

# **Manuel de référence de GNU Guix**

---

Utiliser le gestionnaire de paquet fonctionnel GNU Guix

**Les développeurs de GNU Guix**

---

Édition 1.4.0  
8 June 2023

Copyright © 2012-2022 Ludovic Courtès  
Copyright © 2013, 2014, 2016 Andreas Enge  
Copyright © 2013 Nikita Karetnikov  
Copyright © 2014, 2015, 2016 Alex Kost  
Copyright © 2015, 2016 Mathieu Lirzin  
Copyright © 2014 Pierre-Antoine Rault  
Copyright © 2015 Taylan Ulrich Bayırlı/Kammer  
Copyright © 2015, 2016, 2017, 2019, 2020, 2021 Leo Famulari  
Copyright © 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022 Ricardo Wurmus  
Copyright © 2016 Ben Woodcroft  
Copyright © 2016, 2017, 2018, 2021 Chris Marusich  
Copyright © 2016, 2017, 2018, 2019, 2020, 2021, 2022 Efraim Flashner  
Copyright © 2016 John Darrington  
Copyright © 2016, 2017 Nikita Gillmann  
Copyright © 2016, 2017, 2018, 2019, 2020 Jan Nieuwenhuizen  
Copyright © 2016, 2017, 2018, 2019, 2020, 2021 Julien Lepiller  
Copyright © 2016 Alex ter Weele  
Copyright © 2016, 2017, 2018, 2019, 2020, 2021 Christopher Baines  
Copyright © 2017, 2018, 2019 Clément Lassieur  
Copyright © 2017, 2018, 2020, 2021, 2022 Mathieu Othacehe  
Copyright © 2017 Federico Beffa  
Copyright © 2017, 2018 Carlo Zancanaro  
Copyright © 2017 Thomas Danckaert  
Copyright © 2017 humanitiesNerd  
Copyright © 2017, 2021 Christine Lemmer-Webber  
Copyright © 2017, 2018, 2019, 2020, 2021, 2022 Marius Bakke  
Copyright © 2017, 2019, 2020, 2022 Hartmut Goebel  
Copyright © 2017, 2019, 2020, 2021, 2022 Maxim Cournoyer  
Copyright © 2017-2022 Tobias Geerinckx-Rice  
Copyright © 2017 George Clemmer  
Copyright © 2017 Andy Wingo  
Copyright © 2017, 2018, 2019, 2020 Arun Isaac  
Copyright © 2017 nee  
Copyright © 2018 Rutger Helling  
Copyright © 2018, 2021 Oleg Pykhalov  
Copyright © 2018 Mike Gerwitz  
Copyright © 2018 Pierre-Antoine Rouby  
Copyright © 2018, 2019 Gábor Boskovits  
Copyright © 2018, 2019, 2020, 2022 Florian Pelz  
Copyright © 2018 Laura Lazzati  
Copyright © 2018 Alex Vong  
Copyright © 2019 Josh Holland

Copyright © 2019, 2020 Diego Nicola Barbato  
Copyright © 2019 Ivan Petkov  
Copyright © 2019 Jakob L. Kreuze  
Copyright © 2019 Kyle Andrews  
Copyright © 2019 Alex Griffin  
Copyright © 2019, 2020, 2021, 2022 Guillaume Le Vaillant  
Copyright © 2020 Liliana Marie Prikler  
Copyright © 2019, 2020, 2021, 2022 Simon Tournier  
Copyright © 2020 Wiktor Żelazny  
Copyright © 2020 Damien Cassou  
Copyright © 2020 Jakub Kądziołka  
Copyright © 2020 Jack Hill  
Copyright © 2020 Naga Malleswari  
Copyright © 2020, 2021 Brice Waegeneire  
Copyright © 2020 R Veera Kumar  
Copyright © 2020, 2021 Pierre Langlois  
Copyright © 2020 pinoaffe  
Copyright © 2020 André Batista  
Copyright © 2020, 2021 Alexandru-Sergiu Marton  
Copyright © 2020 raingloom  
Copyright © 2020 Daniel Brooks  
Copyright © 2020 John Soo  
Copyright © 2020 Jonathan Brielmaier  
Copyright © 2020 Edgar Vincent  
Copyright © 2021, 2022 Maxime Devos  
Copyright © 2021 B. Wilson  
Copyright © 2021 Xinglu Chen  
Copyright © 2021 Raghav Gururajan  
Copyright © 2021 Domagoj Stolfa  
Copyright © 2021 Hui Lu  
Copyright © 2021 pukkamustard  
Copyright © 2021 Alice Brenon  
Copyright © 2021, 2022 Josselin Poiret  
Copyright © 2021 muradm  
Copyright © 2021, 2022 Andrew Tropin  
Copyright © 2021 Sarah Morgensen  
Copyright © 2022 Remco van 't Veer  
Copyright © 2022 Aleksandr Vityazev  
Copyright © 2022 Philip M<sup>c</sup>Grath  
Copyright © 2022 Karl Hallsby  
Copyright © 2022 Justin Veilleux  
Copyright © 2022 Reily Siegel  
Copyright © 2022 Simon Streit  
Copyright © 2022 (  
Copyright © 2022 John Kehayias

Vous avez la permission de copier, distribuer ou modifier ce document sous les termes de la Licence GNU Free Documentation, version 1.3 ou toute version ultérieure publiée par la Free Software Foundation ; sans section invariante, texte de couverture et sans texte de quatrième de couverture. Une copie de la licence est incluse dans la section intitulée « GNU Free Documentation License ».

