum Beispiel nimmt sich der Benutzerkonten-Dienst („account service“) die Benutzerkonten und sorgt dafür, dass sie existieren, wenn das System läuft. Der „udev“-Dienst sammelt die Regeln zur Geräteverwaltung an und macht diese für den eudev-Daemon verfügbar. Der `/etc`-Dienst fügt Dateien in das Verzeichnis `/etc` des Systems ein.

Dienste des Guix-Systems werden durch *Erweiterungen* („Extensions“) miteinander verbunden. Zum Beispiel *erweitert* der Secure-Shell-Dienst den Shepherd – Shepherd ist das Initialisierungssystem (auch „init“-System genannt), was als PID 1 läuft –, indem es ihm die Befehlszeilen zum Starten und Stoppen des Secure-Shell-Daemons übergibt (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314). Der UPower-Dienst erweitert den D-Bus-Dienst, indem es ihm seine `.service`-Spezifikation übergibt, und erweitert den udev-Dienst, indem

es ihm Geräteverwaltungsregeln übergibt (siehe Abschnitt 12.9.9 [Desktop-Dienste], Seite 366). Der Guix-Daemon-Dienst erweitert den Shepherd, indem er ihm die Befehlszeilen zum Starten und Stoppen des Daemons übergibt, und er erweitert den Benutzerkontendienst („account service“), indem er ihm eine Liste der benötigten Erstellungsbenutzerkonten übergibt (siehe Abschnitt 12.9.1 [Basisdienste], Seite 281).

Alles in allem bilden Dienste und ihre „Erweitert“-Relationen einen gerichteten azyklischen Graphen (englisch „Directed Acyclic Graph“, kurz DAG). Wenn wir Dienste als Kästen und Erweiterungen als Pfeile darstellen, könnte ein typisches System so etwas hier anbieten:



Ganz unten sehen wir den *Systemdienst*, der das Verzeichnis erzeugt, in dem alles zum Ausführen und Hochfahren enthalten ist, so wie es der Befehl `guix system build` liefert. Siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639, um mehr über die anderen hier gezeigten Diensttypen zu erfahren. Beim [system-extension-graph], Seite 628, finden Sie Informationen darüber, wie Sie diese Darstellung für eine Betriebssystemdefinition Ihrer Wahl generieren lassen.

Technisch funktioniert es so, dass Entwickler *Diensttypen* definieren können, um diese Beziehungen auszudrücken. Im System kann es beliebig viele Dienste zu jedem Typ geben – zum Beispiel können auf einem System zwei Instanzen des GNU-Secure-Shell-Servers

(lsh) laufen, mit zwei Instanzen des Diensttyps `lsh-service-type` mit je unterschiedlichen Parametern.

Der folgende Abschnitt beschreibt die Programmierschnittstelle für Diensttypen und Dienste.

12.18.2 Diensttypen und Dienste

Ein *Diensttyp* („service type“) ist ein Knoten im oben beschriebenen ungerichteten azyklischen Graphen (DAG). Fangen wir an mit einem einfachen Beispiel: dem Diensttyp für den Guix-Erstellungsdaemon (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18):

```
(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))))
```

Damit sind drei Dinge definiert:

1. Ein Name, der nur dazu da ist, dass man leichter die Abläufe verstehen und Fehler suchen kann.
2. Eine Liste von *Diensterweiterungen* („service extensions“). Jede Erweiterung gibt den Ziel-Diensttyp an sowie eine Prozedur, die für gegebene Parameter für den Dienst eine Liste von Objekten zurückliefert, um den Dienst dieses Typs zu erweitern.

   Jeder Diensttyp benutzt mindestens eine Diensterweiterung. Die einzige Ausnahme ist der *boot service type*, der die Grundlage aller Dienste ist.
3. Optional kann ein Vorgabewert für Instanzen dieses Typs angegeben werden.

In diesem Beispiel werden durch `guix-service-type` drei Dienste erweitert:

`shepherd-root-service-type`
: Die Prozedur `guix-shepherd-service` definiert, wie der Shepherd-Dienst erweitert wird, und zwar liefert sie ein `<shepherd-service>`-Objekt, womit definiert wird, wie der `guix-daemon` gestartet und gestoppt werden kann (siehe Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644).

`account-service-type`
: Diese Erweiterung des Dienstes wird durch `guix-accounts` berechnet, eine Prozedur, die eine Liste von `user-group`- und `user-account`-Objekten liefert, die die Erstellungsbenutzerkonten repräsentieren (siehe Abschnitt 2.5 [Aufruf des guix-daemon], Seite 18).

`activation-service-type`
: Hier ist `guix-activation` eine Prozedur, die einen G-Ausdruck liefert. Dieser ist ein Code-Schnipsel, das zur „Aktivierungszeit“ ausgeführt werden soll – z.B. wenn der Dienst hochgefahren wird.

Ein Dienst dieses Typs wird dann so instanziiert:

```
(service guix-service-type
```

```
(guix-configuration
  (build-accounts 5)
  (extra-options '("--gc-keep-derivations"))))
```

Das zweite Argument an die `service`-Form ist ein Wert, der die Parameter dieser bestimmten Dienstinstanz repräsentiert. Siehe [guix-configuration-type], Seite 290, für Informationen über den `guix-configuration`-Datentyp. Wird kein Wert angegeben, wird die Vorgabe verwendet, die im `guix-service-type` angegeben wurde:

```
(service guix-service-type)
```

`guix-service-type` ist ziemlich einfach, weil es andere Dienste erweitert, aber selbst nicht erweitert werden kann.

Der Diensttyp eines *erweiterbaren* Dienstes sieht ungefähr so aus:

```
(define udev-service-type
  (service-type (name 'udev)
                (extensions
                 (list (service-extension shepherd-root-service-type
                                          udev-shepherd-service)))

                (compose concatenate)     ;Liste der Regeln zusammenfügen
                (extend (lambda (config rules) ;Konfiguration und Regeln
                          (match config
                            (($ <udev-configuration> udev initial-rules)
                             (udev-configuration
                              (udev udev) ;zu benutzendes udev-Paket
                              (rules (append initial-rules rules)))))))))
```

Dies ist der Diensttyp für den Geräteverwaltungsdaemon eudev (`https://wiki.gentoo.org/wiki/Project:Eudev`). Verglichen mit dem vorherigen Beispiel sehen wir neben einer Erweiterung des `shepherd-root-service-type` auch zwei neue Felder.

`compose` Die Prozedur, um die Liste der jeweiligen Erweiterungen für den Dienst dieses Typs zu einem Objekt zusammenzustellen (zu „komponieren“, englisch *compose*).

Dienste können den udev-Dienst erweitern, indem sie eine Liste von Regeln („Rules“) an ihn übergeben; wir komponieren mehrere solche Erweiterungen, indem wir die Listen einfach zusammenfügen.

`extend` Diese Prozedur definiert, wie der Wert des Dienstes um die Komposition mit Erweiterungen erweitert („extended“) werden kann.

Udev-Erweiterungen werden zu einer einzigen Liste von Regeln komponiert, aber der Wert des udev-Dienstes ist ein `<udev-configuration>`-Verbundsobjekt. Deshalb erweitern wir diesen Verbund, indem wir die Liste der von Erweiterungen beigetragenen Regeln an die im Verbund gespeicherte Liste der Regeln anhängen.

`description`
Diese Zeichenkette gibt einen Überblick über den Systemtyp. Die Zeichenkette darf mit Texinfo ausgezeichnet werden (siehe Abschnitt “Overview” in *GNU*

Texinfo). Der Befehl `guix system search` durchsucht diese Zeichenketten und zeigt sie an (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

Es kann nur eine Instanz eines erweiterbaren Diensttyps wie `udev-service-type` geben. Wenn es mehrere gäbe, wäre es mehrdeutig, welcher Dienst durch die `service-extension` erweitert werden soll.

Sind Sie noch da? Der nächste Abschnitt gibt Ihnen eine Referenz der Programmierschnittstelle für Dienste.

12.18.3 Service-Referenz

Wir haben bereits einen Überblick über Diensttypen gesehen (siehe Abschnitt 12.18.2 [Diensttypen und Dienste], Seite 637). Dieser Abschnitt hier stellt eine Referenz dar, wie Dienste und Diensttypen manipuliert werden können. Diese Schnittstelle wird vom Modul `(gnu services)` angeboten.

`service` *Typ* [*Wert*] [Scheme-Prozedur]

Liefert einen neuen Dienst des angegebenen *Typ*s. Der *Typ* muss als `<service-type>`-Objekt angegeben werden (siehe unten). Als *Wert* kann ein beliebiges Objekt angegeben werden, das die Parameter dieser bestimmten Instanz dieses Dienstes repräsentiert.

Wenn kein *Wert* angegeben wird, wird der vom *Typ* festgelegte Vorgabewert verwendet; verfügt der *Typ* über keinen Vorgabewert, dann wird ein Fehler gemeldet.

Zum Beispiel bewirken Sie hiermit:

```
(service openssh-service-type)
```

dasselbe wie mit:

```
(service openssh-service-type
         (openssh-configuration))
```

In beiden Fällen ist das Ergebnis eine Instanz von `openssh-service-type` mit der vorgegebenen Konfiguration.

`service?` *Objekt* [Scheme-Prozedur]

Liefert wahr zurück, wenn das *Objekt* ein Dienst ist.

`service-kind` *Dienst* [Scheme-Prozedur]

Liefert den Typ des *Dienst*es – d.h. ein `<service-type>`-Objekt.

`service-value` *Dienst* [Scheme-Prozedur]

Liefert den Wert, der mit dem *Dienst* assoziiert wurde. Er repräsentiert die Parameter des *Dienst*es.

Hier ist ein Beispiel, wie ein Dienst erzeugt und manipuliert werden kann:

```
(define s
  (service nginx-service-type
           (nginx-configuration
            (nginx nginx)
            (log-directory log-Verzeichnis)
            (run-directory run-Verzeichnis)
```

```
              (file config-Datei))))

(service? s)
⇒ #t

(eq? (service-kind s) nginx-service-type)
⇒ #t
```

Die Form `modify-services` ist eine nützliche Methode, die Parameter von einigen der Dienste aus einer Liste wie `%base-services` abzuändern (siehe Abschnitt 12.9.1 [Basisdienste], Seite 281). Sie wird zu einer Liste von Diensten ausgewertet. Natürlich können Sie dazu auch die üblichen Listenkombinatoren wie `map` und `fold` benutzen (siehe Abschnitt "SRFI-1" in *Referenzhandbuch zu GNU Guile*), `modify-services` soll dieses häufig benutzte Muster lediglich durch eine knappere Syntax unterstützen.

`modify-services` *Dienste* (*Typ Variable* `=>` *Rumpf*) ... [Scheme-Syntax]

Passt die von *Dienste* bezeichnete Dienst-Liste entsprechend den angegebenen Klauseln an. Jede Klausel hat die Form:

```
(Typ Variable => Rumpf)
```

wobei *Typ* einen Diensttyp („service type“) bezeichnet – wie zum Beispiel `guix-service-type` – und *Variable* ein Bezeichner ist, der im *Rumpf* an die Dienst-Parameter – z.B. eine `guix-configuration`-Instanz – des ursprünglichen Dienstes mit diesem *Typ* gebunden wird.

Der *Rumpf* muss zu den neuen Dienst-Parametern ausgewertet werden, welche benutzt werden, um den neuen Dienst zu konfigurieren. Dieser neue Dienst wird das Original in der resultierenden Liste ersetzen. Weil die Dienstparameter eines Dienstes mit `define-record-type*` erzeugt werden, können Sie einen kurzen *Rumpf* schreiben, der zu den neuen Dienstparametern ausgewertet wird, indem Sie die Funktionalität namens `inherit` benutzen, die von `define-record-type*` bereitgestellt wird.

Klauseln können auch die folgende Form annehmen:

```
(delete Diensttyp)
```

Mit so einer Klausel werden alle Dienste mit dem angegebenen *Diensttyp* aus der Liste der *Dienste* weggelassen.

Siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250, für ein Anwendungsbeispiel.

Als Nächstes ist die Programmierschnittstelle für Diensttypen an der Reihe. Sie ist etwas, was Sie kennen werden wollen, wenn Sie neue Dienstdefinitionen schreiben, aber wenn Sie nur Ihre `operating-system`-Deklaration anpassen möchten, brauchen Sie diese Schnittstelle wahrscheinlich nicht.

`service-type` [Datentyp]

Die Repräsentation eines *Diensttypen* (siehe Abschnitt 12.18.2 [Diensttypen und Dienste], Seite 637).

`name` Dieses Symbol wird nur verwendet, um die Abläufe im System anzuzeigen und die Fehlersuche zu erleichtern.

`extensions`
: Eine nicht-leere Liste von `<service-extension>`-Objekten (siehe unten).

`compose` (Vorgabe: `#f`)
: Wenn es auf `#f` gesetzt ist, dann definiert der Diensttyp Dienste, die nicht erweitert werden können – d.h. diese Dienste erhalten ihren Wert nicht von anderen Diensten.

  Andernfalls muss es eine Prozedur sein, die ein einziges Argument entgegennimmt. Die Prozedur wird durch `fold-services` aufgerufen und ihr wird die Liste von aus den Erweiterungen angesammelten Werten übergeben. Sie gibt daraufhin einen einzelnen Wert zurück.

`extend` (Vorgabe: `#f`)
: Ist dies auf `#f` gesetzt, dann können Dienste dieses Typs nicht erweitert werden.

  Andernfalls muss es eine zwei Argumente nehmende Prozedur sein, die von `fold-services` mit dem anfänglichen Wert für den Dienst als erstes Argument und dem durch Anwendung von `compose` gelieferten Wert als zweites Argument aufgerufen wird. Als Ergebnis muss ein Wert geliefert werden, der einen zulässigen neuen Parameterwert für die Dienstinstanz darstellt.

`description`
: Eine Zeichenkette, womöglich geschrieben als Texinfo-Markup, die in ein paar Sätzen beschreibt, wofür der Dienst gut ist. Diese Zeichenkette ermöglicht es Nutzern, mittels `guix system search` Informationen über den Dienst zu bekommen (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

`default-value` (Vorgabe: `&no-default-value`)
: Der Vorgabewert, der für Instanzen dieses Diensttyps verwendet wird. Dadurch können Nutzer die `service`-Form ohne ihr zweites Argument benutzen:

  ```
  (service Diensttyp)
  ```

  Der zurückgelieferte Dienst hat dann den durch den *Diensttyp* vorgegebenen Vorgabewert.

Siehe den Abschnitt Abschnitt 12.18.2 [Diensttypen und Dienste], Seite 637, für Beispiele.

`service-extension` *Zieltyp* *Berechner* *Liefert eine neue Erweiterung für den Dienst mit dem* [Scheme-Prozedur]

Zieltyp. Als *Berechner* muss eine Prozedur angegeben werden, die ein einzelnes Argument nimmt: `fold-services` ruft sie auf und übergibt an sie den Wert des erweiternden Dienstes, sie muss dafür einen zulässigen Wert für den *Zieltyp* liefern.

`service-extension?` *Objekt* [Scheme-Prozedur]

Liefert wahr zurück, wenn das *Objekt* eine Diensterweiterung ist.

Manchmal wollen Sie vielleicht einfach nur einen bestehenden Dienst erweitern. Dazu müssten Sie einen neuen Diensttyp definieren und die Erweiterung definieren, für die Sie sich interessieren, was ganz schön wortreich werden kann. Mit der Prozedur `simple-service` können Sie es kürzer fassen.

`simple-service` *Name Zieltyp Wert* [Scheme-Prozedur]
: Liefert einen Dienst, der den Dienst mit dem *Zieltyp* um den *Wert* erweitert. Dazu wird ein Diensttyp mit dem *Name*n für den einmaligen Gebrauch erzeugt, den der zurückgelieferte Dienst instanziiert.

  Zum Beispiel kann mcron (siehe Abschnitt 12.9.2 [Geplante Auftragsausführung], Seite 301) so um einen zusätzlichen Auftrag erweitert werden:

```
(simple-service 'my-mcron-job mcron-service-type
                #~(job '(next-hour (3)) "guix gc -F 2G"))
```

Den Kern dieses abstrakten Modells für Dienste bildet die Prozedur `fold-services`, die für das „Kompilieren“ einer Liste von Diensten hin zu einem einzelnen Verzeichnis verantwortlich ist, in welchem alles enthalten ist, was Sie zum Booten und Hochfahren des Systems brauchen – d.h. das Verzeichnis, das der Befehl `guix system build` anzeigt (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619). Einfach ausgedrückt propagiert `fold-services` Diensterweiterungen durch den Dienstgraphen nach unten und aktualisiert dabei in jedem Knoten des Graphen dessen Parameter, bis nur noch der Wurzelknoten übrig bleibt.

`fold-services` *Dienste* [*#:target-type system-service-type*] [Scheme-Prozedur]
: *Faltet die* `Dienste` *wie die* funktionale Prozedur `fold` zu einem einzigen zusammen, indem ihre Erweiterungen nach unten propagiert werden, bis eine Wurzel vom *target-type* als Diensttyp erreicht wird; dieser so angepasste Wurzeldienst wird zurückgeliefert.

Als Letztes definiert das Modul `(gnu services)` noch mehrere essenzielle Diensttypen, von denen manche im Folgenden aufgelistet sind:

`system-service-type` [Scheme-Variable]
: Die Wurzel des Dienstgraphen. Davon wird das Systemverzeichnis erzeugt, wie es vom Befehl `guix system build` zurückgeliefert wird.

`boot-service-type` [Scheme-Variable]
: Der Typ des „Boot-Dienstes“, der das *Boot-Skript* erzeugt. Das Boot-Skript ist das, was beim Booten durch die initiale RAM-Disk ausgeführt wird.

`etc-service-type` [Scheme-Variable]
: Der Typ des `/etc`-Dienstes. Dieser Dienst wird benutzt, um im `/etc`-Verzeichnis Dateien zu platzieren. Er kann erweitert werden, indem man Name-Datei-Tupel an ihn übergibt wie in diesem Beispiel:

```
(list `("issue" ,(plain-file "issue" "Willkommen!\n")))
```

  Dieses Beispiel würde bewirken, dass eine Datei `/etc/issue` auf die angegebene Datei verweist.

`setuid-program-service-type` [Scheme-Variable]
Der Typ des Dienstes für setuid-Programme, der eine Liste von ausführbaren Dateien ansammelt, die jeweils als G-Ausdrücke übergeben werden und dann zur Menge der setuid- oder setgid-gesetzten Programme auf dem System hinzugefügt werden (siehe Abschnitt 12.10 [Setuid-Programme], Seite 604).

`profile-service-type` [Scheme-Variable]
Der Typ des Dienstes zum Einfügen von Dateien ins *Systemprofil* – d.h. die Programme unter `/run/current-system/profile`. Andere Dienste können ihn erweitern, indem sie ihm Listen von ins Systemprofil zu installierenden Paketen übergeben.

`provenance-service-type` [Scheme-Variable]
Dies ist der Diensttyp des Dienstes, um *Provenienz-Metadaten* zusammen mit dem eigentlichen System zu speichern. Dazu werden mehrere Dateien unter `/run/current-system` erstellt:

`channels.scm`
Sie ist eine „Kanaldatei“, wie sie an `guix pull -C` oder `guix time-machine -C` übergeben werden kann, die die zum Erstellen des Systems notwendigen Kanäle beschreibt, sofern diese Information zur Verfügung gestanden hat (siehe Kapitel 7 [Kanäle], Seite 77).

`configuration.scm`
Jene Datei entspricht derjenigen, die als Wert für diesen `provenance-service-type`-Dienst mitgegeben wurde. Nach Vorgabe übergibt `guix system reconfigure` automatisch die Betriebssystemkonfigurationsdatei, die es auf der Befehlszeile bekommen hat.

`provenance`
Hierin sind dieselben Informationen enthalten, die auch in den anderen beiden Dateien stehen, aber in einem leichter zu verarbeitenden Format.

Im Allgemeinen genügen diese zwei Informationen (Kanäle und Konfigurationsdatei), um das Betriebssystem „aus seinem Quellcode heraus“ zu reproduzieren.

Einschränkungen: Sie benötigen diese Informationen, um Ihr Betriebssystem erneut zu erstellen, aber sie alleine *reichen nicht immer aus*. Insbesondere ist `configuration.scm` alleine nicht hinreichend, wenn es *nicht* eigenständig ist, sondern auf externe Guile-Module oder andere Dateien verweist. Wenn Sie erreichen wollen, dass `configuration.scm` eigenständig wird, empfehlen wir, alle darin verwendeten Module oder Dateien zu Bestandteilen eines Kanals zu machen.

Übrigens sind Provenienzmetadaten „still“ in dem Sinn, dass ihr Vorhandensein nichts an den Bits ändert, die Ihr System ausmachen, *abgesehen von den die Metadaten ausmachenden Bits*. Zwei verschiedene Betriebssystemkonfigurationen und Kanalangaben können also Bit für Bit dasselbe System erzeugen, aber wenn der `provenance-service-type` benutzt wird, enthalten die beiden Systeme trotzdem unterschiedliche Metadaten und damit nicht mehr den gleichen Dateinamen im Store, was es schwerer macht, ihre Gleichheit zu erkennen.

Dieser Dienst wird automatisch zu Ihrer Betriebssystemkonfiguration hinzugefügt, wenn Sie `guix system reconfigure`, `guix system init` oder `guix deploy` benutzen.

`linux-loadable-module-service-type` [Scheme-Variable]
Der Diensttyp des Dienstes, der Listen von Paketen mit Kernel-Modulen, die der Kernel laden können soll, sammelt und dafür sorgt, dass der Kernel sie laden kann.

Der Diensttyp ist dazu gedacht, von anderen Diensttypen erweitert zu werden, etwa so:

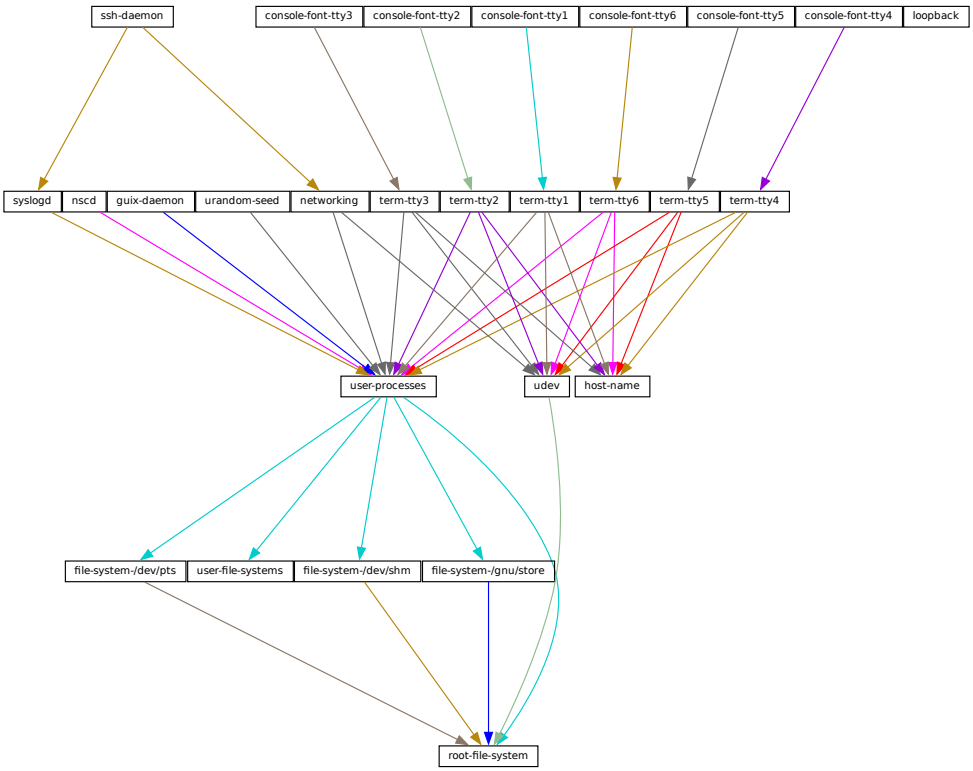
```
(simple-service 'installing-module
                linux-loadable-module-service-type
                (list module-to-install-1
                      module-to-install-2))
```

Die Module werden *nicht* geladen. Sie werden nur zum Kernel-Profil hinzugefügt, damit sie mit anderen Werkzeugen überhaupt erst geladen werden *können*.

12.18.4 Shepherd-Dienste

Das Modul `(gnu services shepherd)` gibt eine Methode an, mit der Dienste definiert werden können, die von GNU Shepherd verwaltet werden, was das Initialisierungssystem (das „init“-System) ist – es ist der erste Prozess, der gestartet wird, wenn das System gebootet wird, auch bekannt als PID 1 (siehe Abschnitt “Introduction” in *The GNU Shepherd Manual*).

Dienste unter dem Shepherd können voneinander abhängen. Zum Beispiel kann es sein, dass der SSH-Daemon erst gestartet werden darf, nachdem der Syslog-Daemon gestartet wurde, welcher wiederum erst gestartet werden kann, sobald alle Dateisysteme eingebunden wurden. Das einfache Betriebssystem, dessen Definition wir zuvor gesehen haben (siehe Abschnitt 12.1 [Das Konfigurationssystem nutzen], Seite 250), ergibt folgenden Dienstgraphen:



Sie können so einen Graphen tatsächlich für jedes Betriebssystem erzeugen lassen, indem Sie den Befehl `guix system shepherd-graph` benutzen (siehe [system-shepherd-graph], Seite 628).

Der `%shepherd-root-service` ist ein Dienstobjekt, das diesen Prozess mit PID 1 repräsentiert. Der Dienst hat den Typ `shepherd-root-service-type`. Sie können ihn erweitern, indem Sie eine Liste von `<shepherd-service>`-Objekten an ihn übergeben.

`shepherd-service` [Datentyp]

Der Datentyp, der einen von Shepherd verwalteten Dienst repräsentiert.

`provision`

Diese Liste von Symbolen gibt an, was vom Dienst angeboten wird.

Das bedeutet, es sind die Namen, die an `herd start`, `herd status` und ähnliche Befehle übergeben werden können (siehe Abschnitt “Invoking herd” in *The GNU Shepherd Manual*). Siehe Abschnitt “Slots of services” in *The GNU Shepherd Manual* für Details.

`requirement` (Vorgabe: `'()`)

Eine Liste von Symbolen, die angeben, von welchen anderen Shepherd-Diensten dieser hier abhängt.

`one-shot?` (Vorgabe: `#f`)
: Gibt an, ob dieser Dienst nur einmal ausgeführt wird („one-shot“). Einmalig ausgeführte Dienste werden gestoppt, sobald ihre `start`-Aktion abgeschlossen wurde. Siehe Abschnitt “Slots of services” in *The GNU Shepherd Manual* für weitere Informationen.

`respawn?` (Vorgabe: `#t`)
: Ob der Dienst neu gestartet werden soll, nachdem er gestoppt wurde, zum Beispiel wenn der ihm zu Grunde liegende Prozess terminiert wird.

`start`
`stop` (Vorgabe: `#~(const #f)`)
: Die Felder `start` und `stop` beziehen sich auf Shepherds Funktionen zum Starten und Stoppen von Prozessen (siehe Abschnitt “Service De- and Constructors” in *The GNU Shepherd Manual*). Sie enthalten G-Ausdrücke, die in eine Shepherd-Konfigurationdatei umgeschrieben werden (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

`actions` (Vorgabe: `'()`)
: Dies ist eine Liste von `shepherd-action`-Objekten (siehe unten), die vom Dienst zusätzlich unterstützte *Aktionen* neben den Standardaktionen `start` und `stop` angeben. Hier aufgeführte Aktionen werden als `herd`-Unterbefehle verfügbar gemacht:

  ```
  herd Aktion Dienst [Argumente...]
  ```

`auto-start?` (Vorgabe: `#t`)
: Ob dieser Dienst automatisch durch Shepherd gestartet werden soll. Wenn es auf `#f` steht, muss der Dienst manuell über `herd start` gestartet werden.

`documentation`
: Eine Zeichenkette zur Dokumentation, die angezeigt wird, wenn man dies ausführt:

  ```
  herd doc Dienstname
  ```

  wobei der *Dienstname* eines der Symbole aus der `provision`-Liste sein muss (siehe Abschnitt “Invoking herd” in *The GNU Shepherd Manual*).

`modules` (Vorgabe: `%default-modules`)
: Dies ist die Liste der Module, die in den Sichtbarkeitsbereich geladen sein müssen, wenn `start` und `stop` ausgewertet werden.

Im folgenden Beispiel wird ein Shepherd-Dienst definiert, der `syslogd`, den Systemprotokollier-Daemon aus den GNU Networking Utilities, startet (siehe Abschnitt “syslogd invocation” in *GNU Inetutils*):

```
(let ((config (plain-file "syslogd.conf" "...")))
  (shepherd-service
    (documentation "Den Syslog-Daemon (syslogd) ausführen.")
    (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))))
```

Die Kernelemente in diesem Beispiel sind die Felder `start` und `stop`: Es sind Code-Schnipsel, die erst später ausgewertet werden; wir sagen, sie sind *staged*. Sie benutzen die von Shepherd bereitgestellte Prozedur `make-forkexec-constructor` und ihr duales Gegenstück, `make-kill-destructor` (siehe das Abschnitt "Service De- and Constructors" in *Handbuch von GNU Shepherd*). Durch das `start`-Feld wird `shepherd` das `syslogd`-Programm mit den angegebenen Befehlszeilenoptionen starten; beachten Sie, dass wir `config` nach `--rcfile` angeben; dabei handelt es sich um eine vorher im Code deklarierte Konfigurationsdatei (deren Inhalt wir hier nicht erklären). Entsprechend wird mit dem Feld `stop` ausgesagt, wie dieser Dienst gestoppt werden kann; in diesem Fall wird er über den Systemaufruf `kill` gestoppt, dem die PID des Prozesses übergeben wird. Code-Staging wird über G-Ausdrücke umgesetzt: Mit `#~` beginnt der später ausgeführte „staged Code“, und `#$` beendet dies und der Code darin wird wirtsseitig hier und jetzt ausgewertet (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175).

`shepherd-action` [Datentyp]

Dieser Datentyp definiert zusätzliche Aktionen, die ein Shepherd-Dienst implementiert (siehe oben).

`name` Die Aktion bezeichnendes Symbol.

`documentation`
: Diese Zeichenkette ist die Dokumentation für die Aktion. Sie können sie sehen, wenn Sie dies ausführen:

```
herd doc Dienst action Aktion
```

`procedure`
: Dies sollte ein G-Ausdruck sein, der zu einer mindestens ein Argument nehmenden Prozedur ausgewertet wird. Das Argument ist der „running“-Wert des Dienstes (siehe Abschnitt "Slots of services" in *The GNU Shepherd Manual*).

Das folgende Beispiel definiert eine Aktion namens `sag-hallo`, die den Benutzer freundlich begrüßt:

```
(shepherd-action
  (name 'sag-hallo)
  (documentation "Sag Hallo!")
  (procedure #~(lambda (running . args)
                 (format #t "Hallo, Freund! Argumente: ~s\n"
                         args)
                 #t)))
```

Wenn wir annehmen, dass wir die Aktion zum Dienst `beispiel` hinzufügen, können Sie Folgendes ausführen:

```
# herd sag-hallo beispiel
Hallo, Freund! Argumente: ()
```

```
# herd sag-hallo beispiel a b c
Hallo, Freund! Argumente: ("a" "b" "c")
```

Wie Sie sehen können, ist das eine sehr ausgeklügelte Art, Hallo zu sagen. Siehe Abschnitt "Service Convenience" in *The GNU Shepherd Manual* für mehr Informationen zu Aktionen.

`shepherd-configuration-action` [Scheme-Prozedur]

Liefert eine Aktion `configuration`, mit der *Datei* angezeigt wird. Dafür sollte der Name der Konfigurationsdatei des Dienstes übergeben werden.

Dienste mit dieser Aktion auszustatten, kann hilfreich sein. Zum Beispiel ist der Tor-Dienst für anonyme Netzwerkrouten (siehe Abschnitt 12.9.5 [Netzwerkdienste], Seite 314) ungefähr so definiert:

```
(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.")))
```

Über diese Aktion haben Administratoren die Möglichkeit, zu prüfen, wie die an `tor` übergebene Konfigurationsdatei aussieht, mit einem Shell-Befehl wie:

```
cat $(herd configuration tor)
```

So gelingt die Fehlersuche!

`shepherd-root-service-type` [Scheme-Variable]

Der Diensttyp für den Shepherd-„Wurzeldienst" – also für PID 1.

Dieser Diensttyp stellt das Ziel für Diensterweiterungen dar, die Shepherd-Dienste erzeugen sollen (siehe Abschnitt 12.18.2 [Diensttypen und Dienste], Seite 637, für ein Beispiel). Jede Erweiterung muss eine Liste von `<shepherd-service>`-Objekten übergeben. Sein Wert muss eine `shepherd-configuration` sein, wie im Folgenden beschrieben.

`shepherd-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration von Shepherd.

`shepherd (Vorgabe: shepherd)`
: Das zu benutzende Shepherd-Paket.

`services (Vorgabe: '())`
: Eine Liste zu startender Shepherd-Dienste als `<shepherd-service>`-Objekte. Wahrscheinlich sollten Sie stattdessen den Mechanismus zur Diensterweiterung benutzen (siehe Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644).

Im folgenden Beispiel wird ein anderes Shepherd-Paket für das Betriebssystem festgelegt:

```
(operating-system
  ;; ...
  (services (append (list openssh-service-type))
            ;; ...
            %desktop-services)
  ;; ...
  ;; Eigenes Shepherd-Paket benutzen.
  (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` [Scheme-Variable]

Dieser Dienst repräsentiert PID 1.

12.18.5 Komplizierte Konfigurationen

Einige Programme haben vielleicht ziemlich komplizierte Konfigurationsdateien oder -formate. Sie können die Hilfsmittel, die in dem Modul `(gnu services configuration)` definiert sind, benutzen, um das Erstellen von Scheme-Anbindungen für diese Konfigurationsdateien leichter zu machen.

Das Werkzeug der Wahl ist das Makro `define-configuration`, mit dem Sie einen Scheme-Verbundstyp definieren (siehe Abschnitt "Record Overview" in *Referenzhandbuch zu GNU Guile*). Ein Scheme-Verbund dieses Typs wird zu einer Konfigurationsdatei serialisiert, indem *Serialisierer* aufgerufen werden. Das sind Prozeduren, die einen Scheme-Wert nehmen und einen G-Ausdruck zurückliefern (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), der wiederum schließlich, nachdem er auf die Platte serialisiert wurde, eine Zeichenkette liefern sollte. Details folgen.

`define-configuration` *`Name Klausel1 Klausel2 ...`* *Einen Verbundstyp mit dem Namen* `Name` [Scheme-Syntax]

erstellen, der die in den Klauseln gefundenen Felder enthalten wird.

Eine Klausel kann eine der folgenden Formen annehmen:

```
(Feldname
 (Typ Vorgabewert)
 Dokumentation)

(Feldname
 (Typ Vorgabewert)
 Dokumentation
 Serialisierer)

(Feldname
 (Typ)
```

```
 Dokumentation)

(Feldname
 (Typ)
 Dokumentation
 Serialisierer)
```

Feldname ist ein Bezeichner, der als Name des Feldes im erzeugten Verbund verwendet werden wird.

Mit *Typ* wird benannt, was als Typ des Werts für *Feldname* gelten soll. Weil Guile typenlos ist, muss es eine entsprechend benannte Prädikatprozedur `Typ?` geben, die auf dem Wert für das Feld aufgerufen werden wird und prüft, dass der Wert des Feldes den geltenden Typ hat. Wenn zum Beispiel `package` als *Typ* angegeben wird, wird eine Prozedur namens `package?` auf den für das Feld angegebenen Wert angewandt werden. Sie muss zurückliefern, ob es sich wirklich um ein `<package>`-Objekt handelt.

Vorgabewert ist der Standardwert für das Feld; wenn keiner festgelegt wird, muss der Benutzer einen Wert angeben, wenn er ein Objekt mit dem Verbundstyp erzeugt.

Dokumentation ist eine Zeichenkette, die mit Texinfo-Syntax formatiert ist. Sie sollte eine Beschreibung des Feldes enthalten.

Serialisierer ist der Name einer Prozedur, die zwei Argumente nimmt, erstens den Namen des Feldes und zweitens den Wert für das Feld, und eine Zeichenkette oder einen G-Ausdruck (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) zurückliefern sollte, welcher Inhalt dafür in die Konfigurationsdatei serialisiert wird. Wenn kein Serialisierer angegeben wird, wird eine Prozedur namens `serialize-Typ` dafür aufgerufen.

Eine einfache Serialisiererprozedur könnte so aussehen:

```
(define (serialize-boolean field-name value)
  (let ((value (if value "true" "false")))
    #~(string-append #$field-name #$value)))
```

Es kommt vor, dass in derselben Datei mehrere Arten von Konfigurationsverbund definiert werden, deren Serialisierer sich für denselben Typ unterscheiden, weil ihre Konfigurationen verschieden formatiert werden müssen. Zum Beispiel braucht es eine andere `serialize-boolean`-Prozedur für einen Getmail-Dienst als für einen Transmission-Dienst. Um leichter mit so einer Situation fertig zu werden, können Sie ein Serialisierer-Präfix nach dem Literal `prefix` in der `define-configuration`-Form angeben. Dann müssen Sie den eigenen *Serialisierer nicht* für jedes Feld angeben.

```
(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-))
```

In manchen Fällen wollen Sie vielleicht überhaupt gar keinen Wert aus dem Verbund serialisieren. Dazu geben Sie das Literal `no-serialization` an. Auch können Sie das Makro `define-configuration/no-serialization` benutzen, was eine Kurzschreibweise dafür ist.

```
;; Nichts wird auf die Platte serialisiert.
(define-configuration foo-configuration
  (field
   (string "test")
   "Some documentation.")
  (no-serialization))

;; Das Gleiche wie oben.
(define-configuration/no-serialization bar-configuration
  (field
   (string "test")
   "Some documentation."))
```

`define-maybe` *Typ* [Scheme-Syntax]

Manchmal soll ein Feld dann *nicht* serialisiert werden, wenn der Benutzer keinen Wert dafür angibt. Um das zu erreichen, können Sie das Makro `define-maybe` benutzen, um einen „maybe type“, zu Deutsch „Vielleicht-Typ“, zu definieren. Wenn der Wert eines Vielleicht-Typs *nicht* gesetzt oder auf den Wert `%unset-value` gesetzt ist, wird er nicht serialisiert.

Beim Definieren eines „Vielleicht-Typs“ ist die Voreinstellung, den dem Grundtyp entsprechenden Serialisierer zu benutzen. Zum Beispiel wird ein Feld vom Typ `maybe-string` nach Voreinstellung mit der Prozedur `serialize-string` serialisiert. Natürlich können Sie stattdessen eine eigene Serialisiererprozedur festlegen. Ebenso muss der Wert entweder den Typ „string“ (also Zeichenkette) aufweisen oder unspezifiziert sein.

```
(define-maybe string)

(define (serialize-string field-name value)
  ...)

(define-configuration baz-configuration
  (name
   ;; Wenn eine Zeichenkette angegeben wird, wird diese mit der
   ;; Prozedur ,,serialize-string‘‘ serialisiert werden.  Sonst
   ;; ist vorgegeben, für dieses Feld nichts zu serialisieren.
   maybe-string
```

```
    "The name of this module."))
```

Wie bei `define-configuration` kann man ein Präfix für Serialisierernamen als Literal mit `prefix` angeben.

```
(define-maybe integer
  (prefix baz-))

(define (baz-serialize-integer field-name value)
  ...)
```

Auch gibt es das Literal `no-serialization`. Wenn es angegeben wird, bedeutet das, es wird kein Serialisierer für den Vielleicht-Typ eingesetzt, ganz gleich ob ein Wert gesetzt wurde oder nicht. `define-maybe/no-serialization` ist eine Kurzschreibweise, um das Literal `no-serialization` festzulegen.

```
(define-maybe/no-serialization symbol)

(define-configuration/no-serialization test-configuration
  (mode
   maybe-symbol
   "Docstring."))
```

`maybe-value-set?` *`Wert`* [(Scheme-Prozedur)]
Mit diesem Prädikat können Sie ermitteln, ob ein Benutzer für das Vielleicht-Feld einen bestimmten Wert angegeben hat.

`serialize-configuration` *`Konfiguration Felder`* *Liefert einen G-Ausdruck mit den Werten zu jedem der* [Scheme-Prozedur]
Felder im Verbundsobjekt *Konfiguration*, das mit `define-configuration` erzeugt wurde. Der G-Ausdruck kann mit z.B. `mixed-text-file` auf die Platte serialisiert werden.

`empty-serializer` *`Feldname Wert`* [Scheme-Prozedur]
Ein Serialisierer, der nur die leere Zeichenkette zurückliefert. Die Prozedur `serialize-package` ist als anderer Name dafür vordefiniert.

Nachdem Sie einen Konfigurationsverbundstyp definiert haben, haben Sie bestimmt auch die Absicht, die Dokumentation dafür zu schreiben, um anderen Leuten zu erklären, wie man ihn benutzt. Dabei helfen Ihnen die folgenden zwei Prozeduren, die hier dokumentiert sind.

`generate-documentation` *`Dokumentation`* [Scheme-Prozedur]
`Dokumentationsname` *Ein Stück Texinfo aus den Docstrings in der*
Dokumentation erzeugen. Als *Dokumentation* geben Sie eine Liste aus `(`*`Bezeichnung Felder Unterdokumentation ...`*`)` an. *Bezeichnung* ist ein Symbol und gibt den Namen des Konfigurationsverbunds an. *Felder* ist eine Liste aller Felder des Konfigurationsverbunds.

Unterdokumentation ist ein Tupel `(`*`Feldname Konfigurationsname`*`)`. *Feldname* ist der Name des Feldes, das einen anderen Konfigurationsverbund als Wert hat. *Konfigurationsname* ist der Name dessen Konfigurationsverbundstyps.

Eine *Unterdokumentation* müssen Sie nur angeben, wenn es verschachtelte Verbundstypen gibt. Zum Beispiel braucht ein Verbundsobjekt `getmail-configuration` (siehe Abschnitt 12.9.12 [Mail-Dienste], Seite 392) ein Verbundsobjekt `getmail-configuration-file` in seinem `rcfile`-Feld, daher ist die Dokumentation für `getmail-configuration-file` verschachtelt in der von `getmail-configuration`.

```
(generate-documentation
  `((getmail-configuration ,getmail-configuration-fields
     (rcfile getmail-configuration-file))
    ...)
  'getmail-configuration)
```

Für *Dokumentationsname* geben Sie ein Symbol mit dem Namen des Konfigurationsverbundstyps an.

`configuration->documentation` [Scheme-Prozedur]

Konfigurationssymbol Für *Konfigurationssymbol*, ein Symbol mit dem Namen, der beim Definieren des Konfigurationsverbundstyp mittels `define-configuration` benutzt wurde, gibt diese Prozedur die Texinfo-Dokumentation seiner Felder aus. Wenn es keine verschachtelten Konfigurationsfelder gibt, bietet sich diese Prozedur an, mit der Sie ausschließlich die Dokumentation der obersten Ebene von Feldern ausgegeben bekommen.

Gegenwärtig gibt es kein automatisiertes Verfahren, um die Dokumentation zu Konfigurationsverbundstypen zu erzeugen und gleich ins Handbuch einzutragen. Stattdessen würden Sie jedes Mal, wenn Sie etwas an den Docstrings eines Konfigurationsverbundstyps ändern, aufs Neue `generate-documentation` oder `configuration->documentation` von Hand aufrufen und die Ausgabe in die Datei `doc/guix.texi` einfügen.

Nun folgt ein Beispiel, wo ein Verbundstyp mit `define-configuration` usw. erzeugt wird.

```
(use-modules (gnu services)
             (guix gexp)
             (gnu services configuration)
             (srfi srfi-26)
             (srfi srfi-1))

;; Feldnamen, in Form von Scheme-Symbolen, zu Zeichenketten machen
(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."))
```

Eine Kontaktelistekonfiguration könnte dann wie folgt erzeugt werden:

```
(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))))))
```

Wenn Sie diese Konfiguration auf die Platte serialisierten, ergäbe sich so eine Datei:

```
[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
```

13 Persönliche Konfiguration

Die Umgebung in Ihrem Persönlichen Verzeichnis können Sie mit Guix einrichten, über eine deklarative Konfiguration dieser *Persönlichen Umgebung* in einem `home-environment`-Objekt. Wir verwenden die Konfigurationsmechanismen, die im vorangehenden Kapitel beschrieben wurden (siehe Abschnitt 12.18 [Dienste definieren], Seite 635), wenden diese aber auf die Konfigurationsdateien („Dotfiles“) und Pakete des Benutzers an. Das geht sowohl auf Guix System als auch auf Fremddistributionen. Jeder Benutzer kann so alle Pakete und Dienste, die für ihn installiert und eingerichtet sein sollen, deklarieren. Sobald man eine Datei mit einem `home-environment`-Verbundsobjekt hat, kann man diese Konfiguration *instanziieren* lassen, ohne besondere Berechtigungen auf dem System zu haben. Dazu rufen Sie den Befehl `guix home` auf (siehe Abschnitt 13.4 [Aufruf von guix home], Seite 674).

> **Anmerkung:** Die in diesem Abschnitt beschriebenen Funktionalitäten befinden sich noch in der Entwicklung und können sich ändern. Kontaktieren Sie uns auf `guix-devel@gnu.org`!

Die Persönliche Umgebung eines Benutzers setzt sich für gewöhnlich aus drei Grundbestandteilen zusammen: Software, Konfiguration und Zustand. In den gängigsten Distributionen wird Software normalerweise systemweit für alle Nutzer installiert, dagegen können die meisten Software-Pakete bei GNU Guix durch jeden Benutzer selbst installiert werden, ohne Administratorrechte vorauszusetzen. Seine Software ist daher nur ein weiterer Teil der Persönlichen Umgebung des Benutzers. Aber Pakete alleine machen nicht glücklich, oft müssen sie erst noch konfiguriert werden, in der Regel mit Konfigurationsdateien in `XDG_CONFIG_HOME` (nach Voreinstellung bedeutet das in `~/.config`) oder anderen Verzeichnissen. Übrig sind verschiedene Arten von Zustand, etwa Mediendateien, Anwendungsdatenbanken und Protokolldateien.

Persönliche Umgebungen mit Guix zu verwalten bringt einige Vorteile:

- Alle Software kann in derselben Sprache konfiguriert werden (Guile Scheme), wodurch Nutzer in die Lage versetzt werden, dieselben Werte in die Konfigurationen verschiedener Programme einzusetzen.
- Eine wohldefinierte Persönliche Umgebung ist eigenständig und kann deklarativ und reproduzierbar verfasst werden. Binärdateien von anderswo herunterzuladen oder eine Konfigurationsdatei von Hand zu bearbeiten ist unnötig.
- Nach jedem Aufruf von `guix home reconfigure` wird eine neue Generation der Persönlichen Umgebung erzeugt. Dadurch können Benutzer zu einer vorherigen Generation der Persönlichen Umgebung zurückwechseln, d.h. sie müssen keine Sorgen haben, dass nach einer Änderung ihre Konfiguration nicht mehr funktioniert.
- Sie können sogar zustandsbehaftete Daten über Guix Home verwalten. Dazu gehört etwa, Git-Repositorys automatisch zu klonen, wenn Sie die Maschine zum ersten Mal einrichten, oder auch Befehle wie `rsync` regelmäßig zu starten, um die Daten mit einem anderen Rechner abzugleichen. Diese Funktionalität befindet sich jedoch noch auf einer experimentellen Entwicklungsstufe.

13.1 Deklaration der Persönlichen Umgebung

Die Persönliche Umgebung können Sie konfigurieren, indem Sie eine `home-environment`-Deklaration in einer Datei speichern, die Sie dem Befehl `guix home` übergeben (siehe Ab-

schnitt 13.4 [Aufruf von guix home], Seite 674). Der einfachste Einstieg gelingt, indem Sie mit `guix home import` eine anfängliche Konfiguration erzeugen lassen:

```
guix home import ~/src/guix-config
```

Durch den Befehl `guix home import` werden einige dieser „Dotfiles“ wie `~/.bashrc` aus Ihrem Persönlichen Verzeichnis gelesen und in das angegebene Verzeichnis kopiert. In diesem Beispiel ist das `~/src/guix-config`. Auch der Inhalt Ihres Profils in `~/.guix-profile` wird gelesen und anhand davon wird eine Konfiguration Ihrer Persönlichen Umgebung in `~/src/guix-config/home-configuration.scm` erzeugt, die Ihrer momentanen Konfiguration nachempfunden ist.

Zu einer einfachen Konfiguration kann z.B. Bash gehören zusammen mit einer eigenen Textdatei wie im folgenden Beispiel. Zögern Sie nicht damit, in der Persönlichen Umgebung etwas zu deklarieren, was mit Ihren bestehenden Dotfiles überlappt, denn bevor irgendeine Konfigurationsdatei installiert wird, legt Guix Home eine Sicherungskopie an einer anderen Stelle im Persönlichen Verzeichnis ab.

> **Anmerkung:** Wir empfehlen sehr, Ihre Shell oder Shells mit Guix Home zu verwalten, weil dadurch sicherlich alle vorausgesetzten Skripte über `source`-Befehle in die Shell-Konfiguration eingebunden werden. Andernfalls müssten Sie sie von Hand einbinden (siehe Abschnitt 13.2 [Shell-Konfiguration], Seite 658).

```
(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")))))))
```

Was das `packages`-Feld tut, sollte klar sein: Dadurch werden die aufgelisteten Pakete in das Profil des Benutzers installiert. Das wichtigste Feld ist `services`, was eine Liste

Persönlicher Dienste zu enthalten hat. Sie stellen die Grundbausteine einer Persönlichen Umgebung dar.

Für Persönliche Dienste gibt es keinen Daemon (zumindest nicht zwingend). Ein Persönlicher Dienst stellt nur ein Element dar, um einen Aspekt einer Persönlichen Umgebung zu deklarieren bzw. andere Aspekte zu erweitern. Dieser im vorherigen Kapitel erörterte Erweiterungsmechanismus (siehe Abschnitt 12.18 [Dienste definieren], Seite 635) sollte nicht verwechselt werden mit Shepherd-Diensten (siehe Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644). Mit dem Erweiterungsmechanismus und etwas Scheme-Code, um die Dinge zusammenzubringen, haben Sie als Nutzer die Freiheit, eigens auf Sie abgestimmte Persönliche Umgebungen zu entwerfen.

Wenn die Konfiguration brauchbar ausschaut, können Sie sie einem ersten Test in einem Wegwerf-„Container“ unterziehen:

```
guix home container config.scm
```

Dieser Befehl öffnet eine Shell, in der Ihre Persönliche Umgebung läuft. Diese Shell befindet sich in einem Container, ist also vom übrigen System isoliert. Deshalb können Sie Ihre Konfiguration so ausprobieren – Sie sehen, ob noch Teile der Konfiguration fehlen oder falsches Verhalten bewirken, ob Daemons gestartet werden und so weiter. Sobald Sie die Shell verlassen, haben Sie wieder die „wirkliche“ Eingabeaufforderung Ihrer ursprünglichen Shell vor sich.

Wenn Ihre Konfigurationsdatei so weit fertig ist und Ihrem Bedarf entspricht, wird es Zeit, Ihre Persönliche Umgebung zu rekonfigurieren, indem Sie diesen Befehl geben:

```
guix home reconfigure config.scm
```

Dadurch wird Ihre Persönliche Umgebung *erstellt* und von `~/.guix-home` darauf verwiesen. Geschafft!

> **Anmerkung:** Achten Sie darauf, dass Ihr Betriebssystem mit elogind, systemd oder einem entsprechenden Mechanismus das XDG-Laufzeitverzeichnis anlegt und die Variable `XDG_RUNTIME_DIR` festlegt. Wenn nicht, kann das Skript `on-first-login` nichts starten und Prozesse wie der benutzereigene Shepherd und alles, was davon abhängt, fahren nicht hoch.

13.2 Shell-Konfiguration

Sie können den folgenden Abschnitt problemlos überspringen, wenn Ihre Shell oder Shells mit Guix Home verwaltet werden. Wenn nicht, lesen Sie ihn bitte genau.

Es gibt ein paar Skripte, die eine Login-Shell auswerten muss, damit die Persönliche Umgebung aktiv wird. Die Dateien, die nur beim Starten von Login-Shells geladen werden, tragen am Ende das Suffix `profile`. Mehr Informationen über Login-Shells finden Sie in dem Referenzhandbuch von GNU Bash Abschnitt “Invoking Bash” in *hier* und Abschnitt “Bash Startup Files” in *hier*.

Als Erstes wird `setup-environment` gesourcet, wodurch alle nötigen Umgebungsvariablen festgelegt werden (einschließlich vom Benutzer festgelegter Variabler) und als Zweites wird mit `on-first-login` Shepherd für den angemeldeten Benutzer gestartet und die über andere Persönliche Dienste deklarierten Aktionen durchgeführt, wenn die Dienste `home-run-on-first-login-service-type` erweitern.

Guix Home wird immer `~/.profile` mit den folgenden Zeilen darin anlegen:

```
HOME_ENVIRONMENT=$HOME/.guix-home
. $HOME_ENVIRONMENT/setup-environment
$HOME_ENVIRONMENT/on-first-login
```

Dadurch werden POSIX-konforme Login-Shells die Persönliche Umgebung aktivieren. Jedoch wird diese Datei in der Regel von den meisten modernen Shells *nicht* gelesen, weil sie nach Voreinstellung *nicht* im POSIX-Modus laufen und stattdessen ihre eigenen `*profile`-Dateien beim Start laden. Zum Beispiel wird Bash der Datei `~/.bash_profile` den Vorzug geben, wenn diese existiert, und nur falls sie *nicht* existiert, liest Bash die Datei `~/.profile`. Zsh (wenn ihr keine zusätzlichen Argumente übergeben wurden) ignoriert `~/.profile` grundsätzlich, auch wenn `~/.zprofile` fehlt.

Damit Ihre Shell `~/.profile` doch beachtet, tragen Sie `. ~/.profile` oder `source ~/.profile` in die beim Start gelesene Datei der Login-Shell ein. Im Fall von Bash ist dies `~/.bash_profile` und für Zsh ist es `~/.zprofile`.

> **Anmerkung:** Dieser Schritt fällt weg, wenn Sie Guix Home Ihre Shell verwalten lassen. Dann nämlich liefe all das vollautomatisch ab.

13.3 Persönliche Dienste

Als *Persönlichen Dienst* (englisch „Home Service“) bezeichnen wir nicht nur Software, die über einen durch Shepherd verwalteten Daemon gestartet wird (siehe Abschnitt “Jump Start” in *Handbuch von GNU Shepherd*), denn darum geht es meistens gar nicht. Ein Persönlicher Dienst bedeutet lediglich ein Baustein der Persönlichen Umgebung, mit dem so etwas deklariert wird wie eine Auswahl in die Persönliche Umgebung zu installierender Pakete, eine Reihe von Konfigurationsdateien, für die symbolische Verknüpfungen in `XDG_CONFIG_HOME` (nach Voreinstellung ist `~/.config` gemeint) angelegt werden oder für Login-Shells festgelegte Umgebungsvariable.

Ein Diensterweiterungsmechanismus (siehe Abschnitt 12.18.1 [Dienstkompositionen], Seite 635) ermöglicht es Persönlichen Diensten, andere Persönliche Dienste zu erweitern und so auf die von ihnen bereits angebotenen Fähigkeiten zurückzugreifen. Beispielsweise kann für einen Dienst ein mcron-Auftrag (siehe *GNU Mcron*) deklariert werden, indem man ihn den Persönlichen Mcron-Dienst erweitern lässt (siehe Abschnitt 13.3.3 [Persönlicher Mcron-Dienst], Seite 666); ein Daemon wird deklariert, indem man ihn den Persönlichen Shepherd-Dienst erweitern lässt (siehe Abschnitt 13.3.5 [Persönlicher Shepherd-Dienst], Seite 668); für Bash werden neue Befehle festgelegt, indem man den Persönlichen Shell-Dienst für Bash erweitern lässt (siehe Abschnitt 13.3.2 [Persönliche Shell-Dienste], Seite 663.

Über die verfügbaren Persönlichen Dienste können Sie sich mit dem Befehl `guix home search` informieren (siehe Abschnitt 13.4 [Aufruf von guix home], Seite 674). Wenn Sie die gewünschten Persönlichen Dienste gefunden haben, fügen Sie das entsprechende Modul ein, indem Sie die `use-modules`-Form für das Modul angeben (siehe Abschnitt “Using Guile Modules” in *Referenzhandbuch von GNU Guile*) oder alternativ das Modul in der `#:use-modules`-Direktive Ihres eigenen Moduls nennen (siehe Abschnitt “Creating Guile Modules” in *Referenzhandbuch zu GNU Guile*). Dann deklarieren Sie einen Persönlichen Dienst über die Funktion `service` oder Sie erweitern einen Dienst über die Prozedur `simple-service` aus `(gnu services)`.

13.3.1 Essenzielle Persönliche Dienste

Ein paar grundlegende Persönliche Dienste sind auch in `(gnu services)` definiert; sie sind in erster Linie zur internen Nutzung und für die Erstellung der Persönlichen Umgebung gedacht, aber manche sind auch für Endanwender interessant.

`home-environment-variables-service-type` [Scheme-Variable]

Der Dienst mit diesem Diensttyp wird von jeder Persönlichen Umgebung automatisch instanziiert; so sind die Vorgabeeinstellungen. Sie müssen ihn also nicht definieren, aber vielleicht möchten Sie Ihn erweitern mit einer Liste von Paaren zum Festlegen von Umgebungsvariablen (englisch „Environment Variables“).

```
(list ("ENV_VAR1" . "Wert1")
      ("ENV_VAR2" . "Wert2"))
```

Es ist am einfachsten, ihn zu erweitern, ohne einen neuen Dienst zu definieren, nämlich indem Sie das `simple-service`-Helferlein aus dem Modul `(gnu services)` einsetzen.

```
(simple-service 'einige-hilfreiche-umgebungsvariable-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)))
```

Wenn Sie einen solchen Dienst in die Definition Ihrer Persönlichen Umgebung aufnehmen, fügt das folgenden Inhalt in Ihr `setup-environment`-Skript ein (das von Ihrer Login-Shell gesourcet werden dürfte):

```
export LESSHISTFILE=$XDG_CACHE_HOME/.lesshst
export SHELL=/gnu/store/2hsg15n644f0glrcbkb1kqknmmqdar03-zsh-5.8/bin/zsh
export _JAVA_AWT_WM_NONREPARENTING
```

Anmerkung: Achten Sie darauf, dass das Modul `(gnu packages shells)` mit `use-modules` oder Ähnlichem importiert wird, denn durch den `(gnu packages shells)`-Namensraum wird die Definition des `zsh`-Pakets verfügbar gemacht, auf die obiges Beispiel Bezug nimmt.

Wir verwenden hier eine assoziative Liste (siehe Abschnitt “Association Lists” in *Referenzhandbuch von GNU Guile*). Das ist eine Datenstruktur aus Schlüssel-Wert-Paaren und für `home-environment-variables-service-type` ist der Schlüssel immer eine Zeichenkette und der Wert entweder eine Zeichenkette, ein G-Ausdruck für Zeichenketten (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), ein dateiartiges Objekt (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175) oder ein Boolescher Ausdruck. Im Fall von G-Ausdrücken wird die Variable auf den Wert des G-Ausdrucks festgelegt, bei dateiartigen Objekten auf den Pfad der Datei im Store (siehe Abschnitt 9.9 [Der Store], Seite 165), bei `#t` wird die Variable ohne Wert exportiert und bei `#f` wird sie ganz weggelassen.

`home-profile-service-type` [Scheme-Variable]

Der Dienst dieses Typs wird durch jede Persönliche Umgebung automatisch instanziiert. Sie müssen ihn nicht erst definieren. Vielleicht wollen Sie ihn aber um eine Liste von Paketen erweitern, wenn Sie weitere Pakete in Ihr Profil installieren wollen. Wenn

andere Dienste Programme für den Nutzer verfügbar machen müssen, erweitern diese dazu auch diesen Diensttyp.

Als Wert der Erweiterung muss eine Liste von Paketen angegeben werden:

```
(list htop vim emacs)
```

Hier können Sie den gleichen Ansatz wie bei `simple-service` (siehe Abschnitt 12.18.3 [Service-Referenz], Seite 639) für `home-environment-variables-service-type` fahren. Achten Sie darauf, die Module mit den angegebenen Paketen über etwas wie `use-modules` zu importieren. Um ein Paket oder Informationen über das es enthaltende Modul zu finden, hilft Ihnen `guix search` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Alternativ können Sie `specification->package` einsetzen, um das Paketverbundsobjekt anhand einer Zeichenkette zu spezifizieren; dann brauchen Sie auch das zugehörige Modul *nicht* zu laden.

Es gibt noch mehr essenzielle Dienste, von denen wir aber nicht erwarten, dass Nutzer sie erweitern.

`home-service-type` [Scheme-Variable]

Die Wurzel des gerichteten azyklischen Graphen aus Persönlichen Diensten. Damit wird ein Verzeichnis erzeugt, auf das später die symbolische Verknüpfung `~/.guix-home` verweist und das Konfigurationsdateien, das Profil mit Binärdateien und Bibliotheken sowie ein paar notwendige Skripte enthält, die die Dinge zusammenhalten.

`home-run-on-first-login-service-type` [Scheme-Variable]

Mit dem Dienst dieses Diensttyps wird ein Guile-Skript erzeugt, das von der Login-Shell ausgeführt werden soll. Es wird dabei nur dann ausgeführt, wenn eine besondere Signaldatei in `XDG_RUNTIME_DIR` *nicht* vorgefunden wird, wodurch das Skript nicht unnötig ausgeführt wird, wenn mehrere Login-Shells geöffnet werden.

Man kann den Diensttyp mit einem G-Ausdruck erweitern. Allerdings sollten Nutzer diesen Dienst *nicht* dazu gebrauchen, Anwendungen automatisch zu starten, denn das macht man besser über Erweiterungen des `home-shepherd-service-type` durch einen Shepherd-Dienst (siehe Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644) oder über eine Erweiterung der von der Shell beim Start geladenen Datei um den benötigten Befehl und mittels des für diese Art Shell geeigneten Diensttyps.

`home-files-service-type` [Scheme-Variable]

Mit dem Dienst dieses Typs können Sie eine Liste von Dateien angeben, die in `~/.guix-home/files` platziert werden; normalerweise stehen in diesem Verzeichnis Konfigurationsdateien (genauer gesagt stehen dort symbolische Verknüpfungen auf die eigentlichen Konfigurationsdateien in `/gnu/store`), die dann in `$XDG_CONFIG_DIR` oder in seltenen Fällen nach `$HOME` kopiert werden sollen. Der Dienst kann erweitert werden mit Werten im folgenden Format:

```
`((".sway/config" ,sway-dateiartig)
  (".tmux.conf" ,(local-file "./tmux.conf")))
```

Jede verschachtelte Liste enthält zwei Werte: ein Unterverzeichnis und ein dateiartiges Objekt. Nachdem Sie die Persönliche Umgebung erstellt haben, wird in `~/.guix-home/files` der entsprechende Inhalt eingefügt und alle Unterverzeichnisse werden dazu erzeugt. Allerdings kümmert sich ein anderer Dienst darum, die Dateien dann

weiter zu verteilen. Vorgegeben ist, dass ein `home-symlink-manager-service-type` die notwendigen symbolischen Verknüpfungen im Persönlichen Verzeichnis auf Dateien aus `~/.guix-home/files` anlegt und Sicherungskopien bereits bestehender, im Konflikt stehender Konfigurationsdateien und anderer Dateien anlegt. Dieser symlink-manager gehört zu den essenziellen Persönlichen Diensten (die nach Vorgabeeinstellungen aktiviert sind), aber es ist möglich, dass Sie alternative Dienste benutzen, um fortgeschrittenere Anwendungsfälle wie ein nur lesbares Persönliches Verzeichnis hinzukriegen. Sie sind eingeladen, zu experimentieren und Ihre Ergebnisse zu teilen.

`home-xdg-configuration-files-service-type` [Scheme-Variable]
Dieser Dienst ist `home-files-service-type` sehr ähnlich (und intern erweitert er ihn auch), aber damit werden Dateien definiert, die in `~/.guix-home/files/.config` platziert werden, für welche dann eine symbolische Verknüpfung in `$XDG_CONFIG_DIR` mittels `home-symlink-manager-service-type` (beispielsweise) angelegt wird, wenn die Persönliche Umgebung aktiviert wird. Er kann erweitert werden um Werte im folgenden Format:

```
`(("sway/config" ,sway-dateiartig)
  ;; -> ~/.guix-home/files/.config/sway/config
  ;; -> $XDG_CONFIG_DIR/sway/config (mittels symlink-manager)
  ("tmux/tmux.conf" ,(local-file "./tmux.conf")))
```

`home-activation-service-type` [Scheme-Variable]
Mit dem Dienst dieses Diensttyps wird ein Guile-Skript erzeugt, das bei jedem Aufruf von `guix home reconfigure` und jeder anderen Aktion ausgeführt wird. Damit wird die Persönliche Umgebung aktiviert.

`home-symlink-manager-service-type` [Scheme-Variable]
Mit dem Dienst dieses Diensttyps wird ein Guile-Skript erzeugt, das bei der Aktivierung der Persönlichen Umgebung ausgeführt wird. Um die symbolischen Verknüpfungen zu verwalten, tut es ein paar Dinge, nämlich:

1. Der Inhalt jedes `files/`-Verzeichnisses der aktuellen und der künftigen Persönlichen Umgebung wird eingelesen.
2. Alle symbolischen Verknüpfungen, die bei der vorherigen Aktivierung durch symlink-manager erzeugt wurden, werden gelöst. Wenn dadurch Unterverzeichnisse leer werden, werden sie gelöscht.
3. Neue symbolische Verknüpfungen werden auf folgende Weise erzeugt: Es wird jedes `files/`-Verzeichnis durchsucht (diese werden in der Regel mit `home-files-service-type`, `home-xdg-configuration-files-service-type` und vielleicht anderen Diensten definiert) und für die Dateien aus dem Unterverzeichnis `files/.config/` werden entsprechende Verknüpfungen in `XDG_CONFIG_DIR` abgelegt. Zum Beispiel landet eine symbolische Verknüpfung zu `files/.config/sway/config` in `$XDG_CONFIG_DIR/sway/config`. Die anderen Dateien in `files/` außerhalb des Unterverzeichnisses `files/.config/` werden ein wenig anders behandelt: Die symbolischen Verknüpfungen werden dafür in `$HOME` abgelegt. `files/.ein-programm/config` landet also in `$HOME/.ein-programm/config`.
4. Fehlende Unterverzeichnisse werden erzeugt.

5. Wenn es schon eine Datei mit so einem Namen gibt, wird von der im Konflikt stehenden Datei eine Sicherungskopie behalten.

symlink-manager ist Teil der essenziellen Persönlichen Dienste und ist somit nach Vorgabeeinstellungen aktiviert und wird benutzt.

13.3.2 Shells

Shells spielen eine ziemlich wichtige Rolle bei der Initialisierung der Umgebung. Sie können Ihre Shells selbst einrichten, wie im Abschnitt Abschnitt 13.2 [Shell-Konfiguration], Seite 658, beschrieben, aber empfehlen tun wir, dass Sie die folgend aufgeführten Dienste benutzen. Das ist sowohl leichter als auch zuverlässiger.

Jede Persönliche Umgebung instanziiert `home-shell-profile-service-type`, was eine Datei `~/.profile` erzeugt, die von allen POSIX-kompatiblen Shells geladen wird. In der Datei sind alle Schritte enthalten, die nötig sind, um die Umgebung ordnungsgemäß zu initialisieren. Allerdings bevorzugen viele moderne Shells wie Bash oder Zsh, ihre eigenen Dateien beim Starten zu laden. Darum brauchen wir für diese jeweils einen Persönlichen Dienst (`home-bash-service-type` und `home-zsh-service-type`), um dafür zu sorgen, dass `~/.profile` durch `~/.bash_profile` bzw. `~/.zprofile` gesourcet werden.

Shell-Profil-Dienst

`home-shell-profile-configuration` [Datentyp]

Verfügbare `home-shell-profile-configuration`-Felder sind:

`profile` (Vorgabe: `()`) (Typ: Konfigurationstexte)

Ein `home-shell-profile`-Dienst wird durch die Persönliche Umgebung automatisch instanziiert. Legen Sie den Dienst also *bloß nicht* manuell an; Sie können ihn lediglich erweitern. Für `profile` wird eine Liste dateiartiger Objekte erwartet, die in die Datei `~/.profile` aufgenommen werden. Nach Vorgabe enthält `~/.profile` nur den Code zur Initialisierung, der von Login-Shells ausgeführt werden muss, um die Persönliche Umgebung für den Benutzer verfügbar zu machen. Andere Befehle können hier auch in die Datei eingefügt werden, wenn sie wirklich unbedingt dort stehen müssen. In den meisten Fällen schreibt man sie besser in die Konfigurationsdateien der Shell, wenn man eigene Anpassungen wünscht. Erweitern Sie den `home-shell-profile`-Dienst nur dann, wenn Sie wissen, was Sie tun.

Persönlicher Bash-Dienst

`home-bash-configuration` [Datentyp]

Verfügbare `home-bash-configuration`-Felder sind:

`package` (Vorgabe: `bash`) (Typ: „package“)

Das Bash-Paket, was benutzt werden soll.

`guix-defaults?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)

Ob vernünftige Voreinstellungen an den Anfang der Datei `.bashrc` angefügt werden sollen, wie `/etc/bashrc` zu lesen und die Ausgaben von `ls` einzufärben.

`environment-variables` (Vorgabe: `()`) (Typ: Assoziative-Liste)
: Eine assoziative Liste der Umgebungsvariablen, die für die Bash-Sitzung gesetzt sein sollen. Dieselben Regeln wie für den `home-environment-variables-service-type` gelten auch hier (siehe Abschnitt 13.3.1 [Essenzielle Persönliche Dienste], Seite 660). Der Inhalt dieses Felds wird anschließend an das, was im `bash-profile`-Feld steht, angefügt.

`aliases` (Vorgabe: `()`) (Typ: Assoziative-Liste)
: Assoziative Liste der Alias-Namen, welche für die Bash-Sitzung eingerichtet werden sollen. Die Alias-Namen werden nach dem Inhalt des `bashrc`-Feldes in der Datei `.bashrc` definiert. Jeder Alias-Name steht automatisch in Anführungszeichen, d.h. ein Eintrag wie:

```
'(("ls" . "ls -alF"))
```

wird zu

```
alias ls="ls -alF"
```

`bash-profile` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Eine Liste dateiartiger Objekte, die in `.bash_profile` eingefügt werden. Damit können benutzereigene Befehle beim Start der Login-Shell ausgeführt werden. (In der Regel handelt es sich um die Shell, die gestartet wird, direkt nachdem man sich auf der Konsole (TTY) anmeldet.) `.bash_login` wird niemals gelesen, weil `.bash_profile` immer existiert.

`bashrc` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Eine Liste dateiartiger Objekte, die in `.bashrc` eingefügt werden. Damit können benutzereigene Befehle beim Start einer interaktiven Shell ausgeführt werden. (Interaktive Shells sind Shells für interaktive Nutzung, die man startet, indem man `bash` eintippt oder die durch Terminal-Anwendungen oder andere Programme gestartet werden.)

`bash-logout` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Eine Liste dateiartiger Objekte, die zu `.bash_logout` hinzugefügt werden. Damit können benutzereigene Befehle beim Verlassen der Login-Shell ausgeführt werden. Die Datei wird in manchen Fällen *nicht* gelesen (wenn die Shell zum Beispiel durch exec eines anderen Prozesses terminiert).

Sie können den Bash-Dienst mit Hilfe einer Konfiguration im Verbundsobjekt `home-bash-extension` erweitern. Dessen Felder spiegeln meistens die von `home-bash-configuration` (siehe [home-bash-configuration], Seite 663). Die Inhalte der Erweiterungen werden ans Ende der zugehörigen Bash-Konfigurationsdateien angehängt (siehe Abschnitt "Bash Startup Files" in *Referenzhandbuch von GNU Bash*.

Zum Beispiel können Sie so einen Dienst definieren, der den Bash-Dienst erweitert, um in `~/.bash_profile` eine weitere Umgebungsvariable `PS1` definieren zu lassen:

```
(define bash-tolle-eingabeaufforderung-service
  (simple-service 'bash-tolle-eingabeaufforderung
                  home-bash-service-type
                  (home-bash-extension
                   (environment-variables
```