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Gérer ses logiciels avec Guix	1
1.2	Distribution GNU	2
<b>2</b>	<b>Installation</b>	<b>5</b>
2.1	Installation binaire	5
2.2	Prérequis	8
2.3	Lancer la suite de tests	9
2.4	Paramétrer le démon	11
2.4.1	Réglages de l’environnement de construction	11
2.4.2	Utiliser le dispositif de déchargement	12
2.4.3	Support de SELinux	16
2.4.3.1	Installer la politique SELinux	16
2.4.3.2	Limitations	16
2.5	Invoquer <code>guix-daemon</code>	17
2.6	Réglages applicatifs	22
2.6.1	Régionalisation	22
2.6.2	Name Service Switch	23
2.6.3	Polices X11	24
2.6.4	Certificats X.509	24
2.6.5	Paquets Emacs	24
2.7	Mettre à niveau Guix	25
<b>3</b>	<b>Installation du système</b>	<b>26</b>
3.1	Limitations	26
3.2	Considérations matérielles	26
3.3	Installation depuis une clef USB ou un DVD	27
	Copie sur une clef USB	27
	Graver sur un DVD	27
	Démarrage	27
3.4	Préparer l’installation	28
3.5	Installation graphique guidée	28
3.6	Installation manuelle	30
3.6.1	Disposition du clavier réseau et partitionnement	30
3.6.1.1	Disposition du clavier	31
3.6.1.2	Réseau	31
3.6.1.3	Partitionnement	32
3.6.2	Effectuer l’installation	34
3.7	Après l’installation du système	35
3.8	Installer Guix sur une machine virtuelle	36
3.9	Construire l’image d’installation	36
3.10	Construire l’image d’installation pour les cartes ARM	36

<b>4</b>	<b>Corriger les problèmes du système</b>	<b>38</b>
4.1	Chrooter dans un système existant	38
<b>5</b>	<b>Pour démarrer</b>	<b>40</b>
<b>6</b>	<b>Gestion de paquets</b>	<b>43</b>
6.1	Fonctionnalités	43
6.2	Invoquer <code>guix package</code>	44
6.3	Substituts	54
6.3.1	Serveur de substituts officiel	54
6.3.2	Autoriser un serveur de substituts	55
6.3.3	Récupérer des substituts d'autres serveurs	56
6.3.4	Authentification des substituts	57
6.3.5	Paramètres de serveur mandataire	58
6.3.6	Échec de substitution	58
6.3.7	De la confiance en des binaires	58
6.4	Des paquets avec plusieurs résultats	59
6.5	Invoquer <code>guix gc</code>	60
6.6	Invoquer <code>guix pull</code>	63
6.7	Invoquer <code>guix time-machine</code>	66
6.8	Inférieurs	68
6.9	Invoquer <code>guix describe</code>	70
6.10	Invoquer <code>guix archive</code>	71
<b>7</b>	<b>Canaux</b>	<b>75</b>
7.1	Spécifier des canaux supplémentaires	75
7.2	Utiliser un canal Guix personnalisé	75
7.3	Répliquer Guix	76
7.4	Authentification des canaux	77
7.5	Canaux avec des substituts	77
7.6	Écrire de nouveaux de canaux	78
7.7	Modules de paquets dans un sous-répertoire	79
7.8	Déclarer des dépendances de canaux	79
7.9	Spécifier les autorisations des canaux	80
7.10	URL primaire	81
7.11	Écrire des nouveautés de canaux	81
<b>8</b>	<b>Développement</b>	<b>83</b>
8.1	Invoquer <code>guix shell</code>	83
8.2	Invoquer <code>guix environment</code>	90
8.3	Invoquer <code>guix pack</code>	96
8.4	La chaîne d'outils GCC	101
8.5	Invoquer <code>guix git authenticate</code>	102

<b>9</b>	<b>Interface de programmation</b>	<b>104</b>
9.1	Modules de paquets	104
9.2	Définition des paquets	105
9.2.1	Référence de <code>package</code>	108
9.2.2	Référence de <code>origin</code>	113
9.3	Définition de variantes de paquets	116
9.4	Écrire un manifeste	120
9.5	Systèmes de construction	124
9.6	Phases de construction	143
9.7	Utilitaires de construction	146
9.7.1	Traitement des noms de fichiers du dépôt	147
9.7.2	Types de fichier	147
9.7.3	Manipulation de fichiers	148
9.7.4	Recherche de fichiers	149
9.7.5	Invocation de programme	150
9.7.6	Phases de construction	151
9.7.7	Enveloppes	152
9.8	Chemins de recherche	154
9.9	Le dépôt	157
9.10	Dérivations	159
9.11	La monade du dépôt	162
9.12	G-Expressions	166
9.13	Invoquer <code>guix repl</code>	176
9.14	Utiliser Guix de manière interactive	177
<b>10</b>	<b>Utilitaires</b>	<b>180</b>
10.1	Invoquer <code>guix build</code>	180
10.1.1	Options de construction communes	180
10.1.2	Options de transformation de paquets	183
10.1.3	Options de construction supplémentaires	188
10.1.4	Débogage des échecs de construction	193
10.2	Invoquer <code>guix edit</code>	194
10.3	Invoquer <code>guix download</code>	195
10.4	Invoquer <code>guix hash</code>	196
10.5	Invoquer <code>guix import</code>	197
10.6	Invoquer <code>guix refresh</code>	205
10.7	Invoquer <code>guix style</code>	210
10.8	Invoquer <code>guix lint</code>	213
10.9	Invoquer <code>guix size</code>	215
10.10	Invoque <code>guix graph</code>	218
10.11	Invoquer <code>guix publish</code>	222
10.12	Invoquer <code>guix challenge</code>	226
10.13	Invoquer <code>guix copy</code>	229
10.14	Invoquer <code>guix container</code>	230
10.15	Invoquer <code>guix weather