```
'(("PS1" . "\\u \\wλ "))))))
```

Den Dienst `bash-tolle-eingabeaufforderung-service` würden Sie anschließend ins `services`-Feld im `home-environment` Ihrer Persönlichen Umgebung eintragen. Nun folgt die Referenz zu `home-bash-extension`.

`home-bash-extension` [Datentyp]

Verfügbare `home-bash-extension`-Felder sind:

`environment-variables` (Vorgabe: `()`) (Typ: Assoziative-Liste)
: Zusätzliche Umgebungsvariable, die festgelegt werden sollen. Diese werden mit den Umgebungsvariablen anderer Erweiterungen und denen des zugrunde liegenden Dienstes zu einem zusammenhängenden Block aus Umgebungsvariablen vereint.

`aliases` (Vorgabe: `()`) (Typ: Assoziative-Liste)
: Zusätzliche Alias-Namen, die festgelegt werden sollen. Diese werden mit den Alias-Namen anderer Erweiterungen und denen des zugrunde liegenden Dienstes zusammengelegt.

`bash-profile` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Zusätzliche Textblöcke, die zu `.bash_profile` hinzugefügt werden sollen. Diese werden zu den Textblöcken anderer Erweiterungen und denen des zugrunde liegenden Dienstes hinzugenommen.

`bashrc` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Zusätzliche Textblöcke, die zu `.bashrc` hinzugefügt werden sollen. Diese werden zu den Textblöcken anderer Erweiterungen und denen des zugrunde liegenden Dienstes hinzugenommen.

`bash-logout` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Zusätzliche Textblöcke, die zu `.bash_logout` hinzugefügt werden sollen. Diese werden zu den Textblöcken anderer Erweiterungen und denen des zugrunde liegenden Dienstes hinzugenommen.

Persönlicher Zsh-Dienst

`home-zsh-configuration` [Datentyp]

Verfügbare `home-zsh-configuration`-Felder sind:

`package` (Vorgabe: `zsh`) (Typ: „package“)
: Das zu benutzende Zsh-Paket.

`xdg-flavor?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Ob alle Konfigurationsdateien in `$XDG_CONFIG_HOME/zsh` platziert werden sollen. Auch wird `~/.zshenv` dann `ZDOTDIR` auf `$XDG_CONFIG_HOME/zsh` festlegen. Der Startvorgang der Shell wird mit `$XDG_CONFIG_HOME/zsh/.zshenv` fortgeführt.

`environment-variables` (Vorgabe: `()`) (Typ: Assoziative-Liste)
: Eine assoziative Liste, welche Umgebungsvariable in der Zsh-Sitzung festgelegt sein sollen.

`zshenv` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Eine Liste dateiartiger Objekte, die zu `.zshenv` hinzugefügt werden. Damit können benutzereigene Shell-Umgebungsvariable festgelegt werden. Enthaltene Befehle dürfen *kein* vorhandenes TTY voraussetzen und sie dürfen *nichts* ausgeben. Die Datei wird immer gelesen. Sie wird vor jeder anderen Datei in `ZDOTDIR` gelesen.

`zprofile` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Eine Liste dateiartiger Objekte, die in `.zprofile` eingefügt werden. Damit können benutzereigene Befehle beim Start der Login-Shell ausgeführt werden. (In der Regel handelt es sich um die Shell, die gestartet wird, direkt nachdem man sich auf der Konsole (TTY) anmeldet.) `.zprofile` wird vor `.zlogin` gelesen.

`zshrc` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Eine Liste dateiartiger Objekte, die in `.zshrc` eingefügt werden. Damit können benutzereigene Befehle beim Start einer interaktiven Shell ausgeführt werden. (Interaktive Shells sind Shells für interaktive Nutzung, die man startet, indem man `zsh` eintippt oder die durch Terminal-Anwendungen oder andere Programme gestartet werden.)

`zlogin` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Eine Liste dateiartiger Objekte, die in `.zlogin` eingefügt werden. Damit können benutzereigene Befehle am Ende des Startvorgangs der Login-Shell ausgeführt werden.

`zlogout` (Vorgabe: `()`) (Typ: Konfigurationstexte)
: Eine Liste dateiartiger Objekte, die zu `.zlogout` hinzugefügt werden. Damit können benutzereigene Befehle beim Verlassen der Login-Shell ausgeführt werden. Die Datei wird in manchen Fällen *nicht* gelesen (wenn die Shell zum Beispiel durch exec eines anderen Prozesses terminiert).

13.3.3 Geplante Auftragsausführung durch Benutzer

Das Modul `(gnu home services mcron)` enthält eine Schnittstelle zu GNU mcron, einem Daemon, der gemäß einem vorher festgelegten Zeitplan Aufträge (sogenannte „Jobs“) ausführt (siehe *GNU mcron*). Es gelten hier dieselben Informationen wie beim mcron für Guix System (siehe Abschnitt 12.9.2 [Geplante Auftragsausführung], Seite 301), außer dass Persönliche mcron-Dienste in einem `home-environment`-Verbundsobjekt deklariert werden statt in einem `operating-system`-Verbundsobjekt.

`home-mcron-service-type` [Scheme-Variable]
: Dies ist der Diensttyp des Persönlichen `mcron`-Dienstes. Als Wert verwendet er ein `home-mcron-configuration`-Objekt. Hiermit können zu geplanten Zeiten Aufgaben durchgeführt werden.

  Dieser Diensttyp kann als Ziel einer Diensterweiterung verwendet werden, die ihn mit zusätzlichen Auftragsspezifikationen versorgt (siehe Abschnitt 12.18.1 [Dienstkompositionen], Seite 635). Mit anderen Worten ist es möglich, Dienste zu definieren, die weitere mcron-Aufträge ausführen lassen.

`home-mcron-configuration` [Datentyp]

Verfügbare `home-mcron-configuration`-Felder sind:

`mcron` (Vorgabe: `mcron`) (Typ: dateiartig)
: Welches mcron-Paket benutzt werden soll.

`jobs` (Vorgabe: `()`) (Typ: Liste-von-G-Ausdrücken)
: Dies muss eine Liste von G-Ausdrücken sein (siehe Abschnitt 9.12 [G-Ausdrücke], Seite 175), die jeweils einer mcron-Auftragsspezifikation (der Spezifikation eines „Jobs“) entsprechen (siehe Abschnitt “Syntax” in *GNU mcron*).

`log?` (Vorgabe: `#t`) (Typ: Boolescher-Ausdruck)
: Lässt Protokolle auf die Standardausgabe schreiben.

`log-format` (Vorgabe: `"~1@*~a ~a: ~a~%"`) (Typ: Zeichenkette)
: Eine Formatzeichenkette gemäß `(ice-9 format)` für die Protokollnachrichten. Mit dem Vorgabewert werden Nachrichten in der Form "'*Prozesskennung Name*: *Nachricht*"' geschrieben (siehe Abschnitt “Invoking mcron” in *GNU mcron*). Außerdem schreibt GNU Shepherd vor jeder Nachricht einen Zeitstempel.

13.3.4 Persönliche Dienste zur Stromverbrauchsverwaltung

Das Modul `(gnu home services pm)` stellt Persönliche Dienste bereit, die mit dem Batterieverbrauch zu tun haben.

`home-batsignal-service-type` [Scheme-Variable]

Hiermit kann ein Dienst für `batsignal` genutzt werden, einem Programm, um den Batterieladestand im Auge zu behalten und den Nutzer mit Benachrichtigungen zu warnen, wenn die Batterie leer wird. Sie können ihn auch so konfigurieren, dass ein Befehl ausgeführt wird, wenn der Ladestand eine „Gefahrstufe“ überschreitet. Dieser Dienst wird mit einem Objekt des `home-batsignal-configuration`-Verbundstyps konfiguriert.

`home-batsignal-configuration` [Datentyp]

Datentyp, der die Konfiguration von batsignal repräsentiert.

`warning-level` (Vorgabe: `15`)
: Bei welchem Batterieladestand eine Warnung vermeldet wird.

`warning-message` (Vorgabe: `#f`)
: Welche Nachricht als Benachrichtigung abgeschickt wird, wenn der Batterieladestand auf `warning-level` fällt. Wenn das auf `#f` gesetzt ist, wird die voreingestellte Nachricht abgeschickt.

`critical-level` (Vorgabe: `5`)
: Bei welchem Batterieladestand eine kritische Lage vermeldet wird.

`critical-message` (Vorgabe: `#f`)
: Welche Nachricht als Benachrichtigung abgeschickt wird, wenn der Batterieladestand auf `critical-level` fällt. Wenn das auf `#f` gesetzt ist, wird die voreingestellte Nachricht abgeschickt.

`danger-level` (Vorgabe: `2`)
: Bei welchem Batterieladestand der Befehl in `danger-command` ausgeführt wird.

`danger-command` (Vorgabe: `#f`)
: Welcher Befehl ausgeführt werden soll, wenn der Batterieladestand auf `danger-level` fällt. Wenn das auf `#f` gesetzt ist, wird gar kein Befehl ausgeführt.

`full-level` (Vorgabe: `#f`)
: Bei welchem Batterieladestand vermeldet wird, dass sie voll geladen ist. Wenn Sie das auf `#f` setzen, wird gar nichts gemeldet bei einer voll geladenen Batterie.

`full-message` (Vorgabe: `#f`)
: Welche Nachricht als Benachrichtigung abgeschickt wird, wenn der Batterieladestand auf `full-level` steigt. Wenn das auf `#f` gesetzt ist, wird die voreingestellte Nachricht abgeschickt.

`batteries` (Vorgabe: `'()`)
: Welche Batterien überwacht werden sollen. Wenn Sie das auf `'()` setzen, wird versucht, die Batterien automatisch zu finden.

`poll-delay` (Vorgabe: `60`)
: Die Zeit in Sekunden, wie lange abgewartet wird, bis die Batterien erneut geprüft werden.

`icon` (Vorgabe: `#f`)
: Ein dateiartiges Objekt, das als Symbolbild für die Batteriebenachrichtigungen verwendet wird. Wenn Sie das auf `#f` setzen, werden keine Symbole bei den Benachrichtigungen angezeigt.

`notifications?` (Vorgabe: `#t`)
: Ob überhaupt Benachrichtigungen abgeschickt werden sollen.

`notifications-expire?` (Vorgabe: `#f`)
: Ob Benachrichtigungen nach einiger Zeit auslaufen.

`notification-command` (Vorgabe: `#f`)
: Mit welchem Befehl Benachrichtigungen abgeschickt werden. Wenn Sie das auf `#f` setzen, werden Benachrichtigungen über `libnotify` zugestellt.

`ignore-missing?` (Vorgabe: `#f`)
: Ob Fehler, dass eine Batterie fehlt, ignoriert werden sollen.

13.3.5 Benutzer-Daemons verwalten

Das Modul `(gnu home services shepherd)` ermöglicht die Definition von Shepherd-Diensten für jeden Nutzer (siehe Abschnitt “Introduction” in *Handbuch von GNU Shepherd*). Dazu erweitern Sie `home-shepherd-service-type` mit neuen Diensten. Guix Home kümmert sich dann darum, dass der `shepherd`-Daemon für Sie beim Anmelden gestartet wird, und er startet wiederum die Dienste, die Sie anfordern.

`home-shepherd-service-type` [Scheme-Variable]

Der Diensttyp für das als normaler Benutzer ausgeführte Shepherd, womit man sowohl durchgängig laufende Prozesse als auch einmalig ausgeführte Aufgaben verwalten. Mit Benutzerrechten ausgeführte Shepherd-Prozesse sind *keine* „init“-Prozesse (PID 1), aber ansonsten sind fast alle Informationen aus Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644, auch für sie anwendbar.

Dieser Diensttyp stellt das Ziel für Diensterweiterungen dar, die Shepherd-Dienste erzeugen sollen (siehe Abschnitt 12.18.2 [Diensttypen und Dienste], Seite 637, für ein Beispiel). Jede Erweiterung muss eine Liste von `<shepherd-service>`-Objekten übergeben. Sein Wert muss eine `home-shepherd-configuration` sein, wie im Folgenden beschrieben.

`home-shepherd-configuration` [Datentyp]

Dieser Datentyp repräsentiert die Konfiguration von Shepherd.

`shepherd (Vorgabe: shepherd)`
: Das zu benutzende Shepherd-Paket.

`auto-start? (Vorgabe: #t)`
: Ob Shepherd bei einer ersten Anmeldung gestartet werden soll.

`services (Vorgabe: '())`
: Eine Liste zu startender Shepherd-Dienste als `<shepherd-service>`-Objekte. Wahrscheinlich sollten Sie stattdessen den Mechanismus zur Diensterweiterung benutzen (siehe Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644).

13.3.6 Secure Shell

Im OpenSSH-Paket (`https://www.openssh.com`) befindet sich ein Clientprogramm, nämlich der Befehl `ssh`, um eine Verbindung zu entfernten Maschinen über das SSH-Protokoll herzustellen (eine „Secure shell“). Mit dem Modul `(gnu home services ssh)` können Sie OpenSSH auf vorhersehbare Weise einrichten, nahezu unabhängig vom Zustand, in dem Ihr lokaler Rechner ist. Dazu instanziieren Sie `home-openssh-service-type` in Ihrer Persönlichen Konfiguration wie im Folgenden erklärt.

`home-openssh-service-type` [Scheme-Variable]

Dies ist der Diensttyp zum Einrichten des OpenSSH-Clients. Dadurch werden mehrere Dinge erledigt:

- Es wird eine Datei `~/.ssh/config` bereitgestellt, damit je nach Ihrer Konfiguration `ssh` die Rechner kennt, mit denen Sie sich regelmäßig verbinden, und Parameter damit assoziiert werden können.
- Es wird eine Datei `~/.ssh/authorized_keys` bereitgestellt, in der öffentliche Schlüssel eingetragen sind, wie man sich ausweisen muss, damit der lokale SSH-Server, `sshd`, Verbindungen zu diesem Benutzerkonto akzeptieren kann.
- Optional wird auch eine Datei `~/.ssh/known_hosts` bereitgestellt, damit `ssh` die Rechner authentifizieren kann, mit denen Sie sich verbinden.

Hier sehen Sie ein Beispiel, wie so ein Dienst aussehen und konfiguriert werden kann, wenn Sie ihn im `services`-Feld innerhalb von `home-environment` in Ihrer Persönlichen Konfiguration eintragen:

```
(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")))))
```

Im obigen Beispiel sind zwei Rechner mit Parametern angegeben, so dass wenn Sie etwa `ssh chbouib` ausführen, automatisch eine Verbindung zu `chbouib.example.org` auf Port 10022 hergestellt wird und Sie als Benutzer '`supercharlie`' angemeldet werden. Außerdem wird der öffentliche Schlüssel in `alice.pub` autorisiert und dessen Eigentümerin darf eingehende Verbindungen an Ihr Konto am Rechner aufbauen.

Mit einer `home-openssh-service-type`-Instanz des Dienstes muss ein `home-openssh-configuration`-Verbundsobjekt assoziiert werden; die Beschreibung folgt nun.

`home-openssh-configuration` [Datentyp]

Der Datentyp, der die Konfiguration für den OpenSSH-Client und auch den -Server bezüglich der Persönlichen Umgebung beschreibt. Dazu gehören die folgenden Felder:

`hosts` (Vorgabe: `'()`)

Eine Liste von `openssh-host`-Verbundsobjekten, mit denen Rechnernamen und damit assoziierte Verbindungsparameter festgelegt werden (siehe unten). Diese Rechnerliste wird in `~/.ssh/config` platziert, was `ssh` beim Start ausliest.

`known-hosts` (Vorgabe: `*unspecified*`)

Es muss eines hiervon sein:

- `*unspecified*`, in diesem Fall überlässt `home-openssh-service-type` es dem `ssh`-Programm und Ihnen als Benutzer, die Liste bekannter Rechner in `~/.ssh/known_hosts` zu pflegen, oder
- eine Liste dateiartiger Objekte, die dann zusammengefügt werden zu `~/.ssh/known_hosts`.

In der Datei `~/.ssh/known_hosts` steht eine Liste von Paaren aus Rechnername und zugehörigem Schlüssel, mit denen `ssh` die Rechner authentifiziert, mit denen Sie sich verbinden. So werden mögliche Angriffe mit Doppelgängern erkannt. Das vorgegebene Verhalten von `ssh` folgt dem Prinzip *TOFU, Trust-on-first-use*: Wenn Sie sich zum ersten Mal verbinden, wird der zum Rechner gehörende Schlüssel in dieser Datei für die Zukunft gespeichert. Genau so verhält sich `ssh`, wenn sie die Einstellung

von `known-hosts` unspezifiziert lassen (d.h. sie den Wert `*unspecified*` hat).

Wenn Sie stattdessen die Liste bekannter Rechnerschlüssel vorab im `known-hosts`-Feld hinterlegen, haben Sie eine eigenständige und zustandslose Konfiguration, die Sie auf anderen Rechnern jederzeit nachbilden können. Dafür stellt es beim ersten Mal einen Mehraufwand dar, die Liste aufzustellen, deshalb ist `*unspecified*` die Vorgabeeinstellung.

`authorized-keys` (Vorgabe: `'()`)
Hierfür muss eine Liste dateiartiger Objekte angegeben werden, von denen jedes einen öffentlichen SSH-Schlüssel enthält, für den es erlaubt ist, sich mit dieser Maschine zu verbinden.

Intern werden die Dateien zusammengefügt und als `~/.ssh/authorized_keys` bereitgestellt. Wenn auf diesem Rechner ein OpenSSH-Server, `sshd`, läuft, *kann* er diese Datei berücksichtigen; so verhält sich `sshd` in seiner Vorgabeeinstellung, aber Sie sollten wissen, dass man `sshd` auch so konfigurieren kann, dass es die Datei ignoriert.

`openssh-host` [Datentyp]

Verfügbare `openssh-host`-Felder sind:

`name` (Typ: Zeichenkette)
Der Name zu dieser Rechnerdeklaration.

`host-name` (Typ: Vielleicht-Zeichenkette)
Der Rechnername, z.B. `"foo.example.org"` oder `"192.168.1.2"`.

`address-family` (Typ: Adressfamilie)
Welche Adressfamilie benutzt werden soll, wenn eine Verbindung zum Rechner hergestellt wird: Entweder `AF_INET` (nur als IPv4-Verbindung), `AF_INET6` (nur als IPv6-Verbindung) oder Sie lassen es bei `*unspecified*` (wenn Ihnen jede Adressfamilie recht ist).

`identity-file` (Typ: Vielleicht-Zeichenkette)
Anhand welcher Identitätsdatei Sie sich authentisieren, z.B. `"/home/charlie/.ssh/id_ed25519"`.

`port` (Typ: Vielleicht-Natürliche-Zahl)
Die TCP-Portnummer, mit der eine Verbindung hergestellt wird.

`user` (Typ: Vielleicht-Zeichenkette)
Der Benutzername am entfernten Rechner.

`forward-x11?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
Ob Verbindungen an entfernte grafische X11-Clients an die grafische lokale X11-Anzeige weitergeleitet werden.

`forward-x11-trusted?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
Ob den entfernten X11-Clients Vollzugriff auf die eigene grafische X11-Anzeige gewährt werden soll.

`forward-agent?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob der Authentisierungsagent (falls vorhanden) an die entfernte Maschine weitergeleitet wird.

`compression?` (Vorgabe: `#f`) (Typ: Boolescher-Ausdruck)
: Ob übertragene Daten komprimiert werden.

`proxy-command` (Typ: Vielleicht-Zeichenkette)
: Was für ein Befehl aufgerufen werden soll, um die Verbindung zu diesem Server herzustellen. Zum Beispiel wäre ein Befehl, um sich mittels eines HTTP-Proxys auf 192.0.2.0 zu verbinden: `"nc -X connect -x 192.0.2.0:8080 %h %p"`.

`host-key-algorithms` (Typ: Vielleicht-Zeichenketten-Liste)
: Die Liste der Schlüsselalgorithmen, die für diesen Rechner akzeptiert werden, etwa `'("ssh-ed25519")`.

`accepted-key-types` (Typ: Vielleicht-Zeichenketten-Liste)
: Die Liste der akzeptierten Schlüsseltypen für öffentliche Schlüssel.

`extra-content` (Vorgabe: `""`) (Typ: Rohe-Konfigurations-Zeichenkette)
: Zusätzlicher Inhalt, der unverändert zu diesem `Host`-Block in `~/.ssh/config` noch angehängt wird.

13.3.7 Persönliche Desktop-Dienste

Das Modul `(gnu home services desktop)` stellt Dienste zur Verfügung, die Ihnen auf „Desktop“-Systemen helfen können, d.h. wenn Sie eine grafische Arbeitsumgebung mit z.B. Xorg gebrauchen.

`home-redshift-service-type` [Scheme-Variable]
: Dies ist der Diensttyp für Redshift (`https://github.com/jonls/redshift`), ein Programm, um die Farbtemperatur des Bildschirms an die Tageszeit anzupassen. Sein zugewiesener Wert muss ein `home-redshift-configuration`-Verbundsobjekt sein wie im folgenden Beispiel:

  Eine typische Konfiguration, bei der wir Längen- und Breitengrad selbst vorgeben, könnte so aussehen:

```
(service home-redshift-service-type
         (home-redshift-configuration
          (location-provider 'manual)
          (latitude 35.81)    ;Nordhalbkugel
          (longitude -0.80))) ;westlich von Greenwich
```

`home-redshift-configuration` [Datentyp]
: Verfügbare `home-redshift-configuration`-Felder sind:

  `redshift` (Vorgabe: `redshift`) (Typ: dateiartig)
  : Das zu verwendende Redshift-Paket.

  `location-provider` (Vorgabe: `geoclue2`) (Typ: Symbol)
  : Anbieter für die Ortsbestimmung („Geolocation“). Entweder Sie geben den Ort manuell ein, dann schreiben Sie `'manual`, oder für eine automa-

tische Ortsbestimmung schreiben Sie `'geoclue2`. Wenn Sie den Ort manuell eingeben möchten, müssen Sie außerdem Breiten- und Längengrad in den Feldern `latitude` und `longitude` festlegen, damit Redshift die Tageszeit bei Ihnen ermitteln kann. Wenn Sie die automatische Ortsbestimmung benutzen möchten, muss der Geoclue-Systemdienst laufen, der die Ortsinformation bringt.

`adjustment-method` (Vorgabe: `randr`) (Typ: Symbol)
Die Methode zur Farbanpassung.

`daytime-temperature` (Vorgabe: `6500`) (Typ: Ganze-Zahl)
Farbtemperatur am Tag (in Kelvin).

`nighttime-temperature` (Vorgabe: `4500`) (Typ: Ganze-Zahl)
Farbtemperatur bei Nacht (in Kelvin).

`daytime-brightness` (Typ: Vielleicht-Inexakte-Zahl)
Bildschirmhelligkeit bei Tag, zwischen 0.1 und 1.0 oder unspezifiziert.

`nighttime-brightness` (Typ: Vielleicht-Inexakte-Zahl)
Bildschirmhelligkeit in der Nacht, zwischen 0.1 und 1.0 oder unspezifiziert.

`latitude` (Typ: Vielleicht-Inexakte-Zahl)
Der Breitengrad, wenn `location-provider` auf `'manual` gestellt ist.

`longitude` (Typ: Vielleicht-Inexakte-Zahl)
Der Längengrad, wenn `location-provider` auf `'manual` gestellt ist.

`dawn-time` (Typ: Vielleicht-Zeichenkette)
Eine selbst festgelegte Uhrzeit, zu der morgens von Nacht auf Tag geschaltet wird, im Format `"HH:MM"`. Wenn Sie dies angeben, wird der Sonnenstand zur Ermittlung von Tag und Nacht *nicht* herangezogen.

`dusk-time` (Typ: Vielleicht-Zeichenkette)
Entsprechend eine selbst festgelegte Uhrzeit, zu der abends von Tag auf Nacht geschaltet wird.

`extra-content` (Vorgabe: `""`) (Typ: Rohe-Konfigurations-Zeichenkette)
Weiterer Text, der unverändert an die Redshift-Konfigurationsdatei angehängt wird. Führen Sie `man redshift` aus, um weitere Informationen über das Format der Konfigurationsdatei zu erfahren.

`home-dbus-service-type` [Scheme-Variable]
Mit diesem Diensttyp können Sie eine Instanz von D-Bus nur für die aktuelle Sitzung ausführen. Er ist gedacht für Anwendungen ohne besondere Berechtigung, die eine laufende D-Bus-Instanz voraussetzen.

`home-dbus-configuration` [Datentyp]
Das Verbundsobjekt mit der Konfiguration des `home-dbus-service-type`.

`dbus` (Vorgabe: `dbus`)
Das Paket mit dem Befehl `/bin/dbus-daemon`.

13.3.8 Persönliche Guix-Dienste

Das Modul `(gnu home services guix)` bietet Dienste an, um Guix für den Benutzer einzurichten.

`home-channels-service-type` [Scheme-Variable]

Dies ist der Diensttyp, um `$XDG_CONFIG_HOME/guix/channels.scm` einzurichten. Mit dieser Datei wird gesteuert, welche Kanäle mit `guix pull` empfangen werden (siehe Kapitel 7 [Kanäle], Seite 77). Sein zugewiesener Wert muss eine Liste von `channel`-Verbundsobjekten sein, wie sie im Modul `(guix channels)` definiert sind.

Es ist im Allgemeinen besser, eine Erweiterung für diesen Dienst zu machen, statt den Dienst direkt zu konfigurieren, denn in seinem Vorgabewert sind die vorgegebenen Kanäle für Guix, die als `%default-channels` definiert sind, bereits enthalten. Wenn Sie sich entscheiden, diesen Dienst direkt zu konfigurieren, müssen Sie darauf achten, dass ein `guix`-Kanal konfiguriert ist. Siehe Abschnitt 7.1 [Weitere Kanäle angeben], Seite 77, und Abschnitt 7.2 [Eigenen Guix-Kanal benutzen], Seite 77, für weitere Details.

Eine typische Konfiguration, um einen Kanal hinzuzufügen, könnte so aussehen:

```
(simple-service 'paketvarianten-dienst
                home-channels-service-type
                (list
                 (channel
                  (name 'paketvarianten)
                  (url "https://example.org/variant-packages.git"))))
```

13.4 `guix home` aufrufen

Sobald Sie eine Deklaration Ihrer Persönlichen Umgebung haben (siehe Abschnitt 13.1 [Deklaration der Persönlichen Umgebung], Seite 656), kann diese *instanziiert* werden, indem Sie den Befehl `guix home` aufrufen. Zusammengefasst:

```
guix home Optionen... Aktion Datei
```

Datei muss der Name einer Datei sein, in der eine Persönliche Umgebung als `home-environment`-Objekt steht. *Aktion* gibt an, wie das Betriebssystem instanziiert wird (abgesehen von unterstützenden Aktionen, wo nichts instanziiert wird). Derzeit werden folgende Werte dafür unterstützt:

`search` Verfügbare Definitionen Persönlicher Dienste anzeigen, die zum angegebenen regulären Ausdruck passen, sortiert nach Relevanz:

```
$ 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

...
```

Wie auch bei `guix search` wird das Ergebnis im `recutils`-Format geliefert, so dass es leicht ist, die Ausgabe zu filtern (siehe *Handbuch von GNU recutils*).

`container`

Eine Shell in einer isolierten Umgebung – einem *Container* – öffnen, die die in *Datei* angegebene Persönliche Umgebung enthält.

Zum Beispiel würden Sie so eine interaktive Shell in einem Container starten, der Ihrer Persönlichen Umgebung entspricht:

```
guix home container config.scm
```

In diesem Wegwerf-Container können Sie Dateien nach Herzenslust verändern, denn innerhalb des Containers gemachte Änderungen oder gestartete Prozesse sind alle wieder weg, sobald Sie die Shell verlassen.

Wie auch `guix shell` hält sich der Container an einige Befehlszeilenoptionen:

`--network`
`-N`
: Netzwerkzugriff im Container erlauben (was nach Voreinstellung abgeschaltet ist).

`--expose=`*`Quelle`*`[=`*`Ziel`*`]`
`--share=`*`Quelle`*`[=`*`Ziel`*`]`
: Wie bei `guix shell` wird das Verzeichnis unter *Quelle* vom Wirtssystem im Container als *Ziel* verfügbar gemacht. Bei `--expose` gibt es nur Lesezugriff und bei `--share` auch Schreibzugriff darauf (siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86).

Des Weiteren können Sie einen Befehl im Container ausführen lassen, anstatt eine interaktive Shell zu starten. Zum Beispiel würden Sie so überprüfen, welche Shepherd-Dienste im Persönlichen Wegwerf-Container gestartet werden:

```
guix home container config.scm -- herd status
```

Den Befehl, der im Container auszuführen ist, geben Sie nach zwei kurzen Strichen `--` an.

`edit` Die Definition des angegebenen Persönlichen Dienstes bearbeiten oder anzeigen.

Beispielsweise öffnet folgender Befehl Ihren Editor, der mit der Umgebungsvariablen `EDITOR` gewählt werden kann, an der Stelle, wo der `home-mcron`-Diensttyp definiert ist:

```
guix home edit home-mcron
```

`reconfigure`

Die in der *Datei* beschriebene Persönliche Umgebung erstellen und zu ihr wechseln. Wechseln bedeutet, dass das Aktivierungsskript ausgewertet wird und (in der Grundeinstellung) symbolische Verknüpfungen auf Konfigurationsdateien, die anhand der Deklaration der Persönlichen Umgebung im `home-environment`-Objekt erzeugt werden, in `~` angelegt werden. Wenn die Datei mit demselben Pfad bereits im Persönlichen Verzeichnis existiert, wird sie nach *`~/Zeitstempel`*`-guix-home-legacy-configs-backup` verschoben. Dabei ist *Zeitstempel* eine Zeitangabe seit der UNIX-Epoche.

> **Anmerkung:** Es ist sehr zu empfehlen, `guix pull` einmal auszuführen, bevor Sie `guix home reconfigure` zum ersten Mal aufrufen (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65).

Dieser Befehl setzt die in der *Datei* festgelegte Konfiguration vollständig um. Der Befehl startet die in der *Datei* angegebenen Shepherd-Dienste, die aktuell nicht laufen; bei aktuell laufenden Diensten wird sichergestellt, dass sie aktualisiert werden, sobald sie das nächste Mal angehalten wurden (z.B. durch `herd stop` *`Dienst`* oder `herd restart` *`Dienst`*).

Dieser Befehl erzeugt eine neue Generation, deren Nummer (wie `guix home list-generations` sie anzeigt) um eins größer als die der aktuellen Generation ist. Wenn die so nummerierte Generation bereits existiert, wird sie überschrieben. Dieses Verhalten entspricht dem von `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

Nach Abschluss wird die neue Persönliche Umgebung unter `~/.guix-home` verfügbar gemacht. Das Verzeichnis enthält *Provenienz-Metadaten*: Dazu gehören die Liste der Kanäle, die benutzt wurden (siehe Kapitel 7 [Kanäle], Seite 77) und die *Datei* selbst, wenn sie verfügbar ist. Sie können die Provenienzinformationen auf diese Weise ansehen:

```
guix home describe
```

Diese Informationen sind nützlich, falls Sie später inspizieren möchten, wie diese spezielle Generation erstellt wurde. Falls die *Datei* eigenständig ist, also keine anderen Dateien zum Funktionieren braucht, dann können Sie tatsächlich die Generation *n* Ihrer Persönlichen Umgebung später erneut erstellen mit:

```
guix time-machine \
  -C /var/guix/profiles/per-user/Benutzer/guix-home-n-link/channels.scm -- \
  home reconfigure \
  /var/guix/profiles/per-user/Benutzer/guix-home-n-link/configuration.scm
```

Sie können sich das als eine Art eingebaute Versionskontrolle vorstellen! Ihre Persönliche Umgebung ist nicht nur ein binäres Erzeugnis: *Es enthält seinen eigenen Quellcode.*

`switch-generation`
: Zu einer bestehenden Generation der Persönlichen Umgebung wechseln. Diese Aktion wechselt das Profil der Persönlichen Umgebung atomar auf die angegebene Generation der Persönlichen Umgebung.

  Die Zielgeneration kann ausdrücklich über ihre Generationsnummer angegeben werden. Zum Beispiel würde folgender Aufruf einen Wechsel zur Generation 7 der Persönlichen Umgebung bewirken:

  ```
  guix home switch-generation 7
  ```

  Die Zielgeneration kann auch relativ zur aktuellen Generation angegeben werden, in der Form `+N` oder `-N`, wobei `+3` zum Beispiel „3 Generationen weiter als die aktuelle Generation“ bedeuten würde und `-1` „1 Generation vor der aktuellen Generation“ hieße. Wenn Sie einen negativen Wert wie `-1` angeben, müssen Sie `--` der Befehlszeilenoption voranstellen, damit die negative Zahl nicht selbst als Befehlszeilenoption aufgefasst wird. Zum Beispiel:

  ```
  guix home switch-generation -- -1
  ```

  Diese Aktion schlägt fehl, wenn die angegebene Generation nicht existiert.

`roll-back`
: Zur vorhergehenden Generation der Persönlichen Umgebung wechseln. Dies ist die Umkehrung von `reconfigure` und tut genau dasselbe wie `switch-generation` mit dem Argument `-1` aufzurufen.

`delete-generations`
: Generationen der Persönlichen Umgebung löschen, wodurch diese zu Kandidaten für den Müllsammler werden (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61, für Informationen, wie Sie den „Müllsammler“ laufen lassen).

  Es funktioniert auf die gleiche Weise wie ‘`guix package --delete-generations`’ (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Wenn keine Argumente angegeben werden, werden alle Generationen der Persönlichen Umgebung außer der aktuellen gelöscht:

  ```
  guix home delete-generations
  ```

  Sie können auch eine Auswahl treffen, welche Generationen Sie löschen möchten. Das folgende Beispiel hat die Löschung aller Generationen der Persönlichen Umgebung zur Folge, die älter als zwei Monate sind:

  ```
  guix home delete-generations 2m
  ```

`build`
: Die Ableitung der Persönlichen Umgebung erstellen, einschließlich aller Konfigurationsdateien und Programme, die benötigt werden. Diese Aktion installiert jedoch nichts davon.

`describe`
: Die aktuelle Generation der Persönlichen Umgebung beschreiben: ihren Dateinamen sowie Provenienzinformationen, falls verfügbar.

  Um installierte Pakete im Profil der aktuellen Persönlichen Generation zu finden, wird die Befehlszeilenoption `--list-installed` angeboten, deren Syntax

dieselbe ist wie bei `guix package --list-installed` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45). Zum Beispiel zeigt der folgende Befehl eine Tabelle mit allen Paketen, deren Name „emacs“ enthält und die ins Profil der aktuellen Persönlichen Generation installiert sind, an:

```
guix home describe --list-installed=emacs
```

`list-generations`
: Eine für Menschen verständliche Zusammenfassung jeder auf der Platte verfügbaren Generation der Persönlichen Umgebung ausgeben. Dies ähnelt der Befehlszeilenoption `--list-generations` von `guix package` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

  Optional kann ein Muster angegeben werden, was dieselbe Syntax wie `guix package --list-generations` benutzt, um damit die Liste anzuzeigender Generationen einzuschränken. Zum Beispiel zeigt der folgende Befehl Generationen an, die bis zu 10 Tage alt sind:

  ```
  guix home list-generations 10d
  ```

  Sie können auch die Befehlszeilenoption `--list-installed` angeben mit derselben Syntax wie in `guix home describe`. Das kann nützlich sein, wenn Sie herausfinden möchten, wann ein bestimmtes Paket zum Persönlichen Profil hinzukam.

`import`
: Eine Deklaration der Persönlichen Umgebung anhand der Pakete im Standardprofil und der Konfigurationsdateien im Persönlichen Verzeichnis des Benutzers anlegen. Dabei werden die Konfigurationsdateien in das angegebene Verzeichnis kopiert und eine `home-configuration.scm` wird darin mit dem, was die Persönliche Umgebung ausmacht, gefüllt. Beachten Sie, dass `guix home import` nicht alle Persönlichen Dienste, die es gibt, unterstützt (siehe Abschnitt 13.3 [Persönliche Dienste], Seite 659).

  ```
  $ guix home import ~/guix-config
  guix home: Alle Konfigurationsdateien für die Persönliche Umgebung wurden in
  ```

Es gibt noch mehr zu sehen! Mit `guix home` können Sie folgende Unterbefehle benutzen, um zu visualisieren, wie die Dienste Ihrer Persönlichen Umgebung voneinander abhängen:

`extension-graph`
: Auf die Standardausgabe den *Diensterweiterungsgraphen* der in der *Datei* definierten Persönlichen Umgebung ausgeben (siehe Abschnitt 12.18.1 [Dienstkompositionen], Seite 635, für mehr Informationen zu Diensterweiterungen). Vorgegeben ist, ihn im Dot-/Graphviz-Format auszugeben, aber Sie können ein anderes Format mit `--graph-backend` auswählen, genau wie bei `guix graph` (siehe Abschnitt 10.10 [Aufruf von guix graph], Seite 228):

  Der Befehl:

  ```
  guix home extension-graph Datei | xdot -
  ```

  zeigt die Erweiterungsrelation unter Diensten an.

`shepherd-graph`
: Auf die Standardausgabe den *Abhängigkeitsgraphen* der Shepherd-Dienste der in der *Datei* definierten Persönlichen Umgebung ausgeben. Siehe

Abschnitt 12.18.4 [Shepherd-Dienste], Seite 644, für mehr Informationen sowie einen Beispielgraphen.

Auch hier wird nach Vorgabe die Ausgabe im Dot-/Graphviz-Format sein, aber Sie können ein anderes Format mit `--graph-backend` auswählen.

Unter den *Optionen* können beliebige gemeinsame Erstellungsoptionen aufgeführt werden (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189). Des Weiteren kann als *Optionen* Folgendes angegeben werden:

`--expression=`*Ausdruck*
`-e` *Ausdruck*

Als Konfiguration der Persönlichen Umgebung das `home-environment`-Objekt betrachten, zu dem der *Ausdruck* ausgewertet wird. Dies ist eine Alternative dazu, die Konfiguration in einer Datei festzulegen.

`--allow-downgrades`

An `guix home reconfigure` die Anweisung erteilen, Systemherabstufungen zuzulassen.

Genau wie bei `guix system` verhindert `guix home reconfigure` standardmäßig, dass Sie Ihre Persönliche Umgebung herabstufen auf ältere Versionen oder auf Versionen, die mit den Kanalversionen Ihrer vorigen Persönlichen Umgebung (`guix home describe` zeigt diese) *nicht* zusammenhängen. Sie können `--allow-downgrades` angeben, um die Überprüfung zu umgehen, und tragen dann selbst die Schuld, wenn Sie Ihre Persönliche Umgebung herabstufen. Vorsicht ist geboten!

14 Dokumentation

In den meisten Fällen liegt den mit Guix installierten Paketen auch Dokumentation bei, die diese beschreibt. Die zwei üblichsten Formate für Dokumentation sind „Info“, ein durchsuchbares Hypertextformat, das für GNU-Software benutzt wird, und sogenannte „Handbuchseiten“ (englisch „Manual Pages“, kurz Man-Pages), das linear aufgebaute Dokumentationsformat, das auf Unix traditionell mitgeliefert wird. Info-Handbücher können mit dem Befehl `info` oder mit Emacs abgerufen werden, auf Handbuchseiten kann mit dem Befehl `man` zugegriffen werden.

Sie können die Dokumentation von auf Ihrem System installierter Software nach einem Schlüsselwort durchsuchen. Zum Beispiel suchen Sie mit folgendem Befehl in den Info-Handbüchern nach „TLS“.

```
$ 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
...
```

Mit folgendem Befehl suchen Sie dasselbe Schlüsselwort in Handbuchseiten[1]:

```
$ man -k TLS
SSL (7)              - OpenSSL SSL/TLS library
certtool (1)         - GnuTLS certificate tool
...
```

Diese Suchvorgänge finden ausschließlich lokal auf Ihrem Rechner statt, wodurch gewährleistet ist, dass die Fundstellen zur von Ihnen auch tatsächlich installierten Software passen, Sie für den Zugriff keine Internetverbindung brauchen und Datenschutz gewährleistet bleibt.

Sobald Sie die Fundstellen kennen, können Sie zum Beispiel so die entsprechende Dokumentation anzeigen lassen:

```
$ info "(gnutls)Core TLS API"
```

oder

```
$ man certtool
```

Info-Handbücher sind in Abschnitte unterteilt und verfügen über Register sowie Hyperlinks, wie jene, die Sie auch von Webseiten kennen. Der `info`-Betrachter (siehe *Stand-alone GNU Info*) und sein Gegenstück für Emacs (siehe Abschnitt “Misc Help” in *The GNU Emacs Manual*) verfügen über leicht erlernbare Tastenkürzel, mit denen Sie in Handbüchern navigieren können. Siehe Abschnitt “Getting Started” in *Info: An Introduction* für eine Einführung in die Info-Navigation.

[1] Die von `man -k` durchsuchte Datenbank wird nur erstellt, wenn Ihre Umgebung das Paket `man-db` enthält.

15 Plattformen

Wir erstellen mit Guix Pakete und Systeme, die, wie die meisten anderen Computerprogramme auch, auf einen festgelegten Befehlssatz eines Prozessors und auf ein festgelegtes Betriebssystem abzielen. Oft laufen sie sogar auf einem bestimmten Kernel und einer bestimmten Systembibliothek. Diese Einschränkungen fasst Guix in `platform`-Verbundsobjekten zusammen.

15.1 `platform`-Referenz

Mit dem `platform`-Datentyp wird eine *Plattform* beschrieben: ein Befehlssatz („Instruction Set Architecture“, ISA) zusammen mit einem Betriebssystem und möglicherweise weiteren systemweiten Einstellungen wie ihrer Binärschnittstelle („Application Binary Interface“, ABI).

`platform` [Datentyp]

Dieser Datentyp steht für eine Plattform.

`target` Mit diesem Feld legt man das der Plattform entsprechende GNU-Tripel als Zeichenkette fest (siehe Abschnitt “Specifying Target Triplets” in *Autoconf*).

`system` Diese Zeichenkette ist der Systemtyp, den man in Guix verwendet und zum Beispiel an die Befehlszeilenoption `--system` bei den meisten Befehlen übergibt.

Dessen Form entspricht meistens `"`*`CPU-Kernel`*`"`, wobei *CPU* den Prozessor eines Zielrechners angibt und *Kernel* den Betriebssystem-Kernel eines Zielrechners angibt.

Möglich ist zum Beispiel `"aarch64-linux"` oder `"armhf-linux"`. Systemtypen brauchen Sie, wenn Sie nativ erstellen möchten (siehe Abschnitt 11.2 [Native Erstellungen], Seite 248).

`linux-architecture` (Vorgabe: `#false`)
Diese optionale Zeichenkette interessiert nur, wenn Linux als Kernel festgelegt ist. In diesem Fall entspricht sie der ARCH-Variablen, die benutzt wird, wenn Linux erstellt wird. Ein Beispiel ist `"mips"`.

`glibc-dynamic-linker`
Dieses Feld enthält den Namen des dynamischen Binders der GNU-C-Bibliothek des entsprechenden Systems in Form einer Zeichenkette. Ein Beispiel ist `"/lib/ld-linux-armhf.so.3"`.

15.2 Unterstützte Plattformen

Die Module namens `(guix platforms ...)` exportieren die folgenden Variablen, von denen jede an ein `platform`-Verbundsobjekt gebunden ist.

`armv7-linux` [Scheme-Variable]

Diese Plattform zielt auf ARMv7-Prozessoren ab, auf denen GNU/Linux läuft.

`aarch64-linux` [Scheme-Variable]
Diese Plattform zielt auf ARMv8-Prozessoren ab, auf denen GNU/Linux läuft.

`mips64-linux` [Scheme-Variable]
Diese Plattform zielt auf MIPS-64-Bit-Prozessoren in Little-Endian ab, auf denen GNU/Linux läuft.

`powerpc-linux` [Scheme-Variable]
Diese Plattform zielt auf PowerPC-32-Bit-Prozessoren in Big-Endian ab, auf denen GNU/Linux läuft.

`powerpc64le-linux` [Scheme-Variable]
Diese Plattform zielt auf PowerPC-64-Bit-Prozessoren in Little-Endian ab, auf denen GNU/Linux läuft.

`riscv64-linux` [Scheme-Variable]
Diese Plattform zielt auf RISC-V-64-Bit-Prozessoren ab, auf denen GNU/Linux läuft.

`i686-linux` [Scheme-Variable]
Diese Plattform zielt auf x86-Prozessoren ab, auf denen GNU/Linux läuft.

`x86_64-linux` [Scheme-Variable]
Diese Plattform zielt auf x86-64-Bit-Prozessoren ab, auf denen GNU/Linux läuft.

`i686-mingw` [Scheme-Variable]
Diese Plattform zielt auf x86-Prozessoren ab, die die Laufzeitunterstützung durch MinGW nutzen.

`x86_64-mingw` [Scheme-Variable]
Diese Plattform zielt auf x86-64-Bit-Prozessoren ab, die die Laufzeitunterstützung durch MinGW nutzen.

`i586-gnu` [Scheme-Variable]
Diese Plattform zielt auf x86-Prozessoren ab, auf denen GNU/Hurd läuft (was auch als „GNU“ bezeichnet wird).

16 Systemabbilder erstellen

Wenn Sie beabsichtigen, Guix System zum ersten Mal auf einer neuen Maschine zu installieren, gibt es im Grunde drei Möglichkeiten, wie Sie vorgehen können. Erstens können Sie ausgehend von einem bestehenden Betriebssystem, das sich bereits auf der Maschine befindet, den Befehl `guix system init` aufrufen (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619). Die zweite Möglichkeit ist, ein Installationsabbild vorzubereiten (siehe Abschnitt 3.9 [Ein Abbild zur Installation erstellen], Seite 38). Von diesem bootfähigen System aus wird dann schließlich `guix system init` durchgeführt. Zu guter Letzt bleibt eine dritte Möglichkeit: Sie bereiten ein Abbild vor, dass direkt eine Instanz des gewünschten Systems enthält. Sie kopieren das Abbild dann auf ein bootfähiges Gerät, sagen wir ein USB-Laufwerk oder eine Speicherkarte, und der Zielrechner bootet direkt davon. Eine Installationsprozedur findet *nicht* statt.

Mit dem Befehl `guix system image` sind Sie in der Lage, aus einer Betriebssystemdefinition ein bootfähiges Abbild, englisch „Image“, anzufertigen. Der Befehl kann mehrere Typen von Abbild bereitstellen wie `efi-raw`, `iso9660` oder `docker`. Jede aktuelle `x86_64`-Maschine dürfte von einem `iso9660`-Abbild aus starten können. Jedoch gibt es auch Maschinen da draußen, für die eigens zugeschnittene Abbildtypen vonnöten sind. In der Regel haben diese Maschinen `ARM`-Prozessoren und sie setzen eventuell voraus, dass bestimmte Partitionen an je einem bestimmten Versatz zu finden sind.

In diesem Kapitel wird erklärt, wie Sie eigene Abbildtypen definieren können und wie Sie daraus bootfähige Images erstellen lassen.

16.1 `image`-Referenz

Der nun beschriebene Verbundstyp `image` ermöglicht es, ein angepasstes, bootfähiges Systemabbild zu definieren.

`image` [Datentyp]

Dieser Datentyp steht für ein Systemabbild.

`name` (Vorgabe: `#false`)
: Der Name, den das Abbild tragen soll, als Symbol; zum Beispiel `'my-iso9660`. Der Name ist optional; vorgegeben ist `#false`.

`format`
: Das Abbildformat als Symbol. Folgende Abbildformate werden unterstützt:

  - `disk-image`, ein rohes Disk-Image aus einer oder mehreren Partitionen.
  - `compressed-qcow2`, ein komprimiertes Abbild im qcow2-Format, das aus einer oder mehreren Partitionen besteht.
  - `docker`, ein Abbild für Docker.
  - `iso9660`, ein Abbild im ISO-9660-Format.
  - `tarball`, ein Abbild als tar.gz-Archiv.
  - `wsl2`, ein Abbild für WSL2.

`platform` (Vorgabe: `#false`)
: Welchem `platform`-Verbundsobjekt der/die Zielrechner des Abbilds entsprechen (siehe Kapitel 15 [Plattformen], Seite 681); zum Beispiel `aarch64-linux`. Nach Vorgabe steht dieses Feld auf `#false`, so dass das Abbild dieselbe Plattform wie beim Wirtssystem als Ziel annimmt.

`size` (Vorgabe: `'guess`)
: Die Größe des Abbilds in Bytes oder das Symbol `'guess`. Wenn `'guess` angegeben wird, was vorgegeben ist, wird das Abbild so groß wie es der Inhalt des Abbilds vorgibt.

`operating-system`
: Das `operating-system`-Verbundsobjekt, das für das Abbild instanziiert wird.

`partition-table-type` (Vorgabe: `'mbr`)
: Der Partitionstabellentyp für das Abbild als ein Symbol. Mögliche Werte sind `'mbr` und `'gpt`. Vorgegeben ist `'mbr`.

`partitions` (Vorgabe: `'()`)
: Die Partitionen auf dem Abbild als eine Liste von `partition`-Verbundsobjekten (siehe Abschnitt 16.1.1 [„partition“-Referenz], Seite 685).

`compression?` (Vorgabe: `#true`)
: Ob der Inhalt des Abbilds komprimiert werden soll, angegeben als Boolescher Ausdruck. Vorgegeben ist `#true` und die Wahl hat nur einen Einfluss, wenn das Abbildformat `'iso9660` lautet.

`volatile-root?` (Vorgabe: `#true`)
: Ob die Wurzelpartition auf dem Abbild flüchtig sein soll, angegeben als Boolescher Ausdruck.

  Dazu wird ein im Arbeitsspeicher vorgehaltenes Dateisystem (overlayfs) über der Wurzelpartition durch die initrd eingebunden. Vorgegeben ist `#true`. Wenn es auf `#false` gesetzt ist, wird die Wurzelpartition auf dem Abbild mit Lese- *und* Schreibzugriff von der initrd eingebunden.

`shared-store?` (Vorgabe: `#false`)
: Ob der Store auf dem Abbild mit dem Wirtssystem geteilt werden soll, angegeben als Boolescher Ausdruck. Das können Sie dann gebrauchen, wenn Sie Abbilder für virtuelle Maschinen erstellen. Wenn es auf `#false` steht, wie es vorgegeben ist, dann wird der Abschluss des erstellten `operating-system` auf das Abbild kopiert. Wenn es dagegen auf `#true` steht, wird angenommen, dass der Store des Wirtssystems beim Start zur Verfügung gestellt werden wird, zum Beispiel, indem er als `9p`-Einhängepunkt eingebunden wird.

`shared-network?` (Vorgabe: `#false`)
: Ob die Netzwerkschnittstelle des Wirtssystems mit dem Abbild geteilt wird, angegeben als Boolescher Ausdruck. Dies hat nur einen Einfluss, wenn als Abbildformat `'docker` verwendet wird. Vorgegeben ist `#false`.

`substitutable?` (Vorgabe: `#true`)
: Ob für die Ableitung des Abbilds Substitute benutzt werden können, angegeben als Boolescher Ausdruck. Vorgegeben ist `true`.

16.1.1 `partition`-Referenz

Ein `image`-Verbundsobjekt kann Partitionen enthalten.

`partition` [Datentyp]
: Dieser Datentyp steht für eine Partition auf einem Abbild.

  `size` (Vorgabe: `'guess`)
  : Die Größe der Partition in Bytes oder das Symbol `'guess`. Wenn `'guess` angegeben wird, was vorgegeben ist, wird die Partition so groß wie es der Inhalt der Partition vorgibt.

  `offset` (Vorgabe: `0`)
  : Bei welchem Versatz in Bytes die Partition losgeht, relativ zum Anfang des Abbilds oder zum Ende der vorherigen Partition. Vorgegeben ist `0`, wodurch kein Versatz gelassen wird.

  `file-system` (Vorgabe: `"ext4"`)
  : Das Dateisystem der Partition als eine Zeichenkette; vorgegeben ist `"ext4"`. Als Wert werden `"vfat"`, `"fat16"`, `"fat32"` und `"ext4"` unterstützt.

  `file-system-options` (Vorgabe: `'()`)
  : Die Befehlszeilenoptionen, die bei der Erstellung der Partition an das Werkzeug zur Partitionserstellung übergeben werden, angegeben als eine Liste von Zeichenketten. Dies wird nur für die Erstellung von `"ext4"`-Partitionen unterstützt.

    Siehe den Abschnitt zu `"extended-options"` in der Handbuchseite des `"mke2fs"`-Werkzeugs für eine umfassendere Übersicht.

  `label` Die Bezeichnung der Partition. Diese Zeichenkette *muss* angegeben werden. Ein Beispiel wäre `"my-root"`.

  `uuid` (Vorgabe: `#false`)
  : Die UUID der Partition als ein `uuid`-Verbundsobjekt (siehe Abschnitt 12.3 [Dateisysteme], Seite 263). Vorgegeben ist `#false`, wodurch das Werkzeug zur Partitionserstellung eine zufällige UUID an die Partition vergeben wird.

  `flags` (Vorgabe: `'()`)
  : Mit welchen Flags die Partition erzeugt wird, angegeben als Liste von Symbolen. Möglich sind `'boot` und `'esp`. Lassen Sie die `'boot`-Flag setzen, wenn Sie von dieser Partition booten möchten. Genau eine Partition sollte diese Flag tragen, in der Regel die Wurzelpartition. Mit der `'esp`-Flag wird eine UEFI-Systempartition gekennzeichnet.

  `initializer` (Vorgabe: `#false`)
  : Die Initialisierungsprozedur der Partition als ein G-Ausdruck. Diese Prozedur wird aufgerufen, um Dateien zur Partition hinzuzufügen.

Wenn keine Initialisierungsprozedur angegeben wird, wird die Prozedur `initialize-root-partition` aus dem Modul `(gnu build image)` verwendet.

16.2 Abbilder instanziieren

Sagen wir, Sie möchten ein MBR-formatiertes Abbild mit drei verschiedenen Partitionen erzeugen:

- Der ESP (EFI-Systempartition), einer Partition mit 40 MiB versetzt um 1024 KiB mit einem vfat-Dateisystem.
- Einer ext4-Partition mit 50 MiB für eine Datei mit Daten und der Bezeichnung „data“.
- Einer bootfähigen ext4-Partition, auf der `%simple-os` als Betriebssystem liegt.

Dazu würden Sie zum Beispiel folgende Betriebssystemdefinition in eine Datei `my-image.scm` schreiben.

```
(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))))))
```

Wir merken an, dass für die erste und dritte Partition hier jeweils die allgemeinen Initialisierungsprozeduren `initialize-efi-partition` und `initialize-root-partition` verwendet werden. Mit `initialize-efi-partition` wird ein GRUB-EFI-Lader installiert, um den GRUB-Bootloader von der Wurzelpartition zu starten. Mit `initialize-root-partition` wird ein vollständiges System instanziiert, entsprechend der Betriebssystemdefinition in `%simple-os`.

Jetzt können Sie das hier ausführen:

```
guix system image my-image.scm
```

und die Abbilddefinition wird instanziiert. Das bedeutet, ein Disk-Image wird erstellt, das die erwartete Struktur hat:

```
$ parted $(guix system image my-image.scm) print
...
Modell:  (file)
Festplatte /gnu/store/yhylv1bp5b2ypb97pd3bbhz6jk5nbhxw-disk-image: 1714MB
Sektorgröße (logisch/physisch): 512B/512B
Partitionstabelle: msdos
Disk-Flags:

Nummer  Anfang  Ende    Größe   Typ      Dateisystem  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
```

Als Größe der `boot`-Partition wurde `1619MB` ermittelt, damit sie groß genug ist, um das Betriebssystem `%simple-os` zu beherbergen.

Es ist auch möglich, dass Sie die bestehenden Definitionen für `image`-Verbundsobjekte verwenden und so `image`-Verbundsobjekte einfacher definieren können, indem Sie Vererbung benutzen. Im Modul `(gnu system image)` werden die folgenden `image`-Variablen definiert.

`efi-disk-image` [Scheme-Variable]
: Ein MBR-formatiertes Disk-Image, das aus zwei Partitionen besteht: einer ESP-Partition für 64-Bit und einer bootfähigen Wurzelpartition. Das Abbild ist sowohl für die meisten `x86_64`- als auch `i686`-Maschinen geeignet, die über BIOS oder UEFI starten können.

`efi32-disk-image` [Scheme-Variable]
: Genau wie `efi-disk-image`, aber die EFI-Partition ist auf 32 Bit ausgelegt.

`iso9660-image` [Scheme-Variable]
: Ein ISO-9660-Abbild, das eine einzelne bootfähige Partition enthält. Dieses Abbild kann ebenso auf den meisten `x86_64`- und `i686`-Maschinen benutzt werden.

`docker-image` [Scheme-Variable]
: Ein Docker-Abbild, mit dem ein Docker-Container gestartet werden kann.

Mit Hilfe des `efi-disk-image` können wir unsere vorherige `image`-Deklaration vereinfachen:

```
(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)))))
```

Dadurch kommt genau dieselbe Abbildinstanz zustande, dennoch ist die `image`-Deklaration so einfacher geworden.

16.3 „image-type“-Referenz

Dem Befehl `guix system image` kann, wie oben gezeigt, eine Datei als Argument übergeben werden, in der eine `image`-Deklaration enthalten ist, und daraus wird das eigentliche Disk-Image erstellt. An den gleichen Befehl kann man auch eine Datei als Argument übergeben, in der eine Betriebssystemdeklaration als `operating-system`-Verbundsobjekt enthalten ist. Doch wie wird in diesem Fall aus dem `operating-system` ein Abbild gemacht?

Hier kommen die Objekte des Verbundstyps `image-type` ins Spiel. Solche Verbundsobjekte legen fest, wie ein `operating-system`-Verbundsobjekt umgewandelt werden muss, damit ein `image`-Verbundsobjekt daraus wird.

`image-type` [Datentyp]

Dieser Datentyp steht für einen Abbildtyp.

`name` Der Name des Abbildtyps als Symbol. Hierfür *muss* etwas angegeben werden, zum Beispiel `'efi32-raw`.

`constructor`
Der Konstruktor für diesen Abbildtyp. Hier *muss* eine Prozedur angegeben werden, die ein Verbundsobjekt vom Typ `operating-system` nimmt und dafür ein Verbundsobjekt vom Typ `image` zurückliefert.

Es werden im Modul `(gnu system image)` und in den Modulen `(gnu system images ...)` mehrere `image-type`-Verbundsobjekte zur Verfügung gestellt.

`efi-raw-image-type` [Scheme-Variable]
Ein Abbild erstellen, das auf dem Abbild `efi-disk-image` basiert.

`efi32-raw-image-type` [Scheme-Variable]
Ein Abbild erstellen, das auf dem Abbild `efi32-disk-image` basiert.

`qcow2-image-type` [Scheme-Variable]
Ein Abbild erstellen, das auf dem Abbild `efi-disk-image` basiert, aber mit dem Abbildformat `compressed-qcow2`.

`iso-image-type` [Scheme-Variable]
Ein komprimiertes Abbild erstellen, das auf dem Abbild `iso9660-image` basiert.

`uncompressed-iso-image-type` [Scheme-Variable]
Ein Abbild erstellen, das auf dem Abbild `iso9660-image` basiert, für das allerdings das Feld `compression?` auf `#false` gesetzt ist.

`docker-image-type` [Scheme-Variable]
Ein Abbild erstellen, das auf dem Abbild `docker-image` basiert.

`raw-with-offset-image-type` [Scheme-Variable]
Ein MBR-formatiertes Abbild erstellen, das eine einzelne Partition mit einem Versatz von `1024KiB` enthält. Das ist dafür gedacht, dass noch etwas Platz verbleibt, um einen Bootloader in der Lücke nach dem MBR zu installieren.

`pinebook-pro-image-type` [Scheme-Variable]
Ein Abbild erstellen, das für eine Pinebook-Pro-Maschine als Ziel gedacht ist. Das MBR-formatierte Abbild enthält eine einzelne Partition mit einem Versatz von `9MiB`. In diese Lücke wird `u-boot-pinebook-pro-rk3399-bootloader` als Bootloader installiert.

`rock64-image-type` [Scheme-Variable]
Ein Abbild erstellen, das für eine Rock64-Maschine als Ziel gedacht ist. Das MBR-formatierte Abbild enthält eine einzelne Partition mit einem Versatz von `16MiB`. In diese Lücke wird `u-boot-rock64-rk3328-bootloader` als Bootloader installiert.

`novena-image-type` [Scheme-Variable]
Ein Abbild erstellen, das für eine Novena-Maschine als Ziel gedacht ist. Die Eigenschaften sind wie bei `raw-with-offset-image-type`.

`pine64-image-type` [Scheme-Variable]
: Ein Abbild erstellen, das für eine Pine64-Maschine als Ziel gedacht ist. Die Eigenschaften sind wie bei `raw-with-offset-image-type`.

`hurd-image-type` [Scheme-Variable]
: Ein Abbild erstellen, das für eine `i386`-Maschine als Ziel gedacht ist, auf der der Kernel von Hurd läuft. Das MBR-formatierte Abbild enthält eine einzelne ext2-Partition mit geeigneten Flags in `file-system-options`.

`hurd-qcow2-image-type` [Scheme-Variable]
: Ein Abbild erstellen, das dem mit `hurd-image-type` erstellten ähnelt, wo aber das `format` auf `'compressed-qcow2` gesetzt ist.

`wsl2-image-type` [Scheme-Variable]
: Ein Abbild für WSL2 (Windows-Subsystem für Linux 2) erstellen. Dort können Sie es importieren mit den Befehlen:

```
wsl --import Guix ./guix ./wsl2-image.tar.gz
wsl -d Guix
```

Erinnern wir uns zurück, dass der Befehl `guix system image` eine Betriebssystemdeklaration nur mit `operating-system` als Argument akzeptiert. Die Vorgabe ist, dass dann `efi-raw-image-type` verwendet wird, um aus dem Betriebssystem im `operating-system`-Objekt ein wirklich bootfähiges Abbild zu machen.

Sie können ein anderes `image-type`-Objekt als Abbildtyp auswählen, indem Sie die Befehlszeilenoption `--image-type` angeben. Mit der Befehlszeilenoption `--list-image-types` werden Ihnen alle unterstützten Abbildtypen angezeigt, was auf eine textuelle Auflistung der oben beschriebenen `image-types`-Variablen hinausläuft (siehe Abschnitt 12.15 [Aufruf von guix system], Seite 619).

16.4 Abbild-Module

Nehmen wir uns das Pine64 als Beispiel vor. Es handelt sich um eine ARM-basierte Maschine. Um ein Abbild für diesen Chip anfertigen zu können, benötigen wir die folgenden Bestandteile:

- Ein `operating-system`-Verbundsobjekt, zu dem mindestens ein geeigneter Kernel (`linux-libre-arm64-generic`) und Bootloader `u-boot-pine64-lts-bootloader`) für Pine64 gehören.
- Womöglich wäre ein `image-type`-Verbundsobjekt geeignet, mit dem aus einem `operating-system`-Verbundsobjekt ein für Pine64 nutzbares `image`-Verbundsobjekt hergeleitet wird.
- Dazu ein `image`-Verbundsobjekt, das über den Befehl `guix system image` instanziiert werden kann.

Im Modul `(gnu system images pine64)` sind all diese Bestandteile zu finden; jeweils `pine64-barebones-os`, `pine64-image-type` und `pine64-barebones-raw-image`.

Das Modul liefert ein Abbild vom Typ `pine64-barebones-raw-image`, so dass wir diesen Befehl benutzen können:

```
guix system image gnu/system/images/pine64.scm
```

Damit könnten wir dank dem im `pine64-image-type`-Verbundsobjekt deklarierten Abbildtyp `'pine64-raw` eine Datei `my-pine.scm` vorbereiten, die dann den folgenden Inhalt hat:

```
(use-modules (gnu system images pine64))
(operating-system
  (inherit pine64-barebones-os)
  (timezone "Europe/Athens"))
```

So passen wir das Betriebssystem aus `pine64-barebones-os` an. Wir führen aus:

```
$ guix system image --image-type=pine64-raw my-pine.scm
```

Es sei angemerkt, dass Sie genauso in anderen Modulen innerhalb des Verzeichnisses `gnu/system/images` Definitionen für Novena-, Pine64-, PinebookPro- und Rock64-Maschinen als Zielsystem finden können.

17 Dateien zur Fehlersuche installieren

Die Binärdateien von Programmen, wie sie zum Beispiel von den GCC-Compilern erzeugt werden, sind in der Regel im ELF-Format gespeichert und enthalten eine Sektion mit *Informationen zur Fehlersuche* (englisch „Debugging Information“). Informationen zur Fehlersuche machen es möglich, dass der Debugger, GDB, Binärcode dem Quellcode zuordnen kann. Das ist nötig, damit es mit etwas Glück leicht ist, Fehler in einem kompilierten Programm zu suchen.

Dieses Kapitel erklärt, wie abgetrennte Informationen zur Fehlersuche („Debug“-Informationen) verwendet werden können, wenn Pakete solche anbieten, und wie Pakete mit Informationen zur Fehlersuche neu erstellt werden können, wenn die Pakete keine anbieten.

17.1 Fehlersuchinformationen abtrennen

Das Problem bei Informationen zur Fehlersuche ist, dass dadurch einiges an Plattenplatz verbraucht wird. Zum Beispiel steuern die Informationen zur Fehlersuche in der GNU-C-Bibliothek mehr als 60 MiB bei. Als ein Nutzer ist es deswegen in der Regel nicht möglich, sämtliche Fehlersuchinformationen für alle installierten Programme zu speichern. Andererseits sollten Platzeinsparnisse nicht auf Kosten der Fehlersuche gehen – besonders im GNU-System, wo es Nutzern leicht fallen sollte, ihre Freiheit, wie sie ihre Rechner benutzen, auszuüben (siehe Abschnitt 1.2 [GNU-Distribution], Seite 2).

Glücklicherweise gibt es in den GNU Binary Utilities (Binutils) und GDB einen Mechanismus, mit dem Nutzer das Beste aus beiden Welten bekommen: Informationen zur Fehlersuche können von den davon beschriebenen Binärdateien losgelöst und in separaten Dateien gespeichert werden. GDB kann dann Fehlersuchinformationen laden, wenn diese Dateien verfügbar sind (siehe Abschnitt “Separate Debug Files” in *Debugging with GDB*).

Die GNU-Distribution nutzt diesen Mechanismus aus, indem sie Informationen zur Fehlersuche im Unterverzeichnis `lib/debug` einer separaten Paketausgabe speichert, die den fantasielosen Namen `debug` trägt. Mit dem folgenden Befehl können Sie zum Beispiel Informationen zur Fehlersuche für die GNU-C-Bibliothek und für GNU Guile installieren:

```
guix install glibc:debug guile:debug
```

GDB muss dann angewiesen werden, im Profil des Nutzers nach Informationen zur Fehlersuche zu schauen, indem Sie die Variable `debug-file-directory` entsprechend setzen (vielleicht möchsten Sie die Variable in der Datei `~/.gdbinit` festlegen, siehe Abschnitt “Startup” in *Debugging with GDB*):

```
(gdb) set debug-file-directory ~/.guix-profile/lib/debug
```

Von da an wird GDB auch aus den `.debug`-Dateien unter `~/.guix-profile/lib/debug` auslesbare Informationen zur Fehlersuche verwenden.

Im Folgenden sehen Sie ein alternatives GDB-Skript, um mit anderen Profilen zu arbeiten. Es macht Gebrauch von der optionalen Guile-Einbindung in GDB. Dieses Schnipsel wird auf Guix System nach Voreinstellung in der `~/.gdbinit`-Datei zur Verfügung gestellt.

```
guile
(use-modules (gdb))
(execute (string-append "set debug-file-directory "
```

```
                        (or (getenv "GDB_DEBUG_FILE_DIRECTORY")
                            "~/.guix-profile/lib/debug")))
end
```

Des Weiteren werden Sie höchstwahrscheinlich wollen, dass GDB den Quellcode, der auf Fehler untersucht wird, anzeigen kann. Dazu müssen sie den Quellcodes des Pakets, für das Sie sich interessieren (laden Sie ihn mit `guix build --source` herunter; siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189), und dann weisen Sie GDB an, in dem Verzeichnis zu suchen, indem Sie den `directory`-Befehl benutzen (siehe Abschnitt "Source Path" in *Debugging with GDB*).

Der Mechanismus mit der `debug`-Ausgabe wird in Guix als Teil des `gnu-build-system` implementiert (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130). Zurzeit ist sie optional – nur für Pakete, in deren Definition ausdrücklich eine `debug`-Ausgabe deklariert wurde, sind Informationen zur Fehlersuche verfügbar. Um zu überprüfen, ob Pakete eine `debug`-Ausgabe mit Informationen zur Fehlersuche haben, benutzen Sie `guix package --list-available` (siehe Abschnitt 6.2 [Aufruf von guix package], Seite 45).

Lesen Sie weiter, um zu erfahren, wie Sie mit Paketen ohne eine `debug`-Ausgabe umgehen können.

17.2 Fehlersuchinformationen erneuern

Wie wir oben gesehen haben, bieten manche Pakete, aber nicht alle, Informationen zur Fehlersuche in einer `debug`-Ausgabe an. Doch was tut man mit denen ohne Fehlersuchinformationen? Die Befehlszeilenoption `--with-debug-info` stellt eine Lösung dafür da: Mit ihr kann jedes Paket, für das solche Debug-Informationen fehlen, – und nur solche – neu erstellt werden und die Anwendung, in der Sie Fehler suchen, wird damit veredelt. Obwohl es also nicht so schnell geht wie eine `debug`-Ausgabe zu installieren, ist es doch ein verhältnismäßig kleiner Aufwand.

Schauen wir uns das näher an. Angenommen, Sie erleben einen Fehler in Inkscape und würden gerne wissen, was dabei in GLib passiert. GLib ist eine Bibliothek, die tief im Abhängigkeitsgraphen von GLib liegt. Wie sich herausstellt, verfügt GLib über keine `debug`-Ausgabe. Die Rückverfolgung („Backtrace"), die GDB anzeigt, ist zu nichts nütze:

```
(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
```

Dagegen hilft, wenn Sie Inkscape so installieren, dass es an eine Variante von GLib gebunden ist, die Informationen zur Fehlersuche enthält.

```
guix install inkscape --with-debug-info=glib
```

Und schon sieht die Fehlersuche wesentlich besser aus:

```
$ 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
...
```

Viel besser!

Beachten Sie, dass es Pakete geben kann, für die sich `--with-debug-info` nicht wie gewünscht auswirkt. Siehe Abschnitt 10.1.2 [Paketumwandlungsoptionen], Seite 192, für mehr Informationen.

18 TEX und LATEX gebrauchen

Guix stellt Pakete für TEX, LATEX, ConTeXt, LuaTeX und verwandte Textsatzsysteme bereit, die aus der TEX-Live-Distribution (`https://www.tug.org/texlive/`) stammen. Weil TEX Live aber außergewöhnlich groß ist und der Umgang in diesem Labyrinth schwierig ist, möchten wir unseren verehrten Nutzern eine Anleitung mitgeben, wie man die nötigen Pakete aufspielt und damit TEX- und LATEX-Dokumente kompiliert.

TEX Live gibt es in zwei Geschmacksrichtungen in Guix:

- Einmal das „monolithische“ `texlive`-Paket: Da ist *absolut jedes TEX-Live-Paket* drinnen (über 7.000 Stück), es ist aber auch gewaltig groß (über 4 GiB für nur ein Paket!).
- Dann die „modularen“ Pakete mit `texlive-` im Namen: Sie installieren `texlive-base`, das die Kernfunktionen und hauptsächlichen Befehle verfügbar macht – also `pdflatex`, `dvips`, `luatex`, `mf` und so – und dazu installieren Sie diejenigen eigenständigen Pakete mit den Funktionen, die Sie brauchen – das wären `texlive-listings` für das `listings`-Paket oder `texlive-hyperref` für `hyperref`, `texlive-beamer` für Beamer, `texlive-pgf` für PGF/TikZ und so weiter.

Wir raten zum modularen Paketsatz, weil er weniger Ressourcen in Anspruch nimmt. Ihre Dokumente erstellen Sie dann mit solchen Befehlen:

```
guix shell texlive-base texlive-wrapfig \
  texlive-hyperref texlive-cm-super -- pdflatex dokument.tex
```

Die Befehlszeilen werden aber bald unangemessen lang. Das Problem lösen Sie mit einem Manifest, in dem etwas wie hier steht:

```
(specifications->manifest
 '("rubber"

   "texlive-base"
   "texlive-wrapfig"

   "texlive-microtype"
   "texlive-listings" "texlive-hyperref"

   ;; PGF/TikZ
   "texlive-pgf"

   ;; Zusätzliche Schriftarten.
   "texlive-cm-super" "texlive-amsfonts"
   "texlive-times" "texlive-helvetic" "texlive-courier"))
```

Das übergeben Sie dann einem beliebigen Befehl mit der Befehlszeilenoption `-m`:

```
guix shell -m manifest.scm -- pdflatex dokument.tex
```

Siehe Abschnitt 9.4 [Manifeste verfassen], Seite 125, wenn Sie mehr über Manifeste lesen möchten. In Zukunft haben wir vor, dass es Paketsammlungen für TEX Live geben wird – „Meta-Pakete“ wie `fontsrecommended`, `humanities` oder `langarabic`, die die Pakete umfassen, die Sie in diesem bestimmten Bereich brauchen. Dadurch müssten Sie weniger Pakete auflisten.

Schwierigkeiten macht beim modularen Paketsatz vor allem, dass Sie gezwungen sind, genau die Pakete auszusuchen, die Sie brauchen. Klar können Sie `guix search` verwenden, aber das richtige Paket zu finden, stellt sich hin und wieder als mühsam heraus. Wenn ein Paket fehlt, schlagen Befehle wie `pdflatex` und Kollegen fehl mit so unverständlichen Meldungen wie:

```
dokument.tex: File `tikz.sty' not found.
dokument.tex:7: Emergency stop.
```

Oder wenn eine Schrift fehlt:

```
kpathsea: Running mktexmf phvr7t
! I can't find file `phvr7t'.
```

Wie findet man heraus, welches Paket fehlt? Im ersten Fall hilft eine Suche:

```
$ guix search texlive tikz
name: texlive-pgf
version: 59745
...
```

Im zweiten Fall wird man mit `guix search` nicht fündig. Wenn man stattdessen in der Paketdatenbank von TEX Live mit dem Befehl `tlmgr` sucht, bekommt man Antworten:

```
$ guix shell texlive-base -- 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
```

Die Datei erhält man als Teil von TEX Lives `helvetic`-Paket, das in Guix `texlive-helvetic` heißt. Es sind viele Schritte, aber sie führen zum Ziel!

Die Sache hat allerdings eine wichtige Einschränkung: Guix stellt bislang eine Teilmenge der TEX-Live-Pakete zur Verfügung. Wenn Ihnen ein fehlendes Paket unterkommt, versuchen Sie, es zu importieren (siehe Abschnitt 10.5 [Aufruf von guix import], Seite 206):

```
guix import texlive Paket
```

Zu den zusätzlichen Optionen gehören:

`--recursive`
`-r`
: Den Abhängigkeitsgraphen des angegebenen Pakets beim Anbieter rekursiv durchlaufen und Paketausdrücke für alle solchen Pakete erzeugen, die es in Guix noch nicht gibt.

  Anmerkung: Pakete für TEX Live zu schreiben, wird noch etwas Zeit beanspruchen, aber Sie können uns dabei helfen! Siehe Kapitel 22 [Mitwirken], Seite 708, für mehr Informationen.

19 Sicherheitsaktualisierungen

Ab und zu werden wichtige Sicherheitsschwachstellen in Software-Paketen entdeckt, die mit Patches behoben werden müssen. Guix-Entwickler geben ihr Bestes, bezüglich bekannter Schwachstellen auf dem Laufenden zu bleiben und so bald wie möglich Patches dafür auf den `master`-Branch von Guix aufzuspielen (einen stabilen „stable“-Branch ohne riskante Änderungen haben wir noch nicht). Das Werkzeug `guix lint` hilft Entwicklern dabei, verwundbare Versionen von Softwarepaketen in der Distribution zu finden:

```
$ guix lint -c cve
gnu/packages/base.scm:652:2: glibc@2.21: Wahrscheinlich angreifbar durch CVE-2015-1781, CVE-2015-7547
gnu/packages/gcc.scm:334:2: gcc@4.9.3: Wahrscheinlich angreifbar durch CVE-2015-5276
gnu/packages/image.scm:312:2: openjpeg@2.1.0: Wahrscheinlich angreifbar durch CVE-2016-1923, CVE-2016-192
...
```

Siehe Abschnitt 10.8 [Aufruf von guix lint], Seite 223, für weitere Informationen.

Guix verfolgt eine funktionale Disziplin bei der Paketverwaltung (siehe Kapitel 1 [Einführung], Seite 1), was impliziert, dass bei jeder Änderung an einem Paket *jedes davon abhängige Paket* neu erstellt werden muss. Ohne einen Mechanismus würde das Ausliefern von Sicherheitsaktualisierungen in Kernpaketen wie libc oder Bash dadurch deutlich verlangsamt – schließlich müsste quasi die gesamte Distribution neu erstellt werden. Vorerstellte Binärdateien zu benutzen, wäre schon einmal eine Hilfe (siehe Abschnitt 6.3 [Substitute], Seite 56), aber die Auslieferung wäre immer noch laangsamer, als wir es uns wünschen.

Als Gegenmittel sind in Guix *Veredelungen* implementiert. Diese stellen einen Mechanismus dar, mit dem kritische Aktualisierungen schnell an Guix' Benutzer ausgeliefert werden können, ohne die Nachteile, zu denen es käme, wenn wir die gesamte Distribution neu erstellen müssten. Die Idee dahinter ist, nur das Paket, das einen Patch braucht, neu zu erstellen, und damit dann Pakete, die der Nutzer ausdrücklich installiert hat und die vorher Referenzen auf das alte Paket enthielten, zu „veredeln“. So eine Veredelung kostet typischerweise nur sehr wenig, d.h. um Größenordnungen weniger, als die ganze Abhängigkeitskette neu zu erstellen.

Nehmen wir also an, eine Sicherheitsaktualisierung müsste auf Bash angewandt werden. Guix-Entwickler schreiben dann eine Paketdefinition für die „reparierte“ Bash, sagen wir `bash-fixed`, auf die gleiche Art wie immer (siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Dann wird die ursprüngliche Paketdefinition um ein `replacement`-Feld (zu Deutsch „Ersetzung“) erweitert, das auf das Paket verweist, in dem der Fehler behoben wurde:

```
(define bash
  (package
    (name "bash")
    ;; ...
    (replacement bash-fixed)))
```

Ab diesem Zeitpunkt wird jedes Paket, das Sie installieren und das direkt oder indirekt von Bash abhängt – also die von `guix gc --requisites` ausgegebenen Pakete (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61) –, automatisch „umgeschrieben“, so dass es `bash-fixed` referenziert, wo es vorher `bash` referenziert hatte. Die Dauer dieses Veredelungsprozesses ist proportional zur Größe des Pakets und liegt auf einer neuen Maschine für ein „durchschnittliches“ Paket bei unter einer Minute. Veredeln ist rekursiv: Wenn eine

indirekte Abhängigkeit veredelt werden muss, „propagiert“ der Veredelungsprozess durch die abhängigen Pakete und endet mit dem Paket, das der Nutzer installiert.

Zurzeit muss der Name und die Version einer Veredelung gleichlang wie die beim ersetzten Paket sein (also bei `bash-fixed` und `bash` im Beispiel oben). Diese Einschränkung kommt daher, dass beim Veredeln der Inhalt von Dateien, einschließlich Binärdateien, durch einfache Ersetzungen „geflickt“ wird. Es gibt noch mehr Einschränkungen: Wenn zum Beispiel ein Paket veredelt wird, das eine gemeinsame Bibliothek („Shared Library“) verwendet, muss der `SONAME` von Original und Ersatz derselbe sein und die beiden müssen binär kompatibel sein.

Mit der Befehlszeilenoption `--no-grafts` können Sie den Veredelungsmechanismus zwingend abschalten (siehe Abschnitt 10.1.1 [Gemeinsame Erstellungsoptionen], Seite 189). Der Befehl

```
guix build bash --no-grafts
```

liefert also den Namen der Store-Datei mit der ursprünglichen Bash, während

```
guix build bash
```

den Namen der Store-Datei für die „reparierte“ Ersatz-Bash liefert. Dadurch können Sie zwischen den beiden Varianten von Bash unterscheiden.

Um zu prüfen, welche Bash Ihr gesamtes Profil referenziert, können Sie diesen Befehl hier laufen lassen (siehe Abschnitt 6.5 [Aufruf von guix gc], Seite 61):

```
guix gc -R $(readlink -f ~/.guix-profile) | grep bash
```

Dann vergleichen Sie die Namen der Store-Objekte, die Sie ausgegeben bekommen, mit den beiden Bash-Paketnamen oben. Ebenso können Sie eine ganze Guix-Systemgeneration überprüfen:

```
guix gc -R $(guix system build my-config.scm) | grep bash
```

Zum Schluss können Sie mit dem Befehl `lsof` überprüfen, welches von den Bash-Paketen die laufenden Prozesse benutzen:

```
lsof | grep /gnu/store/.*bash
```

20 Bootstrapping

Wenn wir von Bootstrapping sprechen, meinen wir damit, wie die Distribution „aus dem Nichts“ erstellt werden kann. Erinnern Sie sich, wie die Erstellungsumgebung für eine Ableitung nichts außer ihren deklarierten Eingaben enthält (siehe Kapitel 1 [Einführung], Seite 1)? Daraus ergibt sich ein Henne-Ei-Problem: Wie kann so das allererste Paket entstehen? Womit wird der Compiler kompiliert?

Man könnte auf die Idee kommen, diese Frage sei nur für eingefleischte Hacker interessant. Doch auch wenn die Antwort darauf technischer Natur ist, hat sie weitreichende Implikationen. Vom Bootstrapping der Distribution hängt ab, wie sehr wir, als Individuen und als Kollektiv aus Nutzern und Hackern, der Software vertrauen können, die wir verwenden. Es ist ein zentrales Anliegen vom Standpunkt der *Informationssicherheit*, aber auch im Hinblick auf die *Freiheit* der Benutzer.

Das GNU-System besteht in erster Linie aus C-Code, dessen Kern die libc ist. Das GNU-Erstellungssystem selbst setzt voraus, dass eine Bourne-Shell und die Kommandozeilenwerkzeuge der GNU-Coreutils, Awk, Findutils, „sed“ und „grep“ verfügbar sind. Des Weiteren sind Programme für die Erstellung – also Programme, die `./configure`, `make`, etc. ausführen – in Guile Scheme geschrieben (siehe Abschnitt 9.10 [Ableitungen], Seite 167). Folglich ist es erforderlich, dass, damit überhaupt irgendetwas erstellt werden kann, Guix vorerstellte Binärdateien von Guile, GCC, Binutils, libc und den anderen oben genannten Paketen verwendet. Diese bezeichnen wir als die *Bootstrap-Binärdateien*.

Diese Bootstrap-Binärdateien werden als „gegeben“ angenommen, obwohl wir sie auch neu erzeugen können, falls nötig (siehe Abschnitt 20.2 [Vorbereitung zur Verwendung der Bootstrap-Binärdateien], Seite 702).

20.1 Das Bootstrapping mit kleinerem Seed

Guix wird – wie andere GNU/Linux-Distributionen auch – traditionell aus einer Menge von Bootstrap-Binärdateien heraus erstellt: der Bourne-Shell, den Befehlszeilenwerkzeugen der GNU Coreutils, Awk, Findutils, „sed“ und „grep“ sowie Guile, GCC, Binutils und der GNU-C-Bibliothek (siehe Kapitel 20 [Bootstrapping], Seite 699). Normalerweise werden diese Bootstrap-Binärdateien „stillschweigend vorausgesetzt“.

Die Bootstrap-Binärdateien stillschweigend vorauszusetzen bedeutet, dass wir sie als korrekte und vertrauenswürdige Grundlage ansehen, als „Seed“, aus dem heraus das komplette System erstellt wird. Darin liegt ein Problem: Die Gesamtgröße all dieser Bootstrapping-Binärdateien beträgt um die 250MB (siehe Abschnitt “Bootstrappable Builds” in *GNU Mes*). Ein Audit oder auch nur eine Inspektion davon ist praktisch unmöglich.

Für `i686-linux` und `x86_64-linux` unterstützt Guix jetzt ein Bootstrapping „mit kleinerem Seed“[1].

Beim Bootstrapping mit kleinerem Seed gehören die kritischsten Werkzeuge – was Vertrauenswürdigkeit angeht – nicht mehr zu den Bootstrapping-Binärdateien: Anstelle von GCC, Binutils und der GNU-C-Bibliothek bleiben nur `bootstrap-mescc-tools` (ein winziger Assembler und Binder) und `bootstrap-mes` (ein kleiner Scheme-Interpretierer sowie ein

[1] Gerne würden wir „Bootstrapping aus dem Quellcode allein“ sagen und wir arbeiten auch daran, aber das schon jetzt zu sagen, wäre eine Übertreibung.

C-Compiler, der in Scheme geschrieben wurde, und die Mes-C-Bibliothek, mit der TinyCC und GCC erstellt werden können).

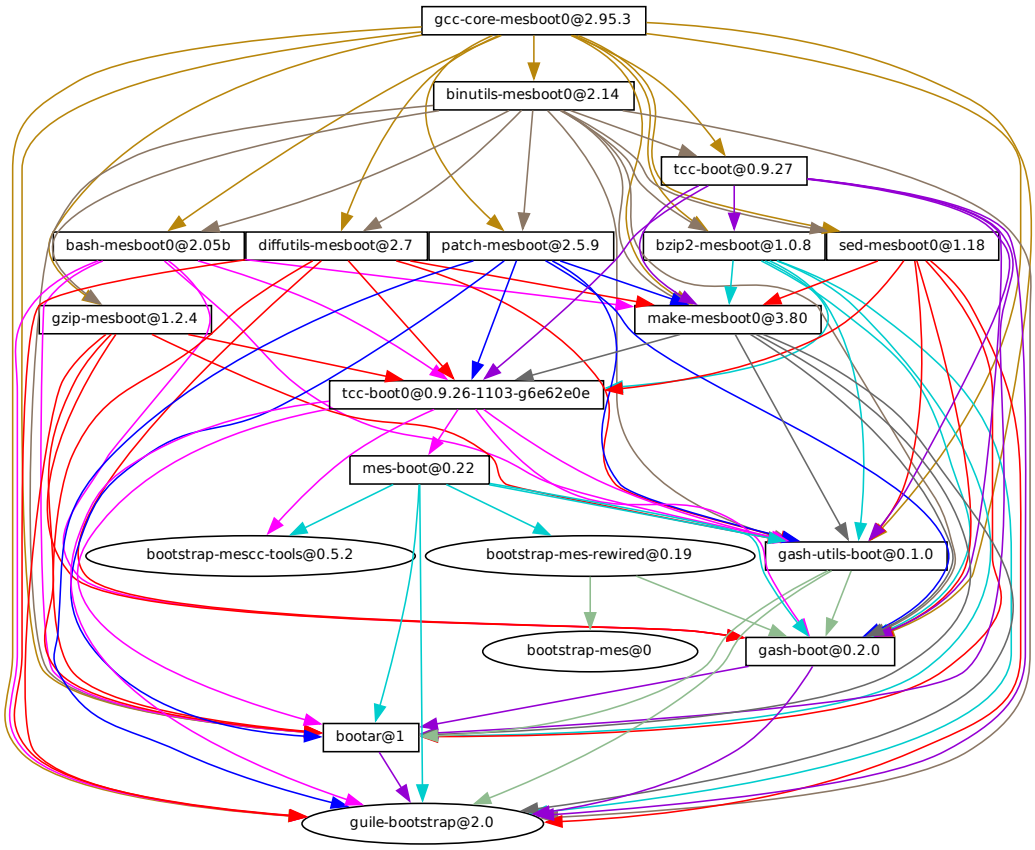
Mit Hilfe dieser neuen Seed-Binärdateien werden die „fehlenden“ Binutils, GCC und die GNU-C-Bibliothek aus dem Quellcode heraus erstellt. Von hier an kann der traditionellere Bootstrapping-Vorgang wie gewohnt weiterlaufen. Durch diesen Ansatz machen die kleineren Bootstrapping-Binärdateien in Guix-Version 1.1 nur noch ungefähr 145MB aus.

Der nächste Schritt war es, die Shell und all ihre Werkzeuge durch in Guile Scheme verfasste Implementierungen zu ersetzen. Nun haben wir ein *Bootstrapping nur in Scheme*. Gash (siehe Abschnitt “Gash” in *The Gash manual*) ist eine POSIX-kompatible Shell, die Bash ersetzt, und mit ihr kommen die Gash Utils als minimalistischer Ersatz für Awk, die GNU Core Utilities, Grep, Gzip, Sed und Tar. Die übrigen Binärdateien unter den Bootstrapping-Seeds wurden entfernt und werden jetzt aus ihrem Quellcode heraus erstellt.

Das Erstellen des GNU-Systems aus seinem Quellcode heraus ist derzeit nur möglich, wenn wir ein paar historische GNU-Pakete als Zwischenschritte hinzufügen[2]. Mit dem Heranreifen von Gash und Gash Utils und der Entwicklung von GNU-Paketen hin zu mehr Bootstrapbarkeit (z.B. wird es neue Veröffentlichungen von GNU Sed auch wieder als Gzip-komprimierte Tarballs geben, einer Alternative zur schwer zu bootstrappenden `xz`-Kompression), wird dieser Satz zusätzlicher Pakete hoffentlich noch einmal reduziert werden können.

Im folgenden Graphen sehen Sie den sich ergebenden Abhängigkeitsgraphen für `gcc-core-mesboot0`, den Bootstrapping-Compiler, mit dem das traditionelle Bootstrapping für den Rest von Guix System vollzogen wird.

[2] Dazu gehören Pakete wie `gcc-2.95.3`, `binutils-2.14`, `glibc-2.2.5`, `gzip-1.2.4`, `tar-1.22` und noch ein paar andere. Details finden Sie in `gnu/packages/commencement.scm`.



Die einzig bedeutsamen verbleibenden Binärdateien unter den Bootstrapping-Seeds[3] sind ein Scheme-Intepretierer und ein Scheme-Compiler: GNU Mes und GNU Guile[4].

Nach dieser weiteren Reduktion macht der binäre Seed nur noch ungefähr 60MB aus für `i686-linux` und `x86_64-linux`.

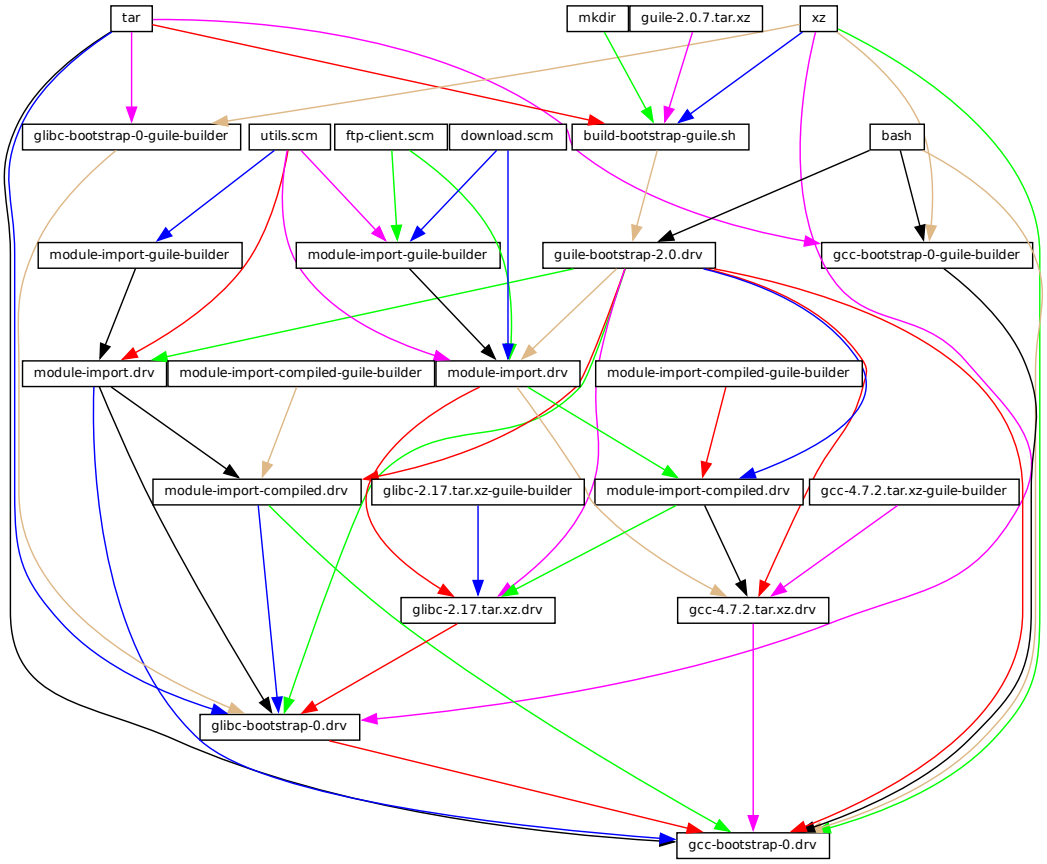
Wir arbeiten daran, alle binären „Blobs“ aus unserem Freie-Software-Bootstrapping zu entfernen, um nur aus Quellcode heraus bootstrappen zu können („Full Source Bootstrap“). Auch ist Arbeit im Gange, ein solches Bootstrapping für die `arm-linux-` und `aarch64-linux`-Architekturen und GNU Hurd anzubieten.

Wenn Sie daran Interesse haben, dann machen Sie bei uns mit auf `#bootstrappable` auf dem Freenode-IRC-Netzwerk oder diskutieren Sie mit auf `bug-mes@gnu.org` oder `gash-devel@nongnu.org`.

[3] Wenn wir die 68KB-großen `mescc-tools` ignorieren, die wir später noch zusammen mit `mes` entfernen werden.

[4] In diesem Graphen werden die statischen Binärdateien für `bash`, `tar` und `xz` nicht gezeigt, die wir benutzen, um Guile zum Laufen zu bringen.

20.2 Vorbereitung zur Verwendung der Bootstrap-Binärdateien



Die Abbildung oben zeigt den Anfang des Abhängigkeitsgraphen der Distribution und entspricht den Paketdefinitionen im `(gnu package bootstrap)`-Modul. Eine ähnliche Grafik kann mit `guix graph` (siehe Abschnitt 10.10 [Aufruf von guix graph], Seite 228) erzeugt werden:

```
guix graph -t derivation \
  -e '(@@ (gnu packages bootstrap) %bootstrap-gcc)' \
  | dot -Tps > gcc.ps
```

oder für das Bootstrapping mit noch kleinerem Seed:

```
guix graph -t derivation \
  -e '(@@ (gnu packages bootstrap) %bootstrap-mes)' \
  | dot -Tps > mes.ps
```

Bei diesem Detaillierungsgrad sind die Dinge recht komplex. Guile selbst besteht aus einer ausführbaren ELF-Datei neben vielen Quelldateien und kompilierten Scheme-Dateien, die dynamisch bei der Ausführung geladen werden. Das wird in dem im Graph gezeigten

`guile-2.0.7.tar.xz`-Archiv gespeichert. Das Archiv ist Teil von Guix' „Quelldistribution“ und wird in den Store mit `add-to-store` (siehe Abschnitt 9.9 [Der Store], Seite 165) eingefügt.

Doch wie schreibt man eine Ableitung, die dieses Tarball-Archiv entpackt und in den Store einfügt? Um dieses Problem zu lösen, benutzt die `guile-bootstrap-2.0.drv`-Ableitung – die erste, die erstellt wird – `bash` als Ersteller, welche wiederum `build-bootstrap-guile.sh` ausführt, was über einen Aufruf von `tar` den Tarball entpackt. Deswegen sind `bash`, `tar`, `xz` und `mkdir` als statisch gebundene Binärdateien auch Teil der Guix-Quelldistribution, die nur dazu da sind, dass der Guile-Tarball entpackt werden kann.

Sobald `guile-bootstrap-2.0.drv` erstellt worden ist, haben wir ein funktionierendes Guile zur Hand, mit dem nachfolgende Erstellungsprogramme ausgeführt werden können. Sein erster Auftrag ist, Tarballs mit den anderen vorerstellten Binärdateien herunterzuladen – das ist die Tätigkeit der `.tar.xz.drv`-Ableitungen. Wir verwenden zu diesem Zweck Guix-Module wie `ftp-client.scm`. Die `module-import.drv`-Ableitungen importieren solche Module und schreiben sie in derselben Verzeichnisstruktur in ein Verzeichnis im Store. Die `module-import-compiled.drv`-Ableitungen kompilieren die Module und schreiben sie in der richtigen Struktur in ein Ausgabeverzeichnis. Dies entspricht dem `#:modules`-Argument von `build-expression->derivation` (siehe Abschnitt 9.10 [Ableitungen], Seite 167).

Schließlich werden die verschiedenen Tarballs durch die Ableitungen `gcc-bootstrap-0.drv`, `glibc-bootstrap-0.drv`, oder `bootstrap-mes-0.drv` und `bootstrap-mescc-tools-0.drv`, entpackt. Zu diesem Zeitpunkt haben wir eine fertige Toolchain für C.

Die Erstellungswerkzeuge erstellen

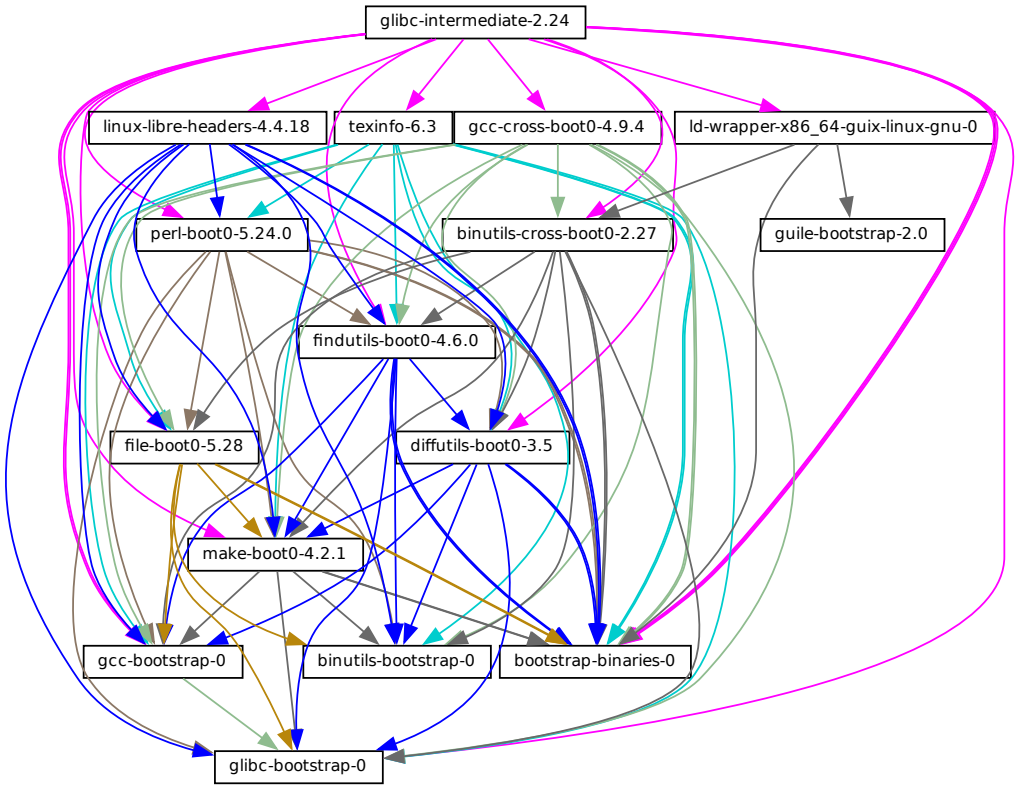
Das Bootstrapping ist abgeschlossen, sobald eine vollständige Toolchain vorliegt, die von den oben erläuterten vorerstellten Bootstrapping-Werkzeugen *nicht* abhängt. Diese Voraussetzung, keine Abhängigkeiten zu haben, überprüft man, indem man schaut, ob die Dateien der endgültigen Toolchain frei von Referenzen auf die `/gnu/store`-Verzeichnisse der Bootstrapping-Eingaben sind. Der Vorgang, diese „finale“ Toolchain zu bekommen, wird von den Paketdefinitionen beschrieben, die Sie im Modul `(gnu packages commencement)` finden.

Mit dem Befehl `guix graph` können wir gegenüber dem obigen Graphen „herauszoomen“, indem wir alles auf der Ebene von Paketobjekten statt auf der von einzelnen Ableitungen betrachten – denken Sie daran, dass ein Paket zu mehreren Ableitungen führen kann; normalerweise einer, die seine Quelldateien herunterlädt, einer, die die benötigten Guile-Module erstellt, und einer, die das Paket dann tatsächlich aus seinem Quellcode heraus erstellt. Der Befehl

```
guix graph -t bag \
  -e '(@@ (gnu packages commencement)
          glibc-final-with-bootstrap-bash)' | xdot -
```

zeigt den Abhängigkeitsgraphen, der zur „finalen“ C-Bibliothek[5] führt. Hier sehen Sie ihn:

[5] Ihnen könnte die `glibc-intermediate`-Markierung auffallen, die darauf hindeutet, dass sie *noch nicht ganz* final ist, aber annäherungsweise betrachten wir sie als final.



Das erste Werkzeug, das mit den Bootstrapping-Binärdateien erstellt wird, ist GNU Make – beachten Sie das oben sichtbare `make-boot0` –, das eine Voraussetzung aller folgenden Pakete ist. Von da aus werden Findutils und Diffutils erstellt.

Es folgt die erste Stufe der Binutils und GCC, die pseudo-crosskompiliert werden – d.h. die `--target`-Befehlszeilenoption entspricht der `--host`-Option. Mit ihnen wird libc erstellt. Dank den Crosskompilierungstricks ist garantiert, dass diese libc keine Referenzen auf die anfängliche Toolchain enthält.

Damit werden die finalen Binutils und GCC erstellt (sie sind oben nicht zu sehen). GCC benutzt den `ld` aus den finalen Binutils und bindet Programme an die gerade erstellte libc. Mit dieser Toolchain erstellen wir die anderen Pakete, die Guix und das GNU-Erstellungssystem benutzen: Guile, Bash, Coreutils, etc.

Und voilà! Wenn das geschafft ist, haben wir die vollständige Menge von Erstellungswerkzeugen, die das GNU-Erstellungssystem erwartet. Sie sind in der Variablen `%final-inputs` des Moduls `(gnu packages commencement)` zu finden und werden von jedem Paket implizit benutzt, das das `gnu-build-system` verwendet (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130).

Die Bootstrapping-Binärdateien erstellen

Weil die finale Toolchain nicht von den Bootstrapping-Binärdateien abhängt, müssen diese nur selten aktualisiert werden. Es ist dennoch sinnvoll, sie automatisiert erzeugen zu können, wenn sie doch aktualisiert werden. Das Modul `(gnu packages make-bootstrap)` ermöglicht dies.

Mit dem folgenden Befehl werden die Tarball-Archive erstellt, die die Bootstrapping-Binärdateien enthalten (beim traditionellen Bootstrapping sind das Binutils, GCC und glibc; beim Bootstrapping mit kleinerem Seed sind es linux-libre-headers, bootstrap-mescc-tools, bootstrap-mes; dazu kommen Guile sowie ein Tarball mit einer Mischung aus Coreutils und anderen grundlegenden Befehlszeilenwerkzeugen):

```
guix build bootstrap-tarballs
```

Die erzeugten Tarballs sind es, auf die im Modul `(gnu packages bootstrap)` verwiesen werden sollte, das am Anfang dieses Abschnitts erwähnt wurde.

Können Sie noch folgen? Dann haben Sie vielleicht schon angefangen, sich zu fragen, wann wir denn einen Fixpunkt erreichen. Das ist eine interessante Frage! Leider wissen wir es nicht, aber wenn Sie es herausfinden wollen (und Ihnen die nennenswerten Rechen- und Speicherkapazitäten dafür zur Verfügung stehen), dann lassen Sie es uns wissen.

Die Menge an Bootstrapping-Binärdateien verkleinern

Zu unserem traditionellen Bootstrapping gehören GCC, GNU Libc, Guile, etc. Das ist ganz schön viel binärer Code! Warum ist das ein Problem? Es ist deswegen ein Problem, weil es praktisch unmöglich ist, solch große Klumpen binären Codes einem Audit zu unterziehen. Dadurch wird es schwer, nachzuvollziehen, welcher Quellcode ihn erzeugt hat. Jede ausführbare Binärdatei, für die kein Audit möglich ist, macht uns verwundbar gegenüber Hintertüren in Compilern, wie Ken Thompson sie in seiner Arbeit von 1984, *Reflections on Trusting Trust*, beschrieben hat.

Wir senken das Risiko, indem wir unsere Bootstrapping-Binärdateien immer mit einer früheren Guix-Version erzeugen. Trotzdem fehlt uns das Niveau an Transparenz, das wir am übrigen Paketabhängigkeitsgraphen wertschätzen, wo Guix immer vom Quellcode eindeutig auf die Binärdateien abbildet. Unser Ziel ist also, die Menge an Bootstrapping-Binärdateien so weit wie möglich zu verkleinern.

Auf dem Webauftritt von Bootstrappable.org (`https://bootstrappable.org`) werden laufende Projekte mit diesem Zweck aufgeführt. Bei einem davon geht es darum, den Bootstrapping-GCC durch eine Folge von Assemblern, Interpretierern und Compilern zunehmender Komplexität zu ersetzen, die von Anfang an aus Quellcode heraus erstellt werden kann, angefangen bei einem einfachen, überprüfbaren Assembler.

Unsere erste große Leistung stellt die Ersetzung von GCC, der GNU-C-Bibliothek und der Binutils durch die MesCC-Tools (einem einfachen Binder für hexadezimal dargestellte Maschinenprogramme und einem Makro-Assembler) und Mes dar (siehe *GNU Mes*, einem Scheme-Interpretierer und in Scheme geschriebenen C-Compiler). Weder MesCC-Tools noch Mes können bereits von Grund auf gebootstrapt werden, daher schleusen wir sie als binäre Seeds ein. Wir nennen das unser Bootstrapping mit kleinerem Seed, weil es die Größe unserer Bootstrapping-Binärdateien halbiert hat! Außerdem haben wir damit keiner-

lei Binärdatei für einen C-Compiler; auf i686-linux und x86_64-linux werden Guix-Pakete ganz ohne binären C-Compiler gebootstrapt.

Wir arbeiten daran, MesCC-Tools und Mes vollständig bootstrappen zu können, und behalten auch andere Bootstrapping-Binärdateien im Blick. Ihre Unterstützung ist willkommen!

21 Auf eine neue Plattform portieren

Wie oben beschrieben ist die GNU-Distribution eigenständig und diese Eigenständigkeit wird erreicht, indem sie aus vorerstellten „Bootstrap-Binärdateien“ heraus erstellt werden kann (siehe Kapitel 20 [Bootstrapping], Seite 699). Diese Binärdateien unterscheiden sich je nach verwendetem Betriebssystem-Kernel, nach der Prozessorarchitektur und der Anwendungsbinärschnittstelle („Application Binary Interface“, kurz ABI). Um die Distribution also auf eine noch nicht unterstützte Plattform zu portieren, muss man diese Bootstrap-Binärdateien für diese Plattform erstellen und das Modul `(gnu packages bootstrap)` aktualisieren, damit es sie benutzt.

Zum Glück kann Guix diese Bootstrap-Binärdateien *cross-kompilieren*. Wenn alles gut geht, und vorausgesetzt, die GNU-Werkzeuge (zusammen werden sie als GNU-„Toolchain“ bezeichnet) unterstützen diese Zielplattform auch, dann kann es völlig ausreichen, dass Sie einen Befehl wie hier ausführen:

```
guix build --target=armv5tel-linux-gnueabi bootstrap-tarballs
```

Damit das funktioniert, muss erst eine neue Plattform eingetragen werden; sie sind im Modul `(guix platform)` registriert. Eine Plattform stellt die Verbindung her zwischen einem GNU-Tripel (siehe Abschnitt “Specifying Target Triplets” in *Autoconf*), dem entsprechenden *System* in Nix-Notation, dem Dateinamen für den dynamischen Binder *glibc-dynamic-linker* von libc auf dieser Plattform und dem zugehörigen Namen der Linux-Architektur, sofern es um Linux geht (siehe Kapitel 15 [Plattformen], Seite 681).

Sobald Sie einen Bootstrap-Tarball haben, muss das Modul `(gnu packages bootstrap)` aktualisiert werden, damit es diese Binärdateien für die Zielplattform benutzt. Das heißt, die Hashes und URLs der Bootstrap-Tarballs für die neue Plattform müssen neben denen für die bisher unterstützten Plattformen aufgeführt werden. Der Bootstrap-Guile-Tarball wird besonders behandelt: Von ihm wird erwartet, dass er lokal verfügbar ist, und `gnu/local.mk` enthält Regeln, um ihn für die unterstützten Architekturen herunterzuladen; eine Regel muss auch für die neue Plattform hinzugefügt werden.

In der Praxis kann es einige Schwierigkeiten geben. Erstens kann es sein, dass das erweiterte GNU-Tripel, das eine Anwendungsbinärschnittstelle (ABI) festlegt (wie es das `eabi`-Suffix oben tut) nicht von allen GNU-Werkzeugen erkannt wird. Typischerweise erkennt glibc manche davon, während für GCC eine zusätzliche Befehlszeilenoption `--with-abi` an configure übergeben werden muss (siehe `gcc.scm` für Beispiele, wie man das macht). Zweitens könnte es sein, dass manche der notwendige Pakete für diese Plattform nicht erfolgreich erstellt werden können. Zuletzt könnten die generierten Binärdateien aus dem einen oder anderen Grund fehlerhaft sein.

22 Mitwirken

Dieses Projekt basiert auf Kooperation, daher benötigen wir Ihre Hilfe, um es wachsen zu lassen! Bitte kontaktieren Sie uns auf `guix-devel@gnu.org` und `#guix` im Libera-Chat-IRC-Netzwerk. Wir freuen uns auf Ihre Ideen, Fehlerberichte, Patches und alles, was hilfreich für das Projekt sein könnte. Besonders willkommen ist Hilfe beim Schreiben von Paketen (siehe Abschnitt 22.4 [Paketrichtlinien], Seite 713).

Wir möchten eine angenehme, freundliche und von Belästigungen freie Umgebung bereitstellen, so dass jeder Beiträge nach seinen Fähigkeiten leisten kann. Zu diesem Zweck verwendet unser Projekt einen „Verhaltenskodex für Mitwirkende“, der von `https://contributor-covenant.org/` übernommen wurde. Eine übersetzte Fassung finden Sie auf `https://www.contributor-covenant.org/de/version/1/4/code-of-conduct` sowie eine mitgelieferte, englische Fassung in der Datei `CODE-OF-CONDUCT` im Quellbaum.

Von Mitwirkenden wird nicht erwartet, dass sie in Patches oder in der Online-Kommunikation ihre echten Namen preisgeben. Sie können einen beliebigen Namen oder ein Pseudonym ihrer Wahl verwenden.

22.1 Erstellung aus dem Git

Wenn Sie an Guix selbst hacken wollen, ist es empfehlenswert, dass Sie die neueste Version aus dem Git-Softwarebestand verwenden:

```
git clone https://git.savannah.gnu.org/git/guix.git
```

Doch wie können Sie sichergehen, dass sie eine unverfälschte Kopie des Repositorys erhalten haben? Dazu müssen Sie `guix git authenticate` ausführen, wobei Sie den Commit und den OpenPGP-Fingerabdruck der *Kanaleinführung* angeben (siehe Abschnitt 8.5 [Aufruf von guix git authenticate], Seite 106):

```
git fetch origin keyring:keyring
guix git authenticate 9edb3f66fd807b096b48283debdcddccfea34bad \
  "BBB0 2DDF 2CEA F6A8 0D1D  E643 A2A0 6DF2 A33A 54FA"
```

Dieser Befehl gibt bei Erfolg am Ende null als Exit-Status zurück, ansonsten wird eine Fehlermeldung ausgegeben und ein anderer Exit-Status zurückgegeben.

Wie Sie sehen können, liegt hier ein Henne-Ei-Problem vor: Sie müssen Guix zuvor bereits installiert haben. Üblicherweise würden Sie Guix System erst auf einem anderen System installieren (siehe Kapitel 3 [Systeminstallation], Seite 27) oder Guix auf einer anderen Distribution installieren (siehe Abschnitt 2.1 [Aus Binärdatei installieren], Seite 5); in beiden Fällen würden Sie die OpenPGP-Signatur auf dem Installationsmedium prüfen. Damit haben Sie ein „Bootstrapping“ der Vertrauenskette durchgeführt.

Der einfachste Weg, eine Entwicklungsumgebung für Guix einzurichten, ist natürlich, Guix zu benutzen! Der folgende Befehl startet eine neue Shell, in der alle Abhängigkeiten und Umgebungsvariablen bereits eingerichtet sind, um an Guix zu arbeiten:

```
guix shell -D guix --pure
```

Siehe Abschnitt 8.1 [Aufruf von guix shell], Seite 86, für weitere Informationen zu diesem Befehl.

Wenn Sie Guix nicht benutzen können, wenn Sie es aus einem Checkout erstellen, werden die folgenden Pakete zusätzlich zu denen benötigt, die in den Installationsanweisungen angegeben sind (siehe Abschnitt 2.2 [Voraussetzungen], Seite 8).

- 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 (optional) (`https://www.gnu.org/software/help2man/`).

Auf Guix können zusätzliche Abhängigkeiten hinzugefügt werden, indem Sie stattdessen `guix shell` ausführen:

```
guix shell -D guix help2man git strace --pure
```

Damit können Sie die Infrastruktur des Erstellungssystems mit Autoconf und Automake erzeugen.

```
./bootstrap
```

Möglicherweise erhalten Sie eine Fehlermeldung wie diese:

```
configure.ac:46: error: possibly undefined macro: PKG_CHECK_MODULES
```

Das bedeutet wahrscheinlich, dass Autoconf `pkg.m4` nicht finden konnte, welches von pkg-config bereitgestellt wird. Stellen Sie sicher, dass `pkg.m4` verfügbar ist. Gleiches gilt für den von Guile bereitgestellten Makrosatz `guile.m4`. Wenn Sie beispielsweise Automake in `/usr/local` installiert haben, würde in `/usr/share` nicht nach `.m4`-Dateien geschaut. In einem solchen Fall müssen Sie folgenden Befehl aufrufen:

```
export ACLOCAL_PATH=/usr/share/aclocal
```

In Abschnitt "Macro Search Path" in *The GNU Automake Manual* finden Sie weitere Informationen.

Anschließend führen Sie dies aus:

```
./configure --localstatedir=/var
```

Dabei ist `/var` der normale `localstatedir`-Wert (weitere Informationen siehe Abschnitt 9.9 [Der Store], Seite 165). Denken Sie daran, dass Sie am Ende wahrscheinlich *nicht* `make install` ausführen möchten (müssen Sie auch nicht), aber es ist dennoch wichtig, die richtige `localstatedir` zu übergeben.

Schließlich können Sie Guix erstellen und, wenn Sie möchten, die Tests ausführen (siehe Abschnitt 2.3 [Den Testkatalog laufen lassen], Seite 10):

```
make
make check
```

Falls etwas fehlschlägt, werfen Sie einen Blick auf die Installationsanweisungen (siehe Kapitel 2 [Installation], Seite 5) oder senden Sie eine E-Mail an die Mailingliste.

Von da an können Sie alle Commits in Ihrem Checkout authentifizieren, indem Sie dies ausführen:

```
make authenticate
```

Die erste Ausführung dauert ein paar Minuten, aber nachfolgende Ausführungen gehen schneller.

Für den Fall, dass die Konfiguration Ihres lokal verfügbaren Git-Repositorys nicht der voreingestellten entspricht, können Sie die Referenz für den `keyring`-Branch über die Variable `GUIX_GIT_KEYRING` angeben. Im folgenden Beispiel nehmen wir an, Sie haben ein Remote-Repository namens '`myremote`' eingerichtet, das auf das offizielle Guix-Repository verweist:

```
make authenticate GUIX_GIT_KEYRING=myremote/keyring
```

Anmerkung: Wir raten Ihnen dazu, nach jedem Aufruf von `git pull` auch `make authenticate` auszuführen. Das stellt sicher, dass Sie weiterhin nur gültige Änderungen in Ihr Repository übernehmen.

Wenn Sie das Repository einmal aktualisieren, kann es passieren, dass `make` mit einer Fehlermeldung ähnlich wie dieser fehlschlägt:

```
error: failed to load 'gnu/packages/dunst.scm':
ice-9/eval.scm:293:34: In procedure abi-check: #<record-type <origin>>: record ABI mis
```

Das bedeutet, dass sich einer der in Guix definierten Verbundstypen (in diesem Beispiel der `origin`-Verbundstyp) geändert hat und alle Teile von Guix für diese Änderung neu kompiliert werden müssen. Um das zu veranlassen, führen Sie `make clean-go` aus, gefolgt von `make`.

22.2 Guix vor der Installation ausführen

Um eine gesunde Arbeitsumgebung zu erhalten, ist es hilfreich, die im lokalen Quellbaum vorgenommenen Änderungen zunächst zu testen, ohne sie tatsächlich zu installieren. So können Sie zwischen Ihrem Endnutzer-„Straßenanzug“ und Ihrem „Faschingskostüm“ unterscheiden.

Zu diesem Zweck können alle Befehlszeilenwerkzeuge auch schon benutzt werden, ohne dass Sie `make install` laufen lassen. Dazu müssen Sie sich in einer Umgebung befinden, in der alle Abhängigkeiten von Guix verfügbar sind (siehe Abschnitt 22.1 [Erstellung aus dem Git], Seite 708) und darin einfach vor jeden Befehl `./pre-inst-env` schreiben (das Skript `pre-inst-env` befindet sich auf oberster Ebene im Verzeichnis, wo Guix erstellt wird; es wird durch Ausführung von `./bootstrap` gefolgt von `./configure` erzeugt). Zum Beispiel würden Sie so das Paket `hello` erstellen lassen, so wie es in der gegenwärtigen Kopie des Guix-Quellbaums definiert wurde (es wird angenommen, dass `guix-daemon` auf Ihrem System bereits läuft, auch wenn es eine andere Version ist):

```
$ ./pre-inst-env guix build hello
```

Entsprechend würden Sie dies eingeben, um eine Guile-Sitzung zu öffnen, die die Guix-Module benutzt:

```
$ ./pre-inst-env guile -c '(use-modules (guix utils)) (pk (%current-system))'

;;; ("x86_64-linux")
```

. . . und auf einer REPL (siehe Abschnitt 9.14 [Interaktiv mit Guix arbeiten], Seite 186):

```
$ ./pre-inst-env guile
scheme@(guile-user)> ,use(guix)
scheme@(guile-user)> ,use(gnu)
scheme@(guile-user)> (define snakes
```

```
                          (fold-packages
                            (lambda (package lst)
                              (if (string-prefix? "python"
                                                  (package-name package))
                                  (cons package lst)
                                  lst))
                            '()))
scheme@(guile-user)> (length snakes)
$1 = 361
```

Wenn Sie am Daemon und damit zu tun habendem Code hacken oder wenn `guix-daemon` nicht bereits auf Ihrem System läuft, können Sie ihn direkt aus dem Verzeichnis heraus starten, wo Sie Guix erstellen lassen[1]:

```
$ sudo -E ./pre-inst-env guix-daemon --build-users-group=guixbuild
```

Das `pre-inst-env`-Skript richtet alle Umgebungsvariablen ein, die nötig sind, um dies zu ermöglichen, einschließlich `PATH` und `GUILE_LOAD_PATH`.

Beachten Sie, dass `./pre-inst-env guix pull` den lokalen Quellbaum *nicht* aktualisiert; es aktualisiert lediglich die symbolische Verknüpfung `~/.config/guix/current` (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65). Um Ihren lokalen Quellbaum zu aktualisieren, müssen Sie stattdessen `git pull` benutzen.

Manchmal, insbesondere wenn Sie das Repository aktualisiert haben, wird beim Ausführen mit `./pre-inst-env` eine Nachricht ähnlich wie in diesem Beispiel erscheinen:

```
;;; note: source file /home/user/projects/guix/guix/progress.scm
;;;       newer than compiled /home/user/projects/guix/guix/progress.go
```

Es handelt sich lediglich um einen Hinweis und Sie können ihn getrost ignorieren. Los werden Sie die Nachricht, indem Sie `make -j4` ausführen. Bis dahin läuft Guile etwas langsamer als sonst, weil es den Quellcode interpretieren muss und nicht auf vorbereitete Guile-Objekt-Dateien (`.go`) zurückgreifen kann.

Sie können `make` automatisch ausführen lassen, während Sie am Code arbeiten, nämlich mit `watchexec` aus dem Paket `watchexec`. Um zum Beispiel jedes Mal neu zu erstellen, wenn Sie etwas an einer Paketdatei ändern, führen Sie '`watchexec -w gnu/packages --make -j4`' aus.

22.3 Perfekt eingerichtet

Um perfekt für das Hacken an Guix eingerichtet zu sein, brauchen Sie an sich dasselbe wie um perfekt für das Hacken mit Guile (siehe Abschnitt "Using Guile in Emacs" in *Guile-Referenzhandbuch*). Zunächst brauchen Sie mehr als ein Textverarbeitungsprogramm, Sie brauchen Emacs (`https://www.gnu.org/software/emacs`) zusammen mit den vom wunderbaren Geiser (`https://nongnu.org/geiser/`) verliehenen Kräften. Um diese zu installieren, können Sie Folgendes ausführen:

```
guix install emacs guile emacs-geiser emacs-geiser-guile
```

Geiser ermöglicht interaktive und inkrementelle Entwicklung aus Emacs heraus: Code kann in Puffern kompiliert und ausgewertet werden. Zugang zu Online-Dokumentation

[1] Die Befehlszeilenoption `-E` von `sudo` stellt sicher, dass `GUILE_LOAD_PATH` richtig gesetzt wird, damit `guix-daemon` und die davon benutzten Werkzeuge die von ihnen benötigten Guile-Module finden können.

(Docstrings) steht ebenso zur Verfügung wie kontextabhängige Vervollständigung, `M-.` um zu einer Objektdefinition zu springen, eine REPL, um Ihren Code auszuprobieren, und mehr (siehe Abschnitt "Introduction" in *Geiser User Manual*). Zur bequemen Guix-Entwicklung sollten Sie Guiles Ladepfad so ergänzen, dass die Quelldateien in Ihrem Checkout gefunden werden.

```
;; Angenommen das Guix-Checkout ist in ~/src/guix.
(with-eval-after-load 'geiser-guile
  (add-to-list 'geiser-guile-load-path "~/src/guix"))
```

Um den Code tatsächlich zu bearbeiten, bietet Emacs schon einen netten Scheme-Modus. Aber Sie dürfen auch Paredit (`https://www.emacswiki.org/emacs/ParEdit`) nicht verpassen. Es bietet Hilfsmittel, um direkt mit dem Syntaxbaum zu arbeiten, und kann so zum Beispiel einen S-Ausdruck hochheben oder ihn umhüllen, ihn verschlucken oder den nachfolgenden S-Ausdruck verwerfen, etc.

Wir bieten auch Vorlagen an für häufige Git-Commit-Nachrichten und Paketdefinitionen im Verzeichnis `etc/snippets`. Diese Vorlagen können benutzt werden, um kurze Auslöse-Zeichenketten zu interaktiven Textschnipseln umzuschreiben. Wenn Sie dazu YASnippet (`https://joaotavora.github.io/yasnippet/`) verwenden, möchten Sie vielleicht das Schnipselverzeichnis `etc/snippets/yas` zu Ihrer *yas-snippet-dirs*-Variablen in Emacs hinzufügen. Wenn Sie Tempel (`https://github.com/minad/tempel/`) verwenden, fügen Sie stattdessen den Pfad `etc/snippets/tempel/*` zur Emacs-Variablen *tempel-path* hinzu.

```
;; Angenommen das Guix-Checkout ist in ~/src/guix.
;; Yasnippet-Konfiguration
(with-eval-after-load 'yasnippet
  (add-to-list 'yas-snippet-dirs "~/src/guix/etc/snippets/yas"))
;; Tempel-Konfiguration
(with-eval-after-load 'tempel
   ;; Wenn tempel-path noch keine Liste ist, sondern eine Zeichenkette
   (unless (listp 'tempel-path)
     (setq tempel-path (list tempel-path)))
   (add-to-list 'tempel-path "~/src/guix/etc/snippets/tempel/*"))
```

Die Schnipsel für Commit-Nachrichten setzen Magit (`https://magit.vc/`) voraus, um zum Commit vorgemerkte Dateien anzuzeigen. Wenn Sie eine Commit-Nachricht bearbeiten, können Sie `add` gefolgt von *`TAB`* eintippen, um eine Commit-Nachrichten-Vorlage für das Hinzufügen eines Pakets zu erhalten; tippen Sie `update` gefolgt von *`TAB`* ein, um eine Vorlage zum Aktualisieren eines Pakets zu bekommen; tippen Sie `https` gefolgt von *`TAB`* ein, um eine Vorlage zum Ändern der Homepage-URI eines Pakets auf HTTPS einzufügen.

Das Hauptschnipsel für `scheme-mode` wird ausgelöst, indem Sie `package...` gefolgt von *`TAB`* eintippen. Dieses Snippet fügt auch die Auslöse-Zeichenkette `origin...` ein, die danach weiter umgeschrieben werden kann. Das `origin`-Schnipsel kann wiederum andere Auslöse-Zeichenketten einfügen, die alle auf `...` enden, was selbst wieder weiter umgeschrieben werden kann.

Außerden stellen wir automatisches Einfügen und Aktualisieren von Urheberrechtsinformationen („Copyright“) über `etc/copyright.el` zur Verfügung. Dazu müssten Sie Ihren vollständigen Namen mit E-Mail-Adresse festlegen und eine Datei laden.

```
(setq user-full-name "Alice Doe")
(setq user-mail-address "alice@mail.org")
;; Assuming the Guix checkout is in ~/src/guix.
(load-file "~/src/guix/etc/copyright.el")
```

Um an der aktuellen Zeile Copyright-Informationen einzufügen, rufen Sie `M-x guix-copyright` auf.

Um Copyright-Informationen aktualisieren zu können, müssen Sie einen regulären Ausdruck `copyright-names-regexp` angeben.

```
(setq copyright-names-regexp
      (format "%s <%s>" user-full-name user-mail-address))
```

Sie können prüfen, ob Ihre Urheberrechtsinformationen aktuell sind, indem Sie `M-x copyright-update` auswerten. Wenn Sie möchten, dass dies automatisch nach jedem Speichern des Puffers geschieht, fügen Sie `(add-hook 'after-save-hook 'copyright-update)` in Emacs hinzu.

22.4 Paketrichtlinien

Die GNU-Distribution ist noch sehr jung und einige Ihrer Lieblingspakete könnten noch fehlen. Dieser Abschnitt beschreibt, wie Sie dabei helfen können, die Distribution wachsen zu lassen.

Pakete mit freier Software werden normalerweise in Form von *Tarballs mit dem Quellcode* angeboten – typischerweise in `tar.gz`-Archivdateien, in denen alle Quelldateien enthalten sind. Ein Paket zur Distribution hinzuzufügen, bedeutet also zweierlei Dinge: Zum einen fügt man ein *Rezept* ein, das beschreibt, wie das Paket erstellt werden kann, einschließlich einer Liste von anderen Paketen, die für diese Erstellung gebraucht werden, zum anderen fügt man *Paketmetadaten* zum Rezept hinzu, wie zum Beispiel eine Beschreibung und Lizenzinformationen.

In Guix sind all diese Informationen ein Teil der *Paketdefinitionen*. In Paketdefinitionen hat man eine abstrahierte, hochsprachliche Sicht auf das Paket. Sie werden in der Syntax der Scheme-Programmiersprache verfasst; tatsächlich definieren wir für jedes Paket eine Variable und binden diese an dessen Definition, um die Variable anschließend aus einem Modul heraus zu exportieren (siehe Abschnitt 9.1 [Paketmodule], Seite 108). Allerdings ist *kein* tiefgehendes Wissen über Scheme erforderlich, um Pakete zu erstellen. Mehr Informationen über Paketdefinitionen finden Sie im Abschnitt Abschnitt 9.2 [Pakete definieren], Seite 109.

Eine fertige Paketdefinition kann, nachdem sie in eine Datei im Quell-Verzeichnisbaum von Guix eingesetzt wurde, mit Hilfe des Befehls `guix build` getestet werden (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189). Wenn das Paket zum Beispiel den Namen `gnew` trägt, können Sie folgenden Befehl aus dem Erstellungs-Verzeichnisbaum von Guix heraus ausführen (siehe Abschnitt 22.2 [Guix vor der Installation ausführen], Seite 710):

```
./pre-inst-env guix build gnew --keep-failed
```

Wenn Sie `--keep-failed` benutzen, ist es leichter, fehlgeschlagene Erstellungen zu untersuchen, weil dann der Verzeichnisbaum der fehlgeschlagenen Erstellung zugänglich bleibt. Eine andere nützliche Befehlszeilenoption bei der Fehlersuche ist `--log-file`, womit das Erstellungsprotokoll eingesehen werden kann.

Wenn der `guix`-Befehl das Paket nicht erkennt, kann es daran liegen, dass die Quelldatei einen Syntaxfehler hat oder ihr eine `define-public`-Klausel fehlt, die die Paketvariable exportiert. Um das herauszufinden, können Sie das Modul aus Guile heraus laden, um mehr Informationen über den tatsächlichen Fehler zu bekommen:

```
./pre-inst-env guile -c '(use-modules (gnu packages gnew))'
```

Sobald Ihr Paket erfolgreich erstellt werden kann, schicken Sie uns bitte einen Patch (siehe Abschnitt 22.6 [Einreichen von Patches], Seite 724). Wenn Sie dabei Hilfe brauchen sollten, helfen wir gerne. Ab dem Zeitpunkt, zu dem der Patch als Commit ins Guix-Repository eingepflegt wurde, wird das neue Paket automatisch durch unser System zur Kontinuierlichen Integration (`https://ci.guix.gnu.org`) auf allen unterstützten Plattformen erstellt.

Benutzern steht die neue Paketdefinition zur Verfügung, nachdem sie das nächste Mal `guix pull` ausgeführt haben (siehe Abschnitt 6.6 [Aufruf von guix pull], Seite 65). Wenn `ci.guix.gnu.org` selbst damit fertig ist, das Paket zu erstellen, werden bei der Installation automatisch Binärdateien von dort heruntergeladen (siehe Abschnitt 6.3 [Substitute], Seite 56). Menschliches Eingreifen muss nur stattfinden, um den Patch zu überprüfen und anzuwenden.

22.4.1 Software-Freiheit

Das GNU-Betriebssystem wurde entwickelt, um Menschen Freiheit zu ermöglichen, wie sie ihre Rechengeräte benutzen. GNU ist *freie Software*, was bedeutet, dass Benutzer die vier wesentlichen Freiheiten (`https://www.gnu.org/philosophy/free-sw.de.html`) haben: das Programm auszuführen, es zu untersuchen, das Programm in Form seines Quellcodes anzupassen und exakte Kopien ebenso wie modifizierte Versionen davon an andere weiterzugeben. Die Pakete, die Sie in der GNU-Distribution finden, stellen ausschließlich solche Software zur Verfügung, die Ihnen diese vier Freiheiten gewährt.

Außerdem befolgt die GNU-Distribution die Richtlinien für freie Systemverteilungen (`https://www.gnu.org/distros/free-system-distribution-guidelines.de.html`). Unter anderem werden unfreie Firmware sowie Empfehlungen von unfreier Software abgelehnt und Möglichkeiten zum Umgang mit Markenzeichen und Patenten werden diskutiert.

Ansonsten freier Paketquellcode von manchen Anbietern enthält einen kleinen und optionalen Teil, der diese Richtlinien verletzt. Zum Beispiel kann dieser Teil selbst unfreier Code sein. Wenn das vorkommt, wird der sie verletzende Teil mit angemessenen Patches oder Code-Schnipseln innerhalb der `origin`-Form des Pakets entfernt (siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Dadurch liefert Ihnen `guix build --source` nur den „befreiten“ Quellcode und nicht den unmodifizierten Quellcode des Anbieters.

22.4.2 Paketbenennung

Tatsächlich sind mit jedem Paket zwei Namen assoziiert: Zum einen gibt es den Namen der *Scheme-Variablen*, der direkt nach `define-public` im Code steht. Mit diesem Namen kann das Paket im Scheme-Code nutzbar gemacht und zum Beispiel als Eingabe eines anderen Pakets benannt werden. Zum anderen gibt es die Zeichenkette im `name`-Feld einer Paketdefinition. Dieser Name wird von Paketverwaltungsbefehlen wie `guix package` und `guix build` benutzt.

Meistens sind die beiden identisch und ergeben sich aus einer Umwandlung des vom Anbieter verwendeten Projektnamens in Kleinbuchstaben, bei der Unterstriche durch Bin-

destriche ersetzt werden. Zum Beispiel wird GNUnet unter dem Paketnamen `gnunet` angeboten und SDL_net als `sdl-net`.

Es gibt eine nennenswerte Ausnahme zu dieser Regel, nämlich wenn der Projektname nur ein einzelnes Zeichen ist oder auch wenn es bereits ein älteres, aktives Projekt mit demselben Namen gibt, egal ob es schon für Guix verpackt wurde. Lassen Sie Ihren gesunden Menschenverstand einen eindeutigen und sprechenden Namen auswählen. Zum Beispiel wurde die Shell, die bei ihrem Anbieter „s“ heißt, `s-shell` genannt und *nicht* `s`. Sie sind eingeladen, Ihre Hackerkollegen um Inspiration zu bitten.

An Bibliothekspakete hängen wir vorne kein `lib` als Präfix an, solange es nicht Teil des offiziellen Projektnamens ist. Beachten Sie aber die Abschnitte Abschnitt 22.4.7 [Python-Module], Seite 719, und Abschnitt 22.4.8 [Perl-Module], Seite 720, in denen Sonderregeln für Module der Programmiersprachen Python und Perl beschrieben sind.

Auch Pakete mit Schriftarten werden anders behandelt, siehe Abschnitt 22.4.12 [Schriftarten], Seite 723.

22.4.3 Versionsnummern

Normalerweise stellen wir nur für die neueste Version eines Freie-Software-Projekts ein Paket bereit. Manchmal gibt es allerdings Fälle wie zum Beispiel untereinander inkompatible Bibliotheksversionen, so dass zwei (oder mehr) Versionen desselben Pakets angeboten werden müssen. In diesem Fall müssen wir verschiedene Scheme-Variablennamen benutzen. Wir benutzen dann für die neueste Version den Namen, wie er im Abschnitt Abschnitt 22.4.2 [Paketbenennung], Seite 714, festgelegt wird, und geben vorherigen Versionen denselben Namen mit einem zusätzlichen Suffix aus `-` gefolgt vom kürzesten Präfix der Versionsnummer, mit dem noch immer zwei Versionen unterschieden werden können.

Der Name innerhalb der Paketdefinition ist hingegen derselbe für alle Versionen eines Pakets und enthält keine Versionsnummer.

Zum Beispiel können für GTK in den Versionen 2.24.20 und 3.9.12 Pakete wie folgt geschrieben werden:

```
(define-public gtk+
  (package
    (name "gtk+")
    (version "3.9.12")
    ...))
(define-public gtk+-2
  (package
    (name "gtk+")
    (version "2.24.20")
    ...))
```

Wenn wir auch GTK 3.8.2 wollten, würden wir das Paket schreiben als

```
(define-public gtk+-3.8
  (package
    (name "gtk+")
    (version "3.8.2")
    ...))
```

Gelegentlich fügen wir auch Pakete für Snapshots aus dem Versionskontrollsystem des Anbieters statt formaler Veröffentlichungen zur Distribution hinzu. Das sollte die Ausnahme bleiben, weil die Entwickler selbst klarstellen sollten, welche Version als die stabile Veröffentlichung gelten sollte, ab und zu ist es jedoch notwendig. Was also sollten wir dann im `version`-Feld eintragen?

Offensichtlich muss der Bezeichner des Commits, den wir als Snapshot aus dem Versionskontrollsystem nehmen, in der Versionszeichenkette zu erkennen sein, aber wir müssen auch sicherstellen, dass die Version monoton steigend ist, damit `guix package --upgrade` feststellen kann, welche Version die neuere ist. Weil Commit-Bezeichner, insbesondere bei Git, nicht monoton steigen, müssen wir eine Revisionsnummer hinzufügen, die wir jedes Mal erhöhen, wenn wir das Paket auf einen neueren Snapshot aktualisieren. Die sich ergebende Versionszeichenkette sieht dann so aus:

```
2.0.11-3.cabba9e
  ^    ^    ^
  |    |    `-- Commit-ID beim Anbieter
  |    |
  |    `--- Revisionsnummer des Guix-Pakets
  |
die neueste Version, die der Anbieter veröffentlicht hat
```

Es ist eine gute Idee, die Commit-Bezeichner im `version`-Feld auf, sagen wir, 7 Ziffern zu beschränken. Das sieht besser aus (wenn das hier eine Rolle spielen sollte) und vermeidet Probleme, die mit der maximalen Länge von Shebangs zu tun haben (127 Bytes beim Linux-Kernel). Bei Paketen, die `git-fetch` oder `hg-fetch` benutzen, können Sie dafür Hilfsfunktionen nutzen (siehe unten). Am besten benutzen Sie in `origin`s jedoch den vollständigen Commit-Bezeichner, um Mehrdeutigkeiten zu vermeiden. Eine typische Paketdefinition könnte so aussehen:

```
(define mein-paket
  (let ((commit "c3f29bc928d5900971f65965feaae59e1272a3f7")
        (revision "1"))          ;Guix-Paketrevision
    (package
      (version (git-version "0.9" revision commit))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "git://example.org/mein-paket.git")
                      (commit commit)))
                (sha256 (base32 "1mbikn..."))
                (file-name (git-file-name name version))))
      ;; ...
      )))
```

`git-version` *VERSION REVISION COMMIT* [Scheme-Prozedur]

Liefert die Zeichenkette für die Version bei Paketen, die `git-fetch` benutzen.

```
(git-version "0.2.3" "0" "93818c936ee7e2f1ba1b315578bde363a7d43d05")
⇒ "0.2.3-0.93818c9"
```

`hg-version` *VERSION REVISION ÄNDERUNGSSATZ* [Scheme-Prozedur]
Liefert die Zeichenkette für die Version bei Paketen, die `hg-fetch` benutzen.<

22.4.4 Zusammenfassungen und Beschreibungen

Wie wir bereits gesehen haben, enthält jedes Paket in GNU Guix eine (im Code englischsprachige) Zusammenfassung (englisch: Synopsis) und eine Beschreibung (englisch: Description; siehe Abschnitt 9.2 [Pakete definieren], Seite 109). Zusammenfassungen und Beschreibungen sind wichtig: Sie werden mit `guix package --search` durchsucht und stellen eine entscheidende Informationsquelle für Nutzer dar, die entscheiden wollen, ob das Paket Ihren Bedürfnissen entspricht, daher sollten Paketentwickler achtgeben, was sie dort eintragen.

Zusammenfassungen müssen mit einem Großbuchstaben beginnen und dürfen nicht mit einem Punkt enden. Sie dürfen nicht mit den Artikeln „a“ oder „the“ beginnen, die meistens ohnehin nichts zum Verständnis beitragen. Zum Beispiel sollte „File-frobbing tool“ gegenüber „A tool that frobs files“ vorgezogen werden. Die Zusammenfassung sollte aussagen, um was es sich beim Paket handelt – z.B. „Core GNU utilities (file, text, shell)“ –, oder aussagen, wofür es benutzt wird – z.B. ist die Zusammenfassung für GNU grep „Print lines matching a pattern“.

Beachten Sie, dass die Zusammenfassung für eine sehr große Leserschaft einen Sinn ergeben muss. Zum Beispiel würde „Manipulate alignments in the SAM format“ vielleicht von einem erfahrenen Forscher in der Bioinformatik verstanden, könnte für die Nicht-Spezialisten in Guix' Zielgruppe aber wenig hilfreich sein oder würde diesen sogar eine falsche Vorstellung geben. Es ist eine gute Idee, sich eine Zusammenfassung zu überlegen, die eine Vorstellung von der Anwendungsdomäne des Pakets vermittelt. Im Beispiel hier würden sich die Nutzer mit „Manipulate nucleotide sequence alignments“ hoffentlich ein besseres Bild davon machen können, ob das Paket ist, wonach sie suchen.

Beschreibungen sollten zwischen fünf und zehn Zeilen lang sein. Benutzen Sie vollständige Sätze und vermeiden Sie Abkürzungen, die Sie nicht zuvor eingeführt haben. Vermeiden Sie bitte Marketing-Phrasen wie „world-leading“ („weltweit führend“), „industrial-strength“ („industrietauglich“) und „next-generation“ („der nächsten Generation“) ebenso wie Superlative wie „the most advanced“ („das fortgeschrittenste“) – davon haben Nutzer nichts, wenn sie ein Paket suchen, und es könnte sogar verdächtig klingen. Versuchen Sie stattdessen, bei den Fakten zu bleiben und dabei Anwendungszwecke und Funktionalitäten zu erwähnen.

Beschreibungen können wie bei Texinfo ausgezeichneten Text enthalten. Das bedeutet, Text kann Verzierungen wie `@code` oder `@dfn`, Auflistungen oder Hyperlinks enthalten (siehe Abschnitt “Overview” in *GNU Texinfo*). Sie sollten allerdings vorsichtig sein, wenn Sie bestimmte Zeichen wie ‘`@`’ und geschweifte Klammern schreiben, weil es sich dabei um die grundlegenden Sonderzeichen in Texinfo handelt (siehe Abschnitt “Special Characters” in *GNU Texinfo*). Benutzungsschnittstellen wie `guix show` kümmern sich darum, solche Auszeichnungen angemessen darzustellen.

Zusammenfassungen und Beschreibungen werden von Freiwilligen bei Weblate (`https://translate.fedoraproject.org/projects/guix/packages`) übersetzt, damit so viele Nutzer wie möglich sie in ihrer Muttersprache lesen können. Mit Schnittstellen für Benutzer können sie in der von der aktuell eingestellten Locale festgelegten Sprache durchsucht und angezeigt werden.

Damit `xgettext` sie als übersetzbare Zeichenketten extrahieren kann, *müssen* Zusammenfassungen und Beschreibungen einfache Zeichenketten-Literale sein. Das bedeutet, dass Sie diese Zeichenketten nicht mit Prozeduren wie `string-append` oder `format` konstruieren können:

```
(package
  ;; ...
  (synopsis "This is translatable")
  (description (string-append "This is " "*not*" " translatable.")))
```

Übersetzen ist viel Arbeit, also passen Sie als Paketentwickler bitte umso mehr auf, wenn Sie Ihre Zusammenfassungen und Beschreibungen formulieren, weil jede Änderung zusätzliche Arbeit für Übersetzer bedeutet. Um den Übersetzern zu helfen, können Sie Empfehlungen und Anweisungen für diese sichtbar machen, indem Sie spezielle Kommentare wie in diesem Beispiel einfügen (siehe Abschnitt "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. ...")
```

22.4.5 „Snippets“ oder Phasen

Die Grenze, wann man Code-Schnipsel im `origin`-„`snippet`“-Feld gegenüber einer Erstellungsphase für Änderungen an den Quelldateien eines Pakets bevorzugen sollte, ist fließend. Schnipsel im Paketursprung werden meistens dazu benutzt, unerwünschte Dateien wie z.B. gebündelte Bibliotheken oder unfreie Quelldateien zu entfernen, oder um einfache Textersetzungen durchzuführen. Die Quelle, die sich aus einem Paketursprung ableitet, sollte Quellcode erzeugen, um das Paket auf jedem vom Anbieter unterstützten System zu erstellen (d.h. um als dessen „corresponding source“, „korrespondierender Quelltext“, herzuhalten). Insbesondere dürfen Snippets im Paketursprung keine Store-Objekte in den Quelldateien einbetten; solche Anpassungen sollten besser in Erstellungsphasen stattfinden. Schauen Sie in die Dokumentation des `origin`-Verbundsobjekts für weitere Informationen (siehe Abschnitt 9.2.2 [„origin“-Referenz], Seite 118).

22.4.6 Emacs-Pakete

Für Emacs-Pakete sollte man bevorzugt das Emacs-Erstellungssystem benutzen (siehe [emacs-build-system], Seite 148), wegen der Einheitlichkeit und der Vorteile durch seine Erstellungsphasen. Dazu gehören das automatische Erzeugen der Autoloads-Datei und das Kompilieren des Quellcodes zu Bytecode (Byte Compilation). Weil es keinen Standard gibt, wie ein Testkatalog eines Emacs-Pakets ausgeführt wird, sind Tests nach Vorgabe abgeschaltet. Wenn es einen Testkatalog gibt, sollte er aktiviert werden, indem Sie das Argument `#:tests?` auf `#true` setzen. Vorgegeben ist, die Tests mit dem Befehl `make check` auszuführen, aber mit dem Argument `#:test-command` kann ein beliebiger anderer Befehl festgelegt werden. Für das Argument `#:test-command` wird eine Liste aus dem Befehl und den Argumenten an den Befehl erwartet. Er wird während der `check`-Phase aufgerufen.

Die Elisp-Abhängigkeiten von Emacs-Paketen werden typischerweise als `propagated-inputs` bereitgestellt, wenn sie zur Laufzeit benötigt werden. Wie bei anderen Paketen sollten Abhängigkeiten zum Erstellen oder Testen als `native-inputs` angegeben werden.

Manchmal hängen Emacs-Pakete von Ressourcenverzeichnissen ab, die zusammen mit den Elisp-Dateien installiert werden sollten. Diese lassen sich mit dem Argument `#:include` kennzeichnen, indem eine Liste dazu passender regulärer Ausdrücke angegeben wird. Idealerweise ergänzen Sie den Vorgabewert des `#:include`-Arguments (aus der Variablen `%default-include`), statt ihn zu ersetzen. Zum Beispiel enthält ein yasnippet-Erweiterungspaket typischerweise ein `snippets`-Verzeichnis, das wie folgt in das Installationsverzeichnis kopiert werden könnte:

```
#:include (cons "^snippets/" %default-include)
```

Wenn Sie auf Probleme stoßen, ist es ratsam, auf eine Kopfzeile `Package-Requires` in der Hauptquellcodedatei des Pakets zu achten, ob allen Abhängigkeiten und deren dort gelisteten Versionen genügt wird.

22.4.7 Python-Module

Zurzeit stellen wir ein Paket für Python 2 und eines für Python 3 jeweils über die Scheme-Variablen mit den Namen `python-2` und `python` zur Verfügung, entsprechend der Erklärungen im Abschnitt Abschnitt 22.4.3 [Versionsnummern], Seite 715. Um Verwirrungen und Namenskollisionen mit anderen Programmiersprachen zu vermeiden, erscheint es als wünschenswert, dass der Name eines Pakets für ein Python-Modul auch das Wort `python` enthält.

Manche Module sind nur mit einer Version von Python kompatibel, andere mit beiden. Wenn das Paket Foo mit Python 3 kompiliert wird, geben wir ihm den Namen `python-foo`. Wenn es mit Python 2 kompiliert wird, wählen wir den Namen `python2-foo`. Pakete sollten dann hinzugefügt werden, wenn sie gebraucht werden. Wir erstellen keine Python-2-Varianten von Paketen, wenn wir sie nicht benutzen wollen.

Wenn ein Projekt bereits das Wort `python` im Namen hat, lassen wir es weg; zum Beispiel ist das Modul python-dateutil unter den Namen `python-dateutil` und `python2-dateutil` verfügbar. Wenn der Projektname mit `py` beginnt (z.B. `pytz`), behalten wir ihn bei und stellen das oben beschriebene Präfix voran.

> **Anmerkung:** Im Moment sind gleich zwei verschiedene Erstellungssysteme für Python-Pakete in Guix in Umlauf: *python-build-system* und *pyproject-build-system*. Lange Zeit hat man sein Python-Paket aus einer Datei `setup.py` heraus erstellt, deren gewachsene Struktur *nicht* formell festgelegt war. Das lief überraschend gut, wenn man sich den Erfolg von Python anschaut, allerdings ließen sich nur schwer Werkzeuge zur Handhabung schreiben. Daraus ergab sich eine Vielzahl alternativer Erstellungssysteme, bis sich die Gemeinschaft auf einen formellen Standard (`https://peps.python.org/pep-0517/`) zum Festlegen der Erstellungsanforderungen geeinigt hatte. *pyproject-build-system* ist die Implementierung dieses Standards in Guix. Wir stufen es als „experimentell“ ein, insofern als dass es noch nicht all die *Build Backends* für PEP-517 unterstützt. Dennoch würden wir es begrüßen, wenn Sie es für neue Python-Pakete verwenden und Probleme melden würden. Schlussendlich wird es für veraltet erklärt werden und in *python-build-system* aufgehen.

22.4.7.1 Abhängigkeiten angeben

Informationen über Abhängigkeiten von Python-Paketen, welche mal mehr und mal weniger stimmen, finden sich normalerweise im Verzeichnisbaum des Paketquellcodes: in der Datei

`pyproject.toml`, der Datei `setup.py`, in `requirements.txt` oder in `tox.ini` (letztere beherbergt hauptsächlich Abhängigkeiten für Tests).

Wenn Sie ein Rezept für ein Python-Paket schreiben, lautet Ihr Auftrag, diese Abhängigkeiten auf angemessene Arten von „Eingaben“ abzubilden (siehe Abschnitt 9.2.1 [„package“-Referenz], Seite 113). Obwohl der `pypi`-Importer hier normalerweise eine gute Arbeit leistet (siehe Abschnitt 10.5 [Aufruf von guix import], Seite 206), könnten Sie die folgende Prüfliste durchgehen wollen, um zu bestimmen, wo welche Abhängigkeit eingeordnet werden sollte.

- Derzeit ist unser Python-Paket so geschrieben, dass es `setuptools` und `pip` mitinstalliert. Das wird sich bald ändern und wir möchten unseren Nutzern nahelegen, für Python-Entwicklungsumgebungen `python-toolchain` zu verwenden.

  `guix lint` wird Sie mit einer Warnung darauf aufmerksam machen, wenn `setuptools` oder `pip` zu den nativen Eingaben hinzugefügt wurden, weil man im Allgemeinen keines der beiden anzugeben braucht.

- Python-Abhängigkeiten, die zur Laufzeit gebraucht werden, stehen im `propagated-inputs`-Feld. Solche werden typischerweise mit dem Schlüsselwort `install_requires` in `setup.py` oder in der Datei `requirements.txt` definiert.

- Python-Pakete, die nur zur Erstellungszeit gebraucht werden – z.B. jene, die in `pyproject.toml` unter `build-system.requires` stehen oder die mit dem Schlüsselwort `setup_requires` in `setup.py` aufgeführt sind – oder Abhängigkeiten, die nur zum Testen gebraucht werden – also die in `tests_require` oder in der `tox.ini` –, schreibt man in `native-inputs`. Die Begründung ist, dass (1) sie nicht propagiert werden müssen, weil sie zur Laufzeit nicht gebraucht werden, und (2) wir beim Cross-Kompilieren die „native“ Eingabe des Wirtssystems wollen.

  Beispiele sind die Testrahmen `pytest`, `mock` und `nose`. Wenn natürlich irgendeines dieser Pakete auch zur Laufzeit benötigt wird, muss es doch in `propagated-inputs` stehen.

- Alles, was nicht in die bisher genannten Kategorien fällt, steht in `inputs`, zum Beispiel Programme oder C-Bibliotheken, die zur Erstellung von Python-Paketen mit enthaltenen C-Erweiterungen gebraucht werden.

- Wenn ein Python-Paket optionale Abhängigkeiten hat (`extras_require`), ist es Ihnen überlassen, sie hinzuzufügen oder nicht hinzuzufügen, je nachdem wie es um deren Verhältnis von Nützlichkeit zu anderen Nachteilen steht (siehe Abschnitt 22.6 [Einreichen von Patches], Seite 724).

22.4.8 Perl-Module

Eigenständige Perl-Programme bekommen einen Namen wie jedes andere Paket, unter Nutzung des Namens beim Anbieter in Kleinbuchstaben. Für Perl-Pakete, die eine einzelne Klasse enthalten, ersetzen wir alle Vorkommen von `::` durch Striche und hängen davor das Präfix `perl-` an. Die Klasse `XML::Parser` wird also zu `perl-xml-parser`. Module, die mehrere Klassen beinhalten, behalten ihren Namen beim Anbieter, in Kleinbuchstaben gesetzt, und auch an sie wird vorne das Präfix `perl-` angehängt. Es gibt die Tendenz, solche Module mit dem Wort `perl` irgendwo im Namen zu versehen, das wird zu Gunsten des Präfixes weggelassen. Zum Beispiel wird aus `libwww-perl` bei uns `perl-libwww`.

22.4.9 Java-Pakete

Eigenständige Java-Programme werden wie jedes andere Paket benannt, d.h. mit ihrem in Kleinbuchstaben geschriebenen Namen beim Anbieter.

Um Verwirrungen und Namenskollisionen mit anderen Programmiersprachen zu vermeiden, ist es wünschenswert, dass dem Namem eines Pakets zu einem Java-Paket das Präfix `java-` vorangestellt wird. Wenn ein Projekt bereits das Wort `java` im Namen trägt, lassen wir es weg; zum Beispiel befindet sich das Java-Paket `ngsjava` in einem Paket namens `java-ngs`.

Bei Java-Paketen, die eine einzelne Klasse oder eine kleine Klassenhierarchie enthalten, benutzen wir den Klassennamen in Kleinbuchstaben und ersetzen dabei alle Vorkommen von `.` durch Striche und setzen das Präfix `java-` davor. Die Klasse `apache.commons.cli` wird also zum Paket `java-apache-commons-cli`.

22.4.10 Rust-Crates

Eigenständige Rust-Programme werden wie jedes andere Paket benannt, d.h. mit ihrem in Kleinbuchstaben geschriebenen Namen beim Anbieter.

Um Namensraumkollisionen vorzubeugen, versehen wir alle anderen Rust-Pakete mit dem Präfix `rust-`. Der Name sollte wie sonst in Kleinbuchstaben geschrieben werden und Bindestriche dort bleiben, wo sie sind.

Im Rust-Ökosystem werden oft mehrere inkompatible Versionen desselben Pakets gleichzeitig benutzt, daher sollte allen Paketdefinitionen ein die Version angebendes Suffix gegeben werden. Das Versionssuffix besteht aus der am weitesten links stehenden Ziffer, die *nicht* null ist (natürlich mit allen führenden Nullen). Dabei folgen wir dem von Cargo gewollten „Caret“-Versionsschema. Beispiele: `rust-clap-2`, `rust-rand-0.6`.

Weil die Verwendung von Rust-Paketen als vorab kompilierte Eingaben für andere Pakete besondere Schwierigkeiten macht, gibt es im Cargo-Erstellungssystem (siehe Abschnitt 9.5 [Erstellungssysteme], Seite 130) die Schlüsselwörter `#:cargo-inputs` und `#:cargo-development-inputs` als Argumente an das Erstellungssystem. Es ist hilfreich, sie sich ähnlich wie `propagated-inputs` und `native-inputs` vorzustellen. Was in Rust `dependencies` und `build-dependencies` sind, sollte unter `#:cargo-inputs` aufgeführt werden, während `dev-dependencies` zu den `#:cargo-development-inputs` gehören. Wenn ein Rust-Paket andere Bibliotheken einbindet, gilt die normale Einordnung in `inputs` usw. wie anderswo auch.

Man sollte aufpassen, dass man die richtige Version von Abhängigkeiten benutzt. Deswegen versuchen wir, Tests nicht mit `#:skip-build?` zu überspringen, wenn es möglich ist. Natürlich geht das nicht immer, vielleicht weil das Paket für ein anderes Betriebssystem entwickelt wurde, Funktionalitäten der allerneuesten „Nightly“-Version des Rust-Compilers voraussetzt oder weil der Testkatalog seit seiner Veröffentlichung verkümmert ist.

22.4.11 Elm-Pakete

Sie dürfen Elm-Anwendungen Namen geben wie bei anderer Software: Elm muss *nicht* im Namen vorkommen.

Die Namen dessen, was bei Elm als Paket bekannt ist (siehe `elm-build-system` im Unterabschnitt Abschnitt 9.5 [Erstellungssysteme], Seite 130), müssen dagegen diesem Format

folgen: *Autor/Projekt*, wo sowohl im *Autor* als auch im *Projekt* Bindestriche vorkommen können, und manchmal im *Autor* sogar Großbuchstaben enthalten sind.

Um daraus einen Guix-Paketnamen zu bilden, folgen wir einer ähnlichen Konvention wie bei Python-Paketen (siehe Abschnitt 22.4.7 [Python-Module], Seite 719), d.h. vorne kommt ein Präfix `elm-`, außer der Name beginnt ohnehin mit `elm-`.

In vielen Fällen lässt sich der Name eines Elm-Pakets, den es beim Anbieter trägt, heuristisch rekonstruieren, aber *nicht* immer, denn bei der Umwandlung in einen Namen im Guix-Stil geht Information verloren. Also achten Sie darauf, in solchen Fällen eine Eigenschaft `'upstream-name` anzugeben, damit '`guix import elm`' funktionieren kann (siehe Abschnitt 10.5 [Aufruf von guix import], Seite 206). Insbesondere ist es nötig, den Namen beim Anbieter ausdrücklich anzugeben, wenn:

1. Der *Autor* gleich `elm` ist und in *Projekt* ein oder mehrere Bindestriche auftauchen, wie bei `elm/virtual-dom`, oder wenn
2. In *Autor* Bindestriche oder Großbuchstaben auftauchen, z.B. `Elm-Canvas/raster-shapes` – sofern *Autor* nicht `elm-explorations` ist, was eine Sonderbehandlung bekommt, so dass Pakete wie `elm-explorations/markdown` die Eigenschaft `'upstream-name` *nicht* zu haben brauchen.

Im Modul `(guix build-system elm)` finden Sie folgende Werkzeuge, mit denen Sie mit den Konventionen für Namen und Ähnliches umgehen können:

`elm-package-origin` *`Elm-Name Version Hash`* *Liefert einen Git-Paketursprung nach den Regeln für* [Scheme-Prozedur]

Repository-Name und Tags, die für öffentliche Elm-Pakete vorgeschrieben sind. Der Name für den Paketursprung ergibt sich aus *Elm-Name*, das ist der Name beim Anbieter, mit der Version *Version* und SHA256-Prüfsumme *Hash*.

Zum Beispiel:

```
(package
  (name "elm-html")
  (version "1.0.0")
  (source
   (elm-package-origin
    "elm/html"
    version
    (base32 "15k1679ja57vvlpinpv06znmrxy09lbhzfkzdc89i01qa8c4gb4a")))
  ...)
```

`elm->package-name` *`Elm-Name`* [Scheme-Prozedur]

Liefert den Paketnamen im Guix-Stil für ein Elm-Paket, das dort *Elm-Name* heißt.

Achtung, `elm->package-name` kann unterschiedliche *Elm-Name* auf dasselbe Ergebnis abbilden.

`guix-package->elm-name` *`Paket`* [Scheme-Prozedur]

Für ein Elm-Paket *Paket* wird ermittelt, welchen Namen es beim Anbieter trägt, oder `#f`, wenn dieser Anbietername weder in den unter *properties* aufgeführten Paketeigenschaften steht noch sich mit `infer-elm-package-name` ableiten lässt.

`infer-elm-package-name` *Guix-Name* [Scheme-Prozedur]
: Liefert für den *guix-name* eines Elm-Pakets den daraus abgeleiteten Namen beim Anbieter oder `#f`, wenn er sich nicht ableiten lässt. Wenn das Ergebnis etwas anderes als `#f` ist, können wir es an `elm->package-name` übergeben und bekommen wieder *guix-name* heraus.

22.4.12 Schriftarten

Wenn Schriftarten in der Regel nicht von Nutzern zur Textbearbeitung installiert werden oder als Teil eines größeren Software-Pakets mitgeliefert werden, gelten dafür die allgemeinen Paketrichtlinien für Software. Zum Beispiel trifft das auf als Teil des X.Org-Systems ausgelieferte Schriftarten zu, oder auf Schriftarten, die ein Teil von TeX Live sind.

Damit es Nutzer leichter haben, nach Schriftarten zu suchen, konstruieren wir die Namen von anderen Paketen, die nur Schriftarten enthalten, nach dem folgenden Schema, egal was der Paketname beim Anbieter ist.

Der Name eines Pakets, das nur eine Schriftfamilie enthält, beginnt mit `font-`. Darauf folgt der Name des Schriftenherstellers und ein Strich `-`, sofern bekannt ist, wer der Schriftenhersteller ist, und dann der Name der Schriftfamilie, in dem Leerzeichen durch Striche ersetzt werden (und wie immer mit Großbuchstaben statt Kleinbuchstaben). Zum Beispiel befindet sich die von SIL hergestellte Gentium-Schriftfamilie im Paket mit dem Namen `font-sil-gentium`.

Wenn ein Paket mehrere Schriftfamilien enthält, wird der Name der Sammlung anstelle des Schriftfamiliennamens benutzt. Zum Beispiel umfassen die Liberation-Schriftarten drei Familien: Liberation Sans, Liberation Serif und Liberation Mono. Man könnte sie getrennt voneinander mit den Namen `font-liberation-sans` und so weiter in Pakete einteilen. Da sie aber unter einem gemeinsamen Namen angeboten werden, packen wir sie lieber zusammen in ein Paket mit dem Namen `font-liberation`.

Für den Fall, dass mehrere Formate derselben Schriftfamilie oder Schriftartensammlung in separate Pakete kommen, wird ein Kurzname für das Format mit einem Strich vorne zum Paketnamen hinzugefügt. Wir benutzen `-ttf` für TrueType-Schriftarten, `-otf` für OpenType-Schriftarten und `-type1` für PostScript-Typ-1-Schriftarten.

22.5 Programmierstil

Im Allgemeinen folgt unser Code den GNU Coding Standards (siehe *GNU Coding Standards*). Da diese aber nicht viel über Scheme zu sagen haben, folgen hier einige zusätzliche Regeln.

22.5.1 Programmierparadigmen

Scheme-Code wird in Guix auf rein funktionale Weise geschrieben. Eine Ausnahme ist Code, der mit Ein- und Ausgabe zu tun hat, und Prozeduren, die grundlegende Konzepte implementieren, wie zum Beispiel die Prozedur `memoize`.

22.5.2 Module

Guile-Module, die beim Erstellen nutzbar sein sollen, müssen im Namensraum `(guix build ...)` leben. Sie dürfen auf keine anderen Guix- oder GNU-Module Bezug nehmen. Jedoch ist es in Ordnung, wenn ein „wirtsseitiges“ Modul ein erstellungsseitiges Modul benutzt.

Module, die mit dem weiteren GNU-System zu tun haben, sollten im Namensraum `(gnu ...)` und nicht in `(guix ...)` stehen.

22.5.3 Datentypen und Mustervergleich

Im klassischen Lisp gibt es die Tendenz, Listen zur Darstellung von allem zu benutzen, und diese dann „händisch“ zu durchlaufen mit `car`, `cdr`, `cadr` und so weiter. Dieser Stil ist aus verschiedenen Gründen problematisch, insbesondere wegen der Tatsache, dass er schwer zu lesen, schnell fehlerbehaftet und ein Hindernis beim Melden von Typfehlern ist.

Guix-Code sollte angemessene Datentypen definieren (zum Beispiel mit `define-record-type*`), statt Listen zu missbrauchen. Außerdem sollte er das `(ice-9 match)`-Modul von Guile zum Mustervergleich benutzen, besonders mit Listen (siehe Abschnitt “Pattern Matching” in *Referenzhandbuch zu GNU Guile*), während bei Verbundstypen `match-record` aus dem Modul `(guix records)` angemessener ist, womit, anders als bei `match`, bereits bei der Makroumschreibung sichergestellt wird, dass die Feldnamen richtig sind.

22.5.4 Formatierung von Code

Beim Schreiben von Scheme-Code halten wir uns an die üblichen Gepflogenheiten unter Scheme-Programmierern. Im Allgemeinen bedeutet das, dass wir uns an Riastradh's Lisp Style Rules (`https://mumble.net/~campbell/scheme/style.txt`) halten. Es hat sich ergeben, dass dieses Dokument auch die Konventionen beschreibt, die im Code von Guile hauptsächlich verwendet werden. Es ist gut durchdacht und schön geschrieben, also lesen Sie es bitte.

Ein paar in Guix eingeführte Sonderformen, wie zum Beispiel das `substitute*`-Makro, haben abweichende Regeln für die Einrückung. Diese sind in der Datei `.dir-locals.el` definiert, die Emacs automatisch benutzt. Beachten Sie auch, dass Emacs-Guix einen Modus namens `guix-devel-mode` bereitstellt, der Guix-Code richtig einrückt und hervorhebt (siehe Abschnitt “Development” in *Referenzhandbuch von Emacs-Guix*).

Falls Sie nicht Emacs verwenden, sollten Sie sicherstellen, dass Ihr Editor diese Regeln kennt. Um eine Paketdefinition automatisch einzurücken, können Sie auch Folgendes ausführen:

```
./pre-inst-env guix style Paket
```

Siehe Abschnitt 10.7 [Aufruf von guix style], Seite 221, für weitere Informationen.

Wenn Sie Code mit Vim bearbeiten, empfehlen wir, dass Sie `:set autoindent` ausführen, damit Ihr Code automatisch eingerückt wird, während Sie ihn schreiben. Außerdem könnte Ihnen `paredit.vim` (`https://www.vim.org/scripts/script.php?script_id=3998`) dabei helfen, mit all diesen Klammern fertigzuwerden.

Wir fordern von allen Prozeduren auf oberster Ebene, dass sie über einen Docstring verfügen. Diese Voraussetzung kann jedoch bei einfachen, privaten Prozeduren im Namensraum `(guix build ...)` aufgeweicht werden.

Prozeduren sollten nicht mehr als vier positionsbestimmte Parameter haben. Benutzen Sie Schlüsselwort-Parameter für Prozeduren, die mehr als vier Parameter entgegennehmen.

22.6 Einreichen von Patches

Die Entwicklung wird mit Hilfe des verteilten Versionskontrollsystems Git durchgeführt. Daher ist eine ständige Verbindung zum Repository nicht unbedingt erforderlich. Wir begrüßen

Beiträge in Form von Patches, die mittels `git format-patch` erstellt und an die Mailingliste `guix-patches@gnu.org` geschickt werden (siehe Abschnitt "Submitting patches to a project" in *Git-Benutzerhandbuch*). In diesem Fall möchten wir Ihnen nahelegen, zunächst einige Git-Repository-Optionen festzulegen (siehe Abschnitt 22.6.1 [Git einrichten], Seite 728), damit Ihr Patch leichter lesbar wird. Erfahrene Guix-Entwickler möchten vielleicht auch einen Blick auf den Abschnitt über Commit-Zugriff werfen (siehe Abschnitt 22.8 [Commit-Zugriff], Seite 733).

Diese Mailing-Liste setzt auf einer Debbugs-Instanz auf, wodurch wir den Überblick über Eingereichtes behalten können (siehe Abschnitt 22.7 [Überblick über gemeldete Fehler und Patches], Seite 731). Jede an diese Mailing-Liste gesendete Nachricht bekommt eine neue Folgenummer zugewiesen, so dass man eine Folge-E-Mail zur Einreichung an *FEHLERNUMMER*`@debbugs.gnu.org` senden kann, wobei *FEHLERNUMMER* für die Folgenummer steht (siehe Abschnitt 22.6.2 [Senden einer Patch-Reihe], Seite 728).

Bitte schreiben Sie Commit-Logs im ChangeLog-Format (siehe Abschnitt "Change Logs" in *GNU Coding Standards*); dazu finden Sie Beispiele unter den bisherigen Commits.

Bevor Sie einen Patch einreichen, der eine Paketdefinition hinzufügt oder verändert, gehen Sie bitte diese Prüfliste durch:

1. Wenn die Autoren der verpackten Software eine kryptografische Signatur bzw. Beglaubigung für den Tarball der Veröffentlichung anbieten, so machen Sie sich bitte die Mühe, die Echtheit des Archivs zu überprüfen. Für eine abgetrennte GPG-Signaturdatei würden Sie das mit dem Befehl `gpg --verify` tun.
2. Nehmen Sie sich die Zeit, eine passende Zusammenfassung und Beschreibung für das Paket zu verfassen. Unter Abschnitt 22.4.4 [Zusammenfassungen und Beschreibungen], Seite 717, finden Sie dazu einige Richtlinien.
3. Verwenden Sie `guix lint` *Paket*, wobei *Paket* das neue oder geänderte Paket bezeichnet, und beheben Sie alle gemeldeten Fehler (siehe Abschnitt 10.8 [Aufruf von guix lint], Seite 223).
4. Verwenden Sie `guix style` *Paket*, um die neue Paketdefinition gemäß den Konventionen des Guix-Projekts zu formatieren (siehe Abschnitt 10.7 [Aufruf von guix style], Seite 221).
5. Stellen Sie sicher, dass das Paket auf Ihrer Plattform erstellt werden kann, indem Sie `guix build` *Paket* ausführen.
6. Wir empfehlen, dass Sie auch versuchen, das Paket auf anderen unterstützten Plattformen zu erstellen. Da Sie vielleicht keinen Zugang zu echter Hardware für diese Plattformen haben, empfehlen wir, den `qemu-binfmt-service-type` zu benutzen, um sie zu emulieren. Um ihn zu aktivieren, fügen Sie `virtualization` zu `use-service-modules` und den folgenden Dienst in die Liste der Dienste („services“) in Ihrer `operating-system`-Konfiguration ein:

   ```
   (service qemu-binfmt-service-type
    (qemu-binfmt-configuration
      (platforms (lookup-qemu-platforms "arm" "aarch64"))))
   ```

   Rekonfigurieren Sie anschließend Ihr System.

   Sie können Pakete für andere Plattformen erstellen lassen, indem Sie die Befehlszeilenoption `--system` angeben. Um zum Beispiel das Paket „hello“ für die Architekturen

armhf oder aarch64 erstellen zu lassen, würden Sie jeweils die folgenden Befehle angeben:

```
guix build --system=armhf-linux --rounds=2 hello
guix build --system=aarch64-linux --rounds=2 hello
```

7. Achten Sie darauf, dass im Paket keine Software gebündelt mitgeliefert wird, die bereits in separaten Paketen zur Verfügung steht.

   Manchmal enthalten Pakete Kopien des Quellcodes ihrer Abhängigkeiten, um Nutzern die Installation zu erleichtern. Als eine Distribution wollen wir jedoch sicherstellen, dass solche Pakete die schon in der Distribution verfügbare Fassung benutzen, sofern es eine gibt. Dadurch wird sowohl der Ressourcenverbrauch optimiert (die Abhängigkeit wird so nur einmal erstellt und gespeichert) als auch der Distribution die Möglichkeit gegeben, ergänzende Änderungen durchzuführen, um beispielsweise Sicherheitsaktualisierungen für ein bestimmtes Paket an nur einem Ort einzuspielen, die aber das gesamte System betreffen – gebündelt mitgelieferte Kopien würden dies verhindern.

8. Schauen Sie sich das von `guix size` ausgegebene Profil an (siehe Abschnitt 10.9 [Aufruf von guix size], Seite 226). Dadurch können Sie Referenzen auf andere Pakete finden, die ungewollt vorhanden sind. Dies kann auch dabei helfen, zu entscheiden, ob das Paket aufgespalten werden sollte (siehe Abschnitt 6.4 [Pakete mit mehreren Ausgaben.], Seite 60) und welche optionalen Abhängigkeiten verwendet werden sollten. Dabei sollten Sie es wegen seiner enormen Größe insbesondere vermeiden, `texlive` als eine Abhängigkeit hinzuzufügen; benutzen Sie stattdessen das Paket `texlive-tiny` oder die Prozedur `texlive-union`.

9. Achten Sie bei wichtigen Änderungen darauf, dass abhängige Pakete (falls vorhanden) nicht von der Änderung beeinträchtigt werden; `guix refresh --list-dependent` *`Paket`* hilft Ihnen dabei (siehe Abschnitt 10.6 [Aufruf von guix refresh], Seite 214).

   Je nachdem, wie viele abhängige Pakete es gibt, und entsprechend wie viele Neuerstellungen dadurch nötig würden, finden Commits auf anderen Branches statt, nach ungefähr diesen Regeln:

   300 abhängige Pakete oder weniger
   : `master`-Branch (störfreie Änderungen).

   zwischen 300 und 1.800 abhängige Pakete
   : `staging`-Branch (störfreie Änderungen). Dieser Branch wird circa alle 6 Wochen mit `master` zusammengeführt. Themenbezogene Änderungen (z.B. eine Aktualisierung der GNOME-Plattform) können stattdessen auch auf einem eigenen Branch umgesetzt werden (wie `gnome-updates`). Dieser Branch ist erst spät im Entwicklungsprozess nutzbar.

   mehr als 1.800 abhängige Pakete
   : `core-updates`-Branch (kann auch größere und womöglich andere Software beeinträchtigende Änderungen umfassen). Dieser Branch wird planmäßig in `master` alle 6 Monate oder so gemerget. Dieser Branch ist erst spät im Entwicklungsprozess nutzbar.

   All diese Branches werden kontinuierlich auf unserer Erstellungsfarm (`https://ci.guix.gnu.org`) erstellt und in `master` gemerget, sobald alles erfolgreich erstellt worden ist. Dadurch können wir Probleme beheben, bevor sie bei Nutzern auftreten, und

zudem das Zeitfenster, während dessen noch keine vorerstellten Binärdateien verfügbar sind, verkürzen.

Sobald wir uns dazu entscheiden, einen der Branches `staging` oder `core-updates` zu erstellen, spalten wir einen neuen Branch davon ab und hängen an seinen Namen das Suffix `-frozen` an. Auf einen „frozen“-Branch dürfen dann nur noch Fehlerbehebungen gepusht werden. Die Branches `core-updates` und `staging` bleiben offen; dorthin gehen Patches für den nächsten Durchlauf. Bitte fragen Sie auf der Mailing-Liste oder im IRC nach, wenn Sie sich unsicher sind, wohin ein Patch gehört.

10. Überprüfen Sie, ob der Erstellungsprozess deterministisch ist. Dazu prüfen Sie typischerweise, ob eine unabhängige Erstellung des Pakets genau dasselbe Ergebnis wie Ihre Erstellung hat, Bit für Bit.

    Dies können Sie leicht tun, indem Sie dasselbe Paket mehrere Male hintereinander auf Ihrer Maschine erstellen (siehe Abschnitt 10.1 [Aufruf von guix build], Seite 189):

    ```
    guix build --rounds=2 mein-paket
    ```

    Dies reicht aus, um eine ganze Klasse häufiger Ursachen von Nichtdeterminismus zu finden, wie zum Beispiel Zeitstempel oder zufallsgenerierte Ausgaben im Ergebnis der Erstellung.

    Eine weitere Möglichkeit ist, `guix challenge` (siehe Abschnitt 10.12 [Aufruf von guix challenge], Seite 238) zu benutzen. Sie können es ausführen, sobald ein Paket commitet und von `ci.guix.gnu.org` erstellt wurde, um zu sehen, ob dort dasselbe Ergebnis wie bei Ihnen geliefert wurde. Noch besser: Finden Sie eine andere Maschine, die das Paket erstellen kann, und führen Sie `guix publish` aus. Da sich die entfernte Erstellungsmaschine wahrscheinlich von Ihrer unterscheidet, können Sie auf diese Weise Probleme durch Nichtdeterminismus erkennen, die mit der Hardware zu tun haben – zum Beispiel die Nutzung anderer Befehlssatzerweiterungen – oder mit dem Betriebssystem-Kernel – zum Beispiel, indem `uname` oder `/proc`-Dateien verwendet werden.

11. Beim Schreiben von Dokumentation achten Sie bitte auf eine geschlechtsneutrale Wortwahl, wenn Sie sich auf Personen beziehen, wie „they“, „their“, „them“ im Singular (`https://en.wikipedia.org/wiki/Singular_they`) und so weiter.

12. Stellen Sie sicher, dass Ihr Patch nur einen Satz zusammengehöriger Änderungen umfasst. Das Zusammenfassen nicht zusammengehöriger Änderungen erschwert und bremst das Durchsehen Ihres Patches.

    Beispiele für nicht zusammengehörige Änderungen sind das Hinzufügen mehrerer Pakete auf einmal, oder das Aktualisieren eines Pakets auf eine neue Version zusammen mit Fehlerbehebungen für das Paket.

13. Bitte befolgen Sie unsere Richtlinien für die Code-Formatierung; womöglich wollen Sie dies automatisch tun lassen durch das Skript `guix style` (siehe Abschnitt 22.5.4 [Formatierung von Code], Seite 724).

14. Benutzen Sie, wenn möglich, Spiegelserver (Mirrors) in der Quell-URL (siehe Abschnitt 10.3 [Aufruf von guix download], Seite 204). Verwenden Sie verlässliche URLs, keine automatisch generierten. Zum Beispiel sind Archive von GitHub nicht immer identisch von einer Generation auf die nächste, daher ist es in diesem Fall besser, als Quelle einen Klon des Repositorys zu verwenden. Benutzen Sie *nicht* das `name`-Feld beim Angeben der URL; er hilft nicht wirklich und wenn sich der Name ändert, stimmt die URL nicht mehr.

15. Überprüfen Sie, ob Guix erstellt werden kann (siehe Abschnitt 22.1 [Erstellung aus dem Git], Seite 708) und kümmern Sie sich um die Warnungen, besonders um solche über nicht definierte Symbole.
16. Stellen Sie sicher, dass Ihre Änderungen Guix nicht beeinträchtigen, und simulieren Sie eine Ausführung von `guix pull` über den Befehl:

```
guix pull --url=/pfad/zu/ihrem/checkout --profile=/tmp/guix.master
```

Bitte benutzen Sie '`[PATCH] ...`' als Betreff, wenn Sie einen Patch an die Mailing-Liste schicken. Soll Ihr Patch auf einen anderen Branch als `master` angewandt werden, z.B. `core-updates`, geben Sie dies im Betreff an als '`[PATCH core-updates] ...`'.

Sie können dazu Ihr E-Mail-Programm oder den Befehl `git send-email` benutzen (siehe Abschnitt 22.6.2 [Senden einer Patch-Reihe], Seite 728). Wir bevorzugen es, Patches als reine Textnachrichten zu erhalten, entweder eingebettet (inline) oder als MIME-Anhänge. Sie sind dazu angehalten, zu überprüfen, ob Ihr Mail-Programm solche Dinge wie Zeilenumbrüche oder die Einrückung verändert, wodurch die Patches womöglich nicht mehr funktionieren.

Rechnen Sie damit, dass es etwas dauert, bis Ihr erster Patch an `guix-patches@gnu.org` zu sehen ist. Sie werden warten müssen, bis Sie eine Bestätigung mit der zugewiesenen Folgenummer bekommen. Spätere Bestätigungen sollten sofort kommen.

Wenn dadurch ein Fehler behoben wurde, schließen Sie bitte den Thread, indem Sie eine E-Mail an *`FEHLERNUMMER`*`-done@debbugs.gnu.org` senden.

22.6.1 Git einrichten

Wenn es noch nicht geschehen ist, wollen Sie vielleicht den Namen und die E-Mail-Adresse festlegen, mit der Ihre Commits ausgestellt werden (siehe Abschnitt "Telling Git your name" in *Git-Benutzerhandbuch*). Wenn Sie einen anderen Namen oder eine andere E-Mail-Adresse nur für Commits in diesem Repository verwenden möchten, können Sie `git config --local` benutzen oder Änderungen in der Datei `.git/config` im Repository statt in `~/.gitconfig` durchführen.

Wir stellen einige Standardeinstellungen in `etc/git/gitconfig` zur Verfügung, die ändern, wie Patches erzeugt werden, damit sie leichter zu lesen und anzuwenden sind. Sie können die Einstellungen von Hand übernehmen, indem Sie sie in die Datei `.git/config` in Ihrem Checkout kopieren, oder Git anweisen, die ganze Datei per `include`-Direktive zu laden:

```
git config --local include.path ../etc/git/gitconfig
```

Von da an werden sich jegliche Änderungen an `etc/git/gitconfig` automatisch auswirken.

Weil der erste Patch in einer Patch-Reihe getrennt versandt werden muss (siehe Abschnitt 22.6.2 [Senden einer Patch-Reihe], Seite 728), kann es helfen, `git format-patch` statt `git send-email` mit der Nachrichtenverkettung zu beauftragen:

```
git config --local format.thread shallow
git config --local sendemail.thread no
```

22.6.2 Senden einer Patch-Reihe

Einzelne Patches

Der Befehl `git send-email` ist die beste Art, wie Sie sowohl einzelne Patches als auch Patch-Reihen (siehe [Mehrere Patches], Seite 730) an die Guix-Mailing-Liste schicken können. Würden Sie Patches als E-Mail-Anhänge schicken, wäre es in manchen Mailprogrammen umständlich, diese zu überprüfen. Und wenn Sie aus `git diff` kopierten, würden die Metadaten des Commits fehlen.

> **Anmerkung:** Der Befehl `git send-email` ist in der Ausgabe namens `send-email` des `git`-Pakets enthalten, also `git:send-email`.

Mit dem folgenden Befehl erzeugen Sie eine E-Mail mit dem Patch aus dem neuesten Commit, öffnen diese in Ihrem gewählten *EDITOR* oder *VISUAL*, um sie zu bearbeiten, und schicken sie an die Guix-Mailing-Liste, damit jemand sie überprüft und merget:

```
$ git send-email -1 -a --base=auto --to=guix-patches@gnu.org
```

> **Tipp:** Wenn Sie in der Betreffzeile zu Ihrem Patch ein zusätzliches Präfix mitgeben möchten, können Sie die Befehlszeilenoption `--subject-prefix` benutzen. Beim Guix-Projekt wird so angegeben, dass der Patch für einen bestimmten Branch oder ein anderes Repository bestimmt ist, statt für den `master`-Branch von `https://git.savannah.gnu.org/cgit/guix.git`.
>
> ```
> git send-email -1 -a --base=auto \
>     --subject-prefix='PATCH core-updates' \
>     --to=guix-patches@gnu.org
> ```

In der Patch-E-Mail finden Sie eine Trennlinie aus drei Bindestrichen unter der Commit-Nachricht. Sie dürfen erklärende Bemerkungen zum Patch unterhalb dieser Linie anbringen. Wenn Sie keine solchen Annotationen in der E-Mail schreiben möchten, können Sie oben auf die Befehlszeilenoption `-a` (der Abkürzung von `--annotate`) verzichten.

Mit der Befehlszeilenoption `--base=auto` wird automatisch eine Bemerkung zum Ende des Patches hinzugefügt, die angibt, auf welchem Commit der Patch basiert. Damit fällt es Betreuern leichter, Ihre Patches mit `rebase` zu übernehmen und zu mergen.

Wenn Sie einen überarbeiteten Patch schicken müssen, geht das anders. Machen Sie es *nicht* so und schicken Sie *keinen* Patch mit einem „Fix“, der als Nächstes angewandt werden müsste, sondern verwenden Sie stattdessen `git commit -a` oder `git rebase` (`https://git-rebase.io`), um den alten Commit zu verändern, und schicken den an die Adresse *FEHLERNUMMER*`@debbugs.gnu.org`, wobei Sie außerdem die Befehlszeilenoption `-v` von `git send-email` angeben.

```
$ git commit -a
$ git send-email -1 -a --base=auto -v REVISION \
      --to=FEHLERNUMMER@debbugs.gnu.org
```

Die *FEHLERNUMMER* finden Sie heraus, indem Sie entweder auf der Mumi-Oberfläche auf `issues.guix.gnu.org` nach dem Namen Ihres Patches suchen oder indem Sie in Ihren E-Mails auf die Eingangsbestätigung schauen, die Debbugs Ihnen automatisch als Antwort auf eingehende Fehlerberichte (Bugs) und Patches hat zukommen lassen. Darin finden Sie die Fehlernummer.

Teams ansprechen

Mit dem Skript `etc/teams.scm` können Sie direkt all die Leute in Kenntnis setzen, die sich für Ihren Patch interessieren könnten (siehe Abschnitt 22.6.3 [Teams], Seite 731). Be-

nutzen Sie `etc/teams.scm list-teams`, um alle Teams angezeigt zu bekommen, damit Sie entscheiden können, welches Team oder welche Teams Ihr Patch betrifft, und sich mit `etc/teams.scm cc` die geeigneten Befehlszeilenoptionen für `git send-email` ausgeben lassen, um die richtigen Teammitglieder anzuschreiben; alternativ werden mit `etc/teams.scm cc-members` die Teams automatisch ermittelt.

Mehrere Patches

Obwohl `git send-email` allein für einen einzelnen Patch genügt, führt eine Unzulänglichkeit in Debbugs dazu, dass Sie beim Versenden mehrerer Patches achtgeben müssen: Wenn Sie alle auf einmal an die Adresse `guix-patches@gnu.org` schicken würden, würde für jeden Patch jeweils ein Fehlerbericht eröffnet!

Wenn Sie die Patches aber als Patch-Reihe abschicken wollen, sollten Sie als Erstes mit Git ein „Deckblatt“ schicken, das den Gutachtern einen Überblich über die Patch-Reihe gibt. Zum Beispiel können Sie ein Verzeichnis namens `ausgehend` anlegen und darin sowohl Ihre Patch-Reihe als auch ein Deckblatt namens `0000-cover-letter.patch` platzieren, indem Sie `git format-patch` aufrufen.

```
$ git format-patch -ANZAHL_COMMITS -o ausgehend \
      --cover-letter --base=auto
```

Jetzt schicken Sie *nur* das Deckblatt an die Adresse `guix-patches@gnu.org`, so dass ein Fehlerbericht aufgemacht wird, an den wir dann die restlichen Patches schicken können.

```
$ git send-email ausgehend/0000-cover-letter.patch -a \
      --to=guix-patches@debbugs.gnu.org \
      $(etc/teams.scm cc-members ...)
$ rm ausgehend/0000-cover-letter.patch # nicht nochmal schicken!
```

Passen Sie auf und bearbeiten die E-Mail nochmal, um ihr vor dem Abschicken eine angemessene Betreffzeile (Subject) und anstelle von Blurb Ihren Text mitzugeben. Sie werden bemerken, dass unterhalb des Textes automatisch ein Shortlog und Diffstat aufgelistet wurden.

Sobald Sie vom Debbugs-Mailer eine Antwort auf Ihre Deckblatt-E-Mail erhalten haben, können Sie die eigentlichen Patches an die neu erzeugte Adresse für diesen Fehlerbericht senden.

```
$ git send-email ausgehend/*.patch \
      --to=FEHLERNUMMER@debbugs.gnu.org \
      $(etc/teams.scm cc-members ...)
$ rm -rf ausgehend # wir sind damit fertig
```

Zum Glück können wir uns diesen Tanz mit `git format-patch` danach sparen, wenn wir eine überarbeitete Patch-Reihe schicken, weil wir die Fehlernummer dann bereits haben.

```
$ git send-email -ANZAHL_COMMITS \
      -vREVISION --base=auto \
      --to FEHLERNUMMER@debbugs.gnu.org
```

Wenn nötig, können Sie mit `--cover-letter -a` auch dann ein weiteres Deckblatt mitschicken, z.B. um zu erklären, was die Änderungen seit der letzten Revision sind und warum sie nötig waren.

22.6.3 Teams

Der Quellcode von Guix ist unter mehreren Mentorenteams aufgeteilt. Um sich eine Liste aller Teams anzeigen zu lassen, führen Sie aus einem Guix-Checkout Folgendes aus:

```
$ ./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>
...
```

Sie können den folgenden Befehl benutzen, um das Team `Mentors` bei einer Patch-Reihe in CC zu setzen:

```
$ git send-email --to FEHLERNUMMER@debbugs.gnu.org $(./etc/teams.scm cc mentors) *.pat
```

Das zuständige Team kann auch automatisch anhand der geänderten Dateien ermittelt werden. Wenn Sie zum Beispiel für die letzten zwei Commits im aktuellen Git-Repository um Überprüfung bitten möchten, führen Sie dies aus:

```
$ guix shell -D guix
[env]$ git send-email --to FEHLERNUMMER@debbugs.gnu.org $(./etc/teams.scm cc-members H
```

22.7 Überblick über gemeldete Fehler und Patches

Dieser Abschnitt beschreibt, wie das Guix-Projekt Fehlerberichte und eingereichte Patches verwaltet.

22.7.1 Der Issue-Tracker

Einen Überblick über gemeldete Fehler („Bugs“) und eingereichte Patches finden Sie derzeit auf der Debbugs-Instanz unter `https://bugs.gnu.org`. Fehler werden für das „Paket“ (so sagt man im Sprachgebrauch von Debbugs) namens `guix` gemeldet, indem Sie eine E-Mail an `bug-guix@gnu.org` schicken. Dagegen werden Patches für das Paket `guix-patches` eingereicht, indem Sie eine E-Mail an `guix-patches@gnu.org` schicken (siehe Abschnitt 22.6 [Einreichen von Patches], Seite 724).

22.7.2 Debbugs-Benutzerschnittstellen

Ihnen steht eine Weboberfläche (tatsächlich sogar *zwei* Weboberflächen!) zur Verfügung, um die Fehlerdatenbank zu durchsuchen:

- `https://issues.guix.gnu.org` erlaubt es Ihnen, über eine hübsche Schnittstelle[2] eingesendete Fehlerberichte („Bug Reports“) und Patches einzusehen und an Diskussionen teilzunehmen.

[2] Die Weboberfläche unter `https://issues.guix.gnu.org` läuft über das Programm Mumi, ein schönes Stück in Guile geschriebene Software, bei der Sie uns helfen können! Siehe `https://git.elephly.net/gitweb.cgi?p=software/mumi.git`.

- Auf `https://bugs.gnu.org/guix` werden gemeldete Fehler aufgeführt,
- auf `https://bugs.gnu.org/guix-patches` eingereichte Patches.

Um Diskussionen zum Fehler mit Fehlernummer *n* einzusehen, schauen Sie auf '`https://issues.guix.gnu.org/n`' oder '`https://bugs.gnu.org/n`'.

Wenn Sie Emacs benutzen, finden Sie es vielleicht bequemer, sich durch Nutzung von `debbugs.el` mit Fehlern zu befassen, was Sie mit folgendem Befehl installieren können:

```
guix install emacs-debbugs
```

Um zum Beispiel alle noch ausstehenden, „offenen“ Fehler bezüglich `guix-patches` anzusehen, geben Sie dies ein:

```
C-u M-x debbugs-gnu RET RET guix-patches RET n y
```

Siehe *Debbugs User Guide* für weitere Informationen zu diesem raffinierten Werkzeug.

22.7.3 Debbugs-Usertags

Debbugs bietet die Möglichkeit, sogenannte *Usertags* zu vergeben, d.h. jeder Benutzer kann jedem Fehlerbericht beliebige Kennzeichnungen zuweisen. Die gemeldeten Fehler können anhand der Usertags gesucht werden, deshalb lassen sich die Meldungen so gut organisieren[3].

Wenn Sie sich zum Beispiel alle Fehlerberichte (oder Patches, wenn Sie `guix-patches` betrachten) mit dem Usertag `powerpc64le-linux` für Benutzer `guix` ansehen möchten, öffnen Sie eine URL wie die folgende in einem Webbrowser: `https://debbugs.gnu.org/cgi-bin/pkgreport.cgi?tag=powerpc64le-linux;users=guix`.

Mehr Informationen, wie Sie Usertags benutzen können, finden Sie in der Dokumentation von Debbugs bzw. wenn Sie ein externes Programm benutzen, um mit Debbugs zu interagieren, in der Dokumentation dieses Programms.

In Guix experimentieren wir damit, Usertags zum Beobachten von Fehlern zu verwenden, die nur bestimmte Architekturen betreffen. Zur leichteren Zusammenarbeit verbinden wir all unsere Usertags einzig mit dem Benutzer `guix`. Für diesen Benutzer gibt es zurzeit folgende Usertags:

`powerpc64le-linux`
: Mit diesem Usertag wollen wir es leichter machen, die Fehler zu finden, die für den Systemtyp `powerpc64le-linux` am wichtigsten sind. Bitte weisen Sie ihn solchen Bugs oder Patches zu, die nur `powerpc64le-linux` und keine anderen Systemtypen betreffen. Des Weiteren können Sie damit Probleme kennzeichnen, die für den Systemtyp `powerpc64le-linux` besonders bedeutsam sind, selbst wenn das Problem sich auch bei anderen Systemtypen zeigt.

`Reproduzierbarkeit`
: Verwenden Sie den Usertag `reproducibility` für Fehler in der Reproduzierbarkeit einer Erstellung. Es wäre zum Beispiel angemessen, den Usertag für einen Fehlerbericht zu einem Paket zuzuweisen, das nicht reproduzierbar erstellt wird.

Wenn Sie Commit-Zugriff haben und einen neuen Usertag verwenden möchten, dann fangen Sie einfach an, ihn mit dem Benutzer `guix` zu benutzen. Erweist sich der Usertag für Sie als nützlich, sollten Sie vielleicht diesen Handbuchabschnitt ergänzen, um andere in Kenntnis zu setzen, was Ihr Usertag bedeutet.

[3] Die Liste der Usertags ist öffentlich zugänglich und jeder kann die Usertag-Liste jedes anderen Nutzers ändern. Sie sollten daran denken, wenn Sie diese Funktionalität gebrauchen möchten.

22.8 Commit-Zugriff

Jeder kann bei Guix mitmachen, auch ohne Commit-Zugriff zu haben (siehe Abschnitt 22.6 [Einreichen von Patches], Seite 724). Für Leute, die häufig zu Guix beitragen, kann es jedoch praktischer sein, Schreibzugriff auf das Repository zu haben. Als Daumenregel sollte ein Mitwirkender 50-mal überprüfte Commits eingebracht haben, um als Committer zu gelten, und schon mindestens 6 Monate im Projekt aktiv sein. Dadurch gab es genug Zusammenarbeit, um Mitwirkende einzuarbeiten und einzuschätzen, ob sie so weit sind, Committer zu werden. Dieser Commit-Zugriff sollte nicht als Auszeichnung verstanden werden, sondern als Verantwortung, die ein Mitwirkender auf sich nimmt, um dem Projekt zu helfen.

Im folgenden Abschnitt wird beschrieben, wie jemand Commit-Zugriff bekommt, was man tun muss, bevor man zum ersten Mal einen Commit pusht, und welche Richtlinien und Erwartungen die Gemeinschaft an Commits richtet, die auf das offizielle Repository gepusht werden.

22.8.1 Um Commit-Zugriff bewerben

Wenn Sie es für angemessen halten, dann sollten Sie in Erwägung ziehen, sich wie folgt um Commit-Zugriff zu bewerben:

1. Finden Sie drei Committer, die für Sie eintreten. Sie können die Liste der Committer unter `https://savannah.gnu.org/project/memberlist.php?group=guix` finden. Jeder von ihnen sollte eine entsprechende Erklärung per E-Mail an `guix-maintainers@gnu.org` schicken (eine private Alias-Adresse für das Kollektiv aus allen Betreuern bzw. Maintainern), die jeweils mit ihrem OpenPGP-Schlüssel signiert wurde.

   Von den Committern wird erwartet, dass sie bereits mit Ihnen als Mitwirkendem zu tun hatten und beurteilen können, ob Sie mit den Richtlinien des Projekts hinreichend vertraut sind. Dabei geht es *nicht* darum, wie wertvoll Ihre Beiträge sind, daher sollte eine Ablehnung mehr als „schauen wir später nochmal“ verstanden werden.

2. Senden Sie eine Nachricht an `guix-maintainers@gnu.org`, in der Sie Ihre Bereitschaft darlegen und die Namen der drei Committer nennen, die Ihre Bewerbung unterstützen. Die Nachricht sollte mit dem OpenPGP-Schlüssel signiert sein, mit dem Sie später auch Ihre Commits signieren, und Sie sollten den Fingerabdruck hinterlegen (siehe unten). Siehe `https://emailselfdefense.fsf.org/de/` für eine Einführung in asymmetrische Kryptografie („Public-Key“) mit GnuPG.

   Richten Sie GnuPG so ein, dass es niemals den SHA1-Hashalgorithmus für digitale Signaturen verwendet, der seit 2019 bekanntlich unsicher ist. Fügen Sie dazu zum Beispiel folgende Zeile in `~/.gnupg/gpg.conf` ein (siehe Abschnitt “GPG Esoteric Options” in *The GNU Privacy Guard Manual*):

   ```
   digest-algo sha512
   ```

3. Betreuer entscheiden letztendlich darüber, ob Ihnen Commit-Zugriff gegeben wird, folgen dabei aber normalerweise der Empfehlung Ihrer Fürsprecher.

4. Wenn und sobald Ihnen Zugriff gewährt wurde, senden Sie bitte eine Nachricht an `guix-devel@gnu.org`, um dies bekanntzugeben, die Sie erneut mit dem OpenPGP-Schlüssel signiert haben, mit dem Sie Commits signieren (tun Sie das, bevor Sie Ihren ersten Commit pushen). Auf diese Weise kann jeder Ihren Beitritt mitbekommen und nachprüfen, dass dieser OpenPGP-Schlüssel wirklich Ihnen gehört.

Wichtig: Bevor Sie zum ersten Mal pushen dürfen, müssen die Betreuer:

1. Ihren OpenPGP-Schlüssel zum `keyring`-Branch hinzugefügt haben,
2. Ihren OpenPGP-Fingerabdruck in die Datei `.guix-authorizations` derjenigen Branches eingetragen haben, auf die Sie commiten möchten.

5. Wenn Sie den Rest dieses Abschnitts jetzt auch noch lesen, steht Ihrer Karriere nichts mehr im Weg!

   Anmerkung: Betreuer geben gerne anderen Leuten Commit-Zugriff, die schon einige Zeit dabei waren und ihre Eignung unter Beweis gestellt haben. Seien Sie nicht schüchtern und unterschätzen Sie Ihre Arbeit nicht!

   Sie sollten sich bewusst sein, dass unser Projekt auf ein besser automatisiertes System hinarbeitet, um Patches zu überprüfen und zu mergen. Als Folge davon werden wir vielleicht weniger Leuten Commit-Zugriff auf das Haupt-Repository geben. Bleiben Sie auf dem Laufenden!

Alle Commits, die auf das zentrale Repository auf Savannah gepusht werden, müssen mit einem OpenPGP-Schlüssel signiert worden sein, und diesen öffentlichen Schlüssel sollten Sie auf Ihr Benutzerkonto auf Savannah und auf öffentliche Schlüsselserver wie `keys.openpgp.org` hochladen. Um Git so zu konfigurieren, dass es Commits automatisch signiert, führen Sie diese Befehle aus:

```
git config commit.gpgsign true

# Ersetzen Sie den Fingerabdruck durch Ihren öffentlichen PGP-Schlüssel.
git config user.signingkey CABBA6EA1DC0FF33
```

Um zu prüfen, ob Commits mit dem richtigen Schlüssel signiert wurden, benutzen Sie:

```
make authenticate
```

Sie können als Vorsichtsmaßnahme, um *nicht* versehentlich unsignierte oder mit dem falschen Schlüssel signierte Commits auf Savannah zu pushen, den Pre-Push-Git-Hook benutzen, der sich unter `etc/git/pre-push` befindet:

```
cp etc/git/pre-push .git/hooks/pre-push
```

Dadurch wird außerdem `make check-channel-news` aufgerufen, um zu gewährleisten, dass die Datei `news.scm` gültig ist.

22.8.2 Commit-Richtlinie

Wenn Sie Commit-Zugriff erhalten, passen Sie bitte auf, dass Sie der folgenden Richtlinie folgen (Diskussionen über die Richtlinie können wir auf `guix-devel@gnu.org` führen).

Nichttriviale Patches sollten immer zuerst an `guix-patches@gnu.org` geschickt werden (zu den trivialen Patches gehört zum Beispiel das Beheben von Schreibfehlern usw.). Was an diese Mailing-Liste geschickt wird, steht danach in der Patch-Datenbank (siehe Abschnitt 22.7 [Überblick über gemeldete Fehler und Patches], Seite 731).

Bei Patches, die nur ein einziges neues Paket hinzufügen, das auch noch einfach ist, ist es in Ordnung, sie zu commiten, wenn Sie von von ihnen überzeugt sind (das bedeutet, Sie sollten es in einer chroot-Umgebung erstellt haben und Urheberrecht und Lizenzen mit angemessener Gründlichkeit geprüft haben). Für Paketaktualisierungen gilt dasselbe, außer die Aktualisierung hat viele Neuerstellungen zur Folge (wenn Sie zum Beispiel GnuTLS

oder GLib aktualisieren). Wir haben eine Mailing-Liste für Commit-Benachrichtigungen (`guix-commits@gnu.org`), damit andere sie bemerken. Bevor Sie Ihre Änderungen pushen, führen Sie `git pull --rebase` aus.

Wenn Sie einen Commit für jemand anderen pushen, fügen Sie bitte eine `Signed-off-by`-Zeile am Ende der Commit-Log-Nachricht hinzu – z.B. mit `git am --signoff`. Dadurch lässt es sich leichter überblicken, wer was getan hat.

Wenn Sie Kanalneuigkeiten hinzufügen (siehe Kapitel 7 [Kanäle], Seite 77), dann sollten Sie prüfen, dass diese wohlgeformt sind, indem Sie den folgenden Befehl direkt vor dem Pushen ausführen:

```
make check-channel-news
```

Alles andere schicken Sie bitte an `guix-patches@gnu.org` und warten eine Weile, ohne etwas zu commiten, damit andere Zeit haben, sich die Änderungen anzuschauen (siehe Abschnitt 22.6 [Einreichen von Patches], Seite 724). Wenn Sie nach zwei Wochen keine Antwort erhalten haben und von den Änderungen überzeugt sind, ist es in Ordnung, sie zu commiten.

Die letzten Anweisungen werden wir vielleicht noch ändern, damit man unstrittige Änderungen direkt commiten kann, wenn man mit von Änderungen betroffenen Teilen vertraut ist.

22.8.3 Mit Fehlern umgehen

Durch Begutachten der von anderen eingereichten Patches („Peer-Review“, siehe Abschnitt 22.6 [Einreichen von Patches], Seite 724) und durch Werkzeuge wie `guix lint` (siehe Abschnitt 10.8 [Aufruf von guix lint], Seite 223) und den Testkatalog (siehe Abschnitt 2.3 [Den Testkatalog laufen lassen], Seite 10) sollten Sie Fehler finden können, ehe sie gepusht wurden. Trotz allem kann es gelegentlich vorkommen, dass nicht bemerkt wird, wenn nach einem Commit etwas in Guix nicht mehr funktioniert. Wenn das passiert, haben zwei Dinge Vorrang: Schadensbegrenzung und Verstehen, was passiert ist, damit es nicht wieder zu vergleichbaren Fehlern kommt. Die Verantwortung dafür tragen in erster Linie die Beteiligten, aber wie bei allem anderen handelt es sich um eine Aufgabe der Gruppe.

Manche Fehler können sofort alle Nutzer betreffen, etwa wenn `guix pull` dadurch nicht mehr funktioniert oder Kernfunktionen von Guix ausfallen, wenn wichtige Pakete nicht mehr funktionieren (zur Erstellungs- oder zur Laufzeit) oder wenn bekannte Sicherheitslücken eingeführt werden.

Die am Verfassen, Begutachten und Pushen von Commits Beteiligten sollten zu den ersten gehören, die für zeitnahe Schadensbegrenzung sorgen, indem sie mit einem nachfolgenden Commit („Follow-up“) den Fehler falls möglich beseitigen oder den vorherigen Commit rückgängig machen („Revert“), damit Zeit ist, das Problem auf ordentliche Weise zu beheben. Auch sollten sie das Problem mit anderen Entwicklern bereden.

Wenn diese Personen nicht verfügbar sind, um den Fehler zeitnah zu beheben, steht es anderen Committern zu, deren Commit(s) zu reverten und im Commit-Log und auf der Mailingliste zu erklären, was das Problem war. Ziel ist, dem oder den ursprünglichen Committer(n), Begutachter(n) und Autoren mit mehr Zeit Gelegenheit zu geben, einen Vorschlag zum weiteren Vorgehen zu machen.

Wenn das Problem erledigt wurde, müssen die Beteiligten ein Verständnis für die Situation sicherstellen. Während Sie sich um Verständnis bemühen, legen Sie den Fokus auf das Sammeln von Informationen und vermeiden Sie Schuldzuweisungen. Lassen Sie die Beteiligten beschreiben, was passiert ist, aber bitten Sie sie nicht, die Situation zu erklären, weil ihnen das implizit Schuld zuspricht, was nicht hilfreich ist. Verantwortung ergibt sich aus Einigkeit über das Problem, dass man daraus lernt und die Prozesse verbessert, damit es nicht wieder vorkommt.

22.8.4 Entzug des Commit-Zugriffs

Damit es weniger wahrscheinlich ist, dass Fehler passieren, werden wir das Savannah-Konto von Committern nach 12 Monaten der Inaktivität aus dem Guix-Projekt bei Savannah löschen und ihren Schlüssel aus `.guix-authorizations` entfernen. Solche Committer können den Betreuern eine E-Mail schicken, um ohne einen Durchlauf des Fürsprecherprozesses wieder Commit-Zugriff zu bekommen.

Betreuer[4] dürfen als letzten Ausweg auch jemandem die Commit-Berechtigung entziehen, wenn die Zusammenarbeit mit der übrigen Gemeinde zu viel Reibung erzeugt hat – selbst wenn sich alles im Rahmen der Verhaltensregeln abgespielt hat (siehe Kapitel 22 [Mitwirken], Seite 708). Davon machen Betreuer nur Gebrauch nach öffentlichen oder privaten Diskussionen mit der betroffenen Person und nach eindeutiger Ankündigung. Beispiele für Verhalten, das die Zusammenarbeit behindert und zu so einer Entscheidung führen könnte, sind:

- wiederholte Verletzung der oben beschriebenen Commit-Richtlinie,
- wiederholtes Ignorieren von Kritik anderer Entwickler,
- verletztes Vertrauen durch mehrere schwere Vorkommnisse in Folge.

Wenn Betreuer diesen Entschluss treffen, benachrichtigen sie die Entwickler auf `guix-devel@gnu.org`; Anfragen können an `guix-maintainers@gnu.org` gesendet werden. Abhängig von der Situation wird ein Mitwirken der Betroffenen trotzdem weiterhin gerne gesehen.

22.8.5 Aushelfen

Eine Sache noch: Das Projekt entwickelt sich nicht nur deswegen schnell, weil Committer ihre eigenen tollen Änderungen pushen, sondern auch, weil sie sich Zeit nehmen, die Änderungen anderer Leute in „Reviews“ zu *überprüfen* und zu pushen. Wir begrüßen es, wenn Sie als Committer Ihre Expertise und Commit-Rechte dafür einsetzen, auch anderen Mitwirkenden zu helfen!

22.9 Das Guix-Paket aktualisieren

Manchmal möchte man die für das Paket `guix` (wie es in `(gnu packages package-management)` definiert ist) verwendete Version auf eine neuere Version aktualisieren, zum Beispiel um neue Funktionalitäten des Daemons dem Diensttyp `guix-service-type` zugänglich zu machen. Um diese Arbeit zu erleichtern, kann folgender Befehl benutzt werden:

```
make update-guix-package
```

[4] Siehe `https://guix.gnu.org/en/about` für eine Liste der Betreuer. Sie können ihnen eine private E-Mail schicken an `guix-maintainers@gnu.org`.

Das make-Ziel `update-guix-package` wird den neuesten bekannten *Commit*, also den, der `HEAD` in Ihrem Guix-Checkout entspricht, verwenden, den Hash des zugehörigen Quellbaums berechnen und die Einträge für `commit`, `revision` und den Hash in der Paketdefinition von `guix` anpassen.

Um zu prüfen, dass das aktualisierte `guix`-Paket die richtigen Hashes benutzt und erfolgreich erstellt werden kann, können Sie den folgenden Befehl aus dem Verzeichnis Ihres Guix-Checkouts heraus ausführen:

```
./pre-inst-env guix build guix
```

Als Schutz vor einer versehentlichen Aktualisierung des `guix`-Pakets auf einen Commit, den andere Leute gar nicht haben, wird dabei geprüft, ob der benutzte Commit bereits im bei Savannah angebotenen Guix-Git-Repository vorliegt.

Diese Prüfung können Sie *auf eigene Gefahr hin* abschalten, indem Sie die Umgebungsvariable `GUIX_ALLOW_ME_TO_USE_PRIVATE_COMMIT` setzen. Wenn diese Variable gesetzt ist, wird der aktualisierte Paketquellcode außerdem in den Store eingefügt. Dies stellt einen Teil des Prozesses zur Veröffentlichung neuer Versionen von Guix dar.

22.10 Dokumentation schreiben

Die Dokumentation für Guix wird mit dem Texinfo-System bereitgestellt. Wenn Sie damit noch nicht vertraut sind, nehmen wir auch Beiträge zur Dokumentation in den meisten anderen Formaten an. Reine Textdateien, Markdown, Org und so weiter sind alle in Ordnung.

Schicken Sie Beiträge zur Dokumentation an `guix-patches@gnu.org`. Der Betreff sollte mit '`[DOCUMENTATION]`' beginnen.

Wenn es um mehr als eine kleine Erweiterung der Dokumentation geht, ist es uns allerdings lieber, wenn Sie einen ordentlichen Patch und keine solche E-Mail schicken. Siehe Abschnitt 22.6 [Einreichen von Patches], Seite 724, für mehr Informationen, wie Sie Patches abschicken.

Änderungen an der Dokumentation können Sie vornehmen, indem Sie `doc/guix.texi` und `doc/contributing.texi` bearbeiten (letztere enthält diesen Mitwirkungsteil der Dokumentation) oder indem Sie `doc/guix-cookbook.texi` bearbeiten, worin Sie das Kochbuch finden. Wenn Sie schon einmal die Dateien im Guix-Repository kompiliert haben, werden dort auch viele weitere `.texi`-Dateien zu finden sein; es handelt sich um übersetzte Fassungen dieser Dokumente. Verändern Sie diese *nicht*, denn die Übersetzung organisieren wir über Weblate (`https://translate.fedoraproject.org/projects/guix`). Siehe Abschnitt 22.11 [Guix übersetzen.], Seite 738, für weitere Informationen.

Um die Dokumentation in lesefreundlichere Formate zu übertragen, ist der erste Schritt, `./configure` in Ihrem Guix-Quellbaum laufen zu lassen, wenn es noch nicht geschehen ist (siehe Abschnitt 22.2 [Guix vor der Installation ausführen], Seite 710). Danach geht es mit diesen Befehlen weiter:

- '`make doc/guix.info`', um das Handbuch im Info-Format zu kompilieren. Sie können es mit `info doc/guix.info` überprüfen.
- '`make doc/guix.html`', um die HTML-Version zu kompilieren. Lassen Sie Ihren Browser die entsprechende Datei im Verzeichnis `doc/guix.html` anzeigen.
- '`make doc/guix-cookbook.info`', so kompilieren Sie das Kochbuch im Info-Format.
- '`make doc/guix-cookbook.html`', das kompiliert die HTML-Version des Kochbuchs.

22.11 Guix übersetzen.

Softwareentwicklung und Paketierung sind nicht die einzigen Möglichkeiten, um bedeutsam zu Guix beizutragen. Übersetzungen in andere Sprachen sind auch ein wertvoller Beitrag. In diesem Abschnitt ist der Übersetzungsprozess beschrieben, mit Hinweisen dazu wie du mitmachen kannst, was übersetzt werden kann, welche Fehler du vermeiden solltest und wie wir dich unterstützen können.

Guix ist ein großes Projekt und hat mehrere Komponenten, die übersetzt werden können. Wir koordinieren die Arbeit an der Übersetzung auf einer Weblate-Instanz (`https://translate.fedoraproject.org/projects/guix/`), die von unseren Freunden bei Fedora zur Verfügung gestellt wird. Sie brauchen dort ein Konto, um Übersetzungen einzureichen.

Manche mit Guix auslieferbaren Pakete verfügen auch über Übersetzungen. An deren Übersetzungsplattform sind wir nicht beteiligt. Wenn Sie ein in Guix angebotenes Paket übersetzen möchten, sollten Sie mit dessen Entwicklern in Kontakt treten oder auf deren Webauftritt nach Informationen suchen. Sie können die Homepage zum Beispiel des `hello`-Pakets finden, indem Sie `guix show hello` ausführen. Auf der Zeile „homepage“ sehen Sie, dass `https://www.gnu.org/software/hello/` die Homepage ist.

Viele GNU-Pakete und Nicht-GNU-Pakete können beim Translation Project (`https://translationproject.org`) übersetzt werden. Manche Projekte, die aus mehreren Komponenten bestehen, haben ihre eigene Plattform. Zum Beispiel hat GNOME die Übersetzungsplattform Damned Lies (`https://l10n.gnome.org/`).

Guix hat fünf Komponenten, die auf Weblate übersetzt werden können.

- `guix` umfasst alle Zeichenketten der Guix-Software (also das geführte Installationsprogramm, die Paketverwaltung etc.) außer den Paketen.
- `packages` enthält die Zusammenfassungen (einzeilige Kurzbeschreibung) und die ausführlicheren Beschreibungen der Pakete in Guix.
- `website` enthält den offiziellen Webauftritt von Guix außer Blogeinträgen und Multimedia.
- `documentation-manual` entspricht diesem Handbuch.
- `documentation-cookbook` ist die Komponente für das Kochbuch.

Wie Sie allgemein vorgehen

Sobald Sie ein Konto haben, sollten Sie eine Komponente des Guix-Projekts (`https://translate.fedoraproject.org/projects/guix/`) und eine Sprache auswählen können. Wenn Ihre Sprache in der Liste fehlt, gehen Sie ans Ende und klicken Sie auf den „Neue Übersetzung starten“-Knopf. Nachdem Sie die Sprache, in die Sie übersetzen möchten, ausgewählt haben, wird eine neue Übersetzung angelegt.

Wie zahlreiche andere Freie-Software-Pakete benutzt Guix GNU Gettext (`https://www.gnu.org/software/gettext`) für seine Übersetzungen. Damit werden übersetzbare Zeichenketten aus dem Quellcode in sogenannte PO-Dateien extrahiert.

Obwohl es sich bei PO-Dateien um Textdateien handelt, sollten Änderungen nicht mit einem Texteditor vorgenommen werden, sondern mit Software eigens zum Bearbeiten von PO-Dateien. In Weblate sind PO-Bearbeitungsfunktionen integriert. Alternativ haben Übersetzer die Wahl zwischen vielen Freie-Software-Werkzeugen zum Eintragen von Übersetzungen; ein Beispiel ist Poedit (`https://poedit.net/`). Nachdem Sie sich

angemeldet haben, laden (`https://docs.weblate.org/en/latest/user/files.html`) Sie die geänderte Datei in Weblate hoch. Es gibt auch einen speziellen PO-Bearbeitungsmodus (`https://www.emacswiki.org/emacs/PoMode`) für Nutzer von GNU Emacs. Mit der Zeit finden Übersetzer heraus, welche Software sie bevorzugen und welche die Funktionen bietet, die sie brauchen.

In Weblate finden Sie an vielen Stellen Verweise auf den Editor, um Teilmengen der Zeichenketten oder alle davon zu übersetzen. Sehen Sie sich um und werfen Sie einen Blick auf Weblates Dokumentation (`https://docs.weblate.org/en/latest/`), um sich mit der Plattform vertraut zu machen.

Übersetzungskomponenten

In diesem Abschnitt erklären wir den Übersetzungsprozess genau und geben Details, was Sie tun sollten und was nicht. Im Zweifelsfall kontaktieren Sie uns bitte; wir helfen gerne!

guix

Guix ist in der Programmiersprache Guile geschrieben und manche Zeichenketten enthalten besondere Formatzeichen, die von Guile interpretiert werden. Weblate sollte diese besonderen Formatzeichen hervorheben. Sie beginnen mit `~` gefolgt von einem oder mehreren Zeichen.

Beim Anzeigen der Zeichenkette ersetzt Guile die speziellen Formatierungsmuster durch echte Werte. Zum Beispiel würde in die Zeichenkette ‘`ambiguous package specification `~a'`’ die Paketspezifikation anstelle von `~a` eingesetzt. Um die Zeichenkette richtig zu übersetzen, müssen Sie den Formatierungscode in Ihrer Übersetzung erhalten, aber Sie können ihn an die in Ihrer Sprache passenden Stelle in der Übersetzung verlegen. Die französische Übersetzung ist zum Beispiel ‘`spécification du paquet « ~a » ambiguë`’, weil das Adjektiv dort ans Ende vom Satz gehört.

Wenn mehrere Formatsymbole vorkommen, sollten Sie darauf achten, die Reihenfolge beizubehalten. Guile weiß nicht, in welcher Reihenfolge die eingesetzten Symbole auftauchen sollen, also setzt es sie in derselben Abfolge wie im englischen Satz ein.

Es geht zum Beispiel *nicht*, den Satz ‘`package '~a' has been superseded by '~a'`’ wie ‘`'~a' superseeds package '~a'`’ zu übersetzen, weil die Bedeutung umgekehrt wäre. Wenn *foo* durch *bar* abgelöst wird, würde die Übersetzung behaupten, ‘`„foo“ löst Paket „bar“ ab`’. Um dieses Problem zu umgehen, ist es möglich, fortgeschrittene Formatierung einzusetzen, mit der ein bestimmtes Datum ausgewählt wird statt der englischen Reihenfolge zu folgen. Siehe Abschnitt “Formatted Output” in *Referenzhandbuch zu GNU Guile* für weitere Informationen zu formatierter Ausgabe in Guile.

Pakete

Manchmal trifft man in einer Paketbeschreibung auf in Texinfo-Markup ausgezeichnete Wörter (siehe Abschnitt 22.4.4 [Zusammenfassungen und Beschreibungen], Seite 717). Texinfo-Auszeichnungen sehen aus wie ‘`@code{rm -rf}`’, ‘`@emph{important}`’ usw. Wenn Sie übersetzen, belassen Sie Auszeichnungen bitte, wie sie sind.

Die „@“ nachfolgenden Zeichen bilden den Namen der Auszeichnung und der Text zwischen „{“ und „}“ ist deren Inhalt. Im Allgemeinen sollten Sie *nicht*

den Inhalt von Auszeichnungen wie `@code` übersetzen, weil Code wortwörtlich so aussehen muss und sich *nicht* mit der Sprache ändert. Dagegen können Sie den Inhalt von zur Formatierung dienenden Auszeichnungen wie `@emph`, `@i`, `@itemize`, `@item` durchaus ändern. Übersetzen Sie aber *nicht* den Namen der Auszeichnung, sonst wird sie nicht erkannt. Übersetzen Sie *nicht* das auf `@end` folgende Wort, denn es ist der Name der Auszeichnung, die an dieser Position geschlossen wird (z.B. `@itemize ... @end itemize`).

documentation-manual und documentation-cookbook

Der erste Schritt für eine erfolgreiche Übersetzung des Handbuchs ist, folgende Zeichenketten *als Erstes* zu finden und zu übersetzen:

- `version.texi`: Übersetzen Sie diese Zeichenkette mit `version-xx.texi` wobei `xx` Ihr Sprachcode ist (wie er in der URL auf Weblate steht).
- `contributing.texi`: Übersetzen Sie diese Zeichenkette mit `contributing.xx.texi`, wobei `xx` derselbe Sprachcode ist.
- `Top`: Übersetzen Sie diese Zeichenkette *nicht*; sie ist wichtig für Texinfo. Würden Sie sie übersetzen, würde das Dokument leer (weil ein Top-Knoten fehlt). Bitte suchen Sie die Zeichenkette und tragen Sie `Top` als Übersetzung ein.

Wenn diese Zeichenketten als erste ausgefüllt wurden, ist gewährleistet, dass wir die Übersetzung ins Guix-Repository übernehmen können, ohne dass der make-Vorgang oder die Maschinerie hinter `guix pull` zusammenbricht.

Das Handbuch und das Kochbuch verwenden beide Texinfo. Wie bei `packages` belassen Sie Texinfo-Auszeichnungen bitte so, wie sie sind. Im Handbuch gibt es mehr mögliche Auszeichnungstypen als in den Paketbeschreibungen. Im Allgemeinen sollten Sie den Inhalt von `@code`, `@file`, `@var`, `@value` usw. *nicht* übersetzen. Sie sollten aber den Inhalt von Formatierungsauszeichnungen wie `@emph`, `@i` usw. übersetzen.

Das Handbuch enthält Abschnitte, auf die man anhand ihres Namens mit `@ref`, `@xref` und `@pxref` verweisen kann. Wir haben einen Mechanismus eingebaut, damit Sie deren Inhalt *nicht* übersetzen müssen. Lassen Sie einfach den englischen Titel stehen, dann werden wir ihn automatisch durch Ihre Übersetzung des Titels ersetzen. So wird garantiert, dass Texinfo den Knoten auf jeden Fall finden kann. Wenn Sie später die Übersetzung ändern, ändern sich die Verweise darauf automatisch mit und Sie müssen sie nicht alle selbst anpassen.

Nur wenn Sie Verweise vom Kochbuch auf das Handbuch übersetzen, dann müssen Sie den Namen des Handbuchs und den Namen des Abschnitts ersetzen. Zum Beispiel übersetzen Sie `@pxref{Defining Packages,,, guix, GNU Guix Reference Manual}`, indem Sie `Defining Packages` durch den Titel dieses Abschnittes im übersetzten Handbuch ersetzen, aber *nur dann*, wenn der Titel bereits übersetzt wurde. Wenn er nicht übersetzt wurde, dann übersetzen Sie ihn auch hier *nicht*, sonst funktioniert der Verweis nicht. Schreiben Sie `guix.xx` statt `guix`, wobei `xx` Ihr Sprachcode ist. Bei `GNU Guix Reference Manual` handelt es sich um den Text, mit dem der Verweis angezeigt wird. Übersetzen Sie ihn wie immer Sie mögen.

website

Die Seiten des Webauftritts sind in SXML geschrieben, einer Version von HTML, der Grundlage des Webs, aber mit S-Ausdrücken. Wir haben einen Prozess, um übersetzbare Zeichenketten aus dem Quellcode zu extrahieren und komplexe S-Ausdrücke als vertrautere XML-Auszeichnungen darzustellen. Dabei werden die Auszeichnungen nummeriert. Übersetzer können deren Reihenfolge nach Belieben ändern, wie im folgenden Beispiel zu sehen ist.

```
#. 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
```

Allerdings müssen Sie dieselben Auszeichnungen verwenden. Sie können keine weglassen.

Wenn Ihnen ein Fehler unterläuft, kann die Komponente in Ihrer Sprache womöglich *nicht* erstellt werden oder guix pull kann sogar fehlschlagen. Um dies zu verhindern, haben wir einen Prozess, um den Inhalt der Dateien vor einem Push auf das Repository zu überprüfen. In so einem Fall werden wir die Übersetzung für Ihre Sprache *nicht* aktualisieren können und werden Sie deshalb benachrichtigen (über Weblate und/oder eine E-Mail), damit Sie die Möglichkeit bekommen, das Problem zu beheben.

Abseits von Weblate

Momentan können manche Teile von Guix noch nicht mit Weblate übersetzt werden. Wir freuen uns über Hilfe!

- Bei `guix pull` angezeigte Neuigkeiten können in `news.scm` übersetzt werden, jedoch *nicht* über Weblate. Wenn Sie eine Übersetzung bereitstellen wollen, können Sie uns einen Patch wie oben beschrieben schicken oder uns einfach die Übersetzung zusammen mit Ihrer Sprache und dem Namen des Neuigkeiteneintrags, den Sie übersetzt haben, zukommen lassen. Siehe Abschnitt 7.11 [Kanalneuigkeiten verfassen], Seite 84, für weitere Informationen zu Neuigkeiten über Kanäle.
- Blogeinträge zu Guix können derzeit *nicht* übersetzt werden.
- Das Installations-Skript (für Fremddistributionen) gibt es nur auf Englisch.
- Manche der Bibliotheken, die Guix benutzt, können *nicht* übersetzt werden oder sie werden außerhalb von Guix übersetzt. Guile selbst ist *nicht* internationalisiert.
- Andere Handbücher, auf die vom Handbuch oder Kochbuch aus verwiesen wird, sind oft *nicht* übersetzt.

Bedingungen für die Aufnahme

Es müssen keine Bedingungen erfüllt werden, damit neue Übersetzungen der Komponenten `guix` und `guix-packages` übernommen werden, außer dass sie mindestens eine übersetzte Zeichenkette enthalten. Neue Sprachen werden so bald wie möglich aufgenommen. Die Dateien können wieder entfernt werden, wenn sie keine Übersetzungen mehr enthalten, weil die Zeichenketten, die übersetzt wurden, nicht mehr mit den in Guix verwendeten übereinstimmen.

Aufgrund dessen, dass sich der Webauftritt an neue Benutzer richtet, möchten wir, dass dessen Übersetzung so vollständig wie möglich ist, bevor wir sie ins Menü zur Sprachauswahl aufnehmen. Eine neue Sprache muss dazu zumindest zu 80% vollständig sein. Einmal aufgenommen kann eine Sprache dann wieder entfernt werden, wenn sie eine Zeit lang nicht mehr übereinstimmt und zu weniger als 60% übersetzt ist.

Handbuch und Kochbuch sind automatisch Teil des Standard-Ziels beim Kompilieren. Jedes Mal, wenn wir die Übersetzungen in Guix mit Weblate abgleichen, müssen die Entwickler alle übersetzten Hand- und Kochbücher neu kompilieren. Das lohnt sich nicht, wenn so wenig übersetzt wurde, dass wir größtenteils wieder das englische Hand- oder Kochbuch vor uns haben. Aus diesem Grund nehmen wir eine neue Sprache hier nur auf, wenn mindestens 10% der jeweiligen Komponente übersetzt sind. Eine bereits aufgenommene Sprache, die eine Zeit lang nicht übereinstimmt und wo weniger als 5% übersetzt sind, kann entfernt werden.

Übersetzungsinfrastruktur

Hinter Weblate steht ein Git-Repository, aus dem die Weblate-Instanz neue zu übersetzende Zeichenketten bezieht und in das sie die neuen und aktualisierten Übersetzungen pusht. Normalerweise würde es ausreichen, Weblate Commit-Zugriff auf unsere Repositorys zu geben, aber wir haben uns aus zwei Gründen dagegen entschieden. Erstens müssten wir sonst Weblate Commit-Zugriff geben und seinen Signierschlüssel autorisieren, aber wir haben in Weblate nicht auf gleiche Art Vertrauen wie in die Guix-Entwickler, besonders weil wir die Instanz nicht selbst verwalten. Zweitens könnten durch ein Missgeschick der Übersetzer dann das Erzeugen des Webauftritts und/oder guix-pull-Aufrufe für all unsere Nutzer aufhören zu funktionieren, ganz gleich was deren Spracheinstellungen sind.

Daher haben wir ein separates Repository für die Übersetzungen, das wir mit unseren guix- und guix-artwork-Repositorys abgleichen, nachdem wir geprüft haben, dass es durch die Übersetzung zu keinem Fehler kommt.

Entwickler können die neuesten PO-Dateien von Weblate in ihr Guix-Repository laden, indem sie den Befehl `make download-po` ausführen. Er wird die neuesten Dateien von Weblate automatisch herunterladen, in eine kanonische Form bringen und überprüfen, dass sie keine Fehler verursachen. Das Handbuch muss neu erstellt werden, um sicherzugehen, dass keine neuen Probleme zum Absturz von Texinfo führen.

Bevor sie neue Übersetzungsdateien pushen, sollten Entwickler achtgeben, dass jene Dateien eingetragen sind, damit die Make-Maschinerie die Übersetzungen auch tatsächlich bereitstellt. Der Prozess dafür unterscheidet sich je nach Komponente.

- Neue PO-Dateien für die Komponenten `guix` und `packages` müssen registriert werden, indem man die neue Sprache in `po/guix/LINGUAS` oder `po/packages/LINGUAS` hinzufügt.
- Neue PO-Dateien für die Komponente `documentation-manual` müssen registriert werden, indem man deren Dateinamen zu `DOC_PO_FILES` in `po/doc/local.mk` hinzufügt, die erzeugte Handbuchdatei `%D%/guix.xx.texi` zu `info_TEXINFOS` in `doc/local.mk` hinzufügt und sowohl die erzeugte `%D%/guix.xx.texi` als auch `%D%/contributing.xx.texi` zu `TRANSLATED_INFO` auch wieder in `doc/local.mk` hinzufügt.

- Neue PO-Dateien für die Komponente `documentation-cookbook` müssen registriert werden, indem man deren Dateinamen zu `DOC_COOKBOOK_PO_FILES` in `po/doc/local.mk` hinzufügt, die erzeugte Handbuchdatei `%D%/guix-cookbook.xx.texi` zu `info_TEXINFOS` in `doc/local.mk` hinzufügt und die erzeugte `%D%/guix-cookbook.xx.texi` zu `TRANSLATED_INFO` auch wieder in `doc/local.mk` hinzufügt.
- Neue PO-Dateien für die Komponente `website` müssen zum Repository `guix-artwork` ins dortige Verzeichnis `website/po/` hinzugefügt werden. `website/po/LINGUAS` und `website/po/ietf-tags.scm` müssen entsprechend aktualisiert werden (siehe `website/i18n-howto.txt` für Details zum Prozess).

23 Danksagungen

Guix baut auf dem Nix-Paketverwaltungsprogramm (`https://nixos.org/nix/`) auf, das von Eelco Dolstra entworfen und entwickelt wurde, mit Beiträgen von anderen Leuten (siehe die Datei `nix/AUTHORS` in Guix). Nix hat für die funktionale Paketverwaltung die Pionierarbeit geleistet und noch nie dagewesene Funktionalitäten vorangetrieben wie transaktionsbasierte Paketaktualisierungen und die Rücksetzbarkeit selbiger, eigene Paketprofile für jeden Nutzer und referenziell transparente Erstellungsprozesse. Ohne diese Arbeit gäbe es Guix nicht.

Die Nix-basierten Software-Distributionen Nixpkgs und NixOS waren auch eine Inspiration für Guix.

GNU Guix ist selbst das Produkt kollektiver Arbeit mit Beiträgen durch eine Vielzahl von Leuten. Siehe die Datei `AUTHORS` in Guix für mehr Informationen, wer diese wunderbaren Menschen sind. In der Datei `THANKS` finden Sie eine Liste der Leute, die uns geholfen haben, indem Sie Fehler gemeldet, sich um unsere Infrastruktur gekümmert, künstlerische Arbeit und schön gestaltete Themen beigesteuert, Vorschläge gemacht und noch vieles mehr getan haben – vielen Dank!

Anhang A GNU-Lizenz für freie Dokumentation

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.

Konzeptverzeichnis

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Programmierverzeichnis

#

%

,

(

,

>

‘

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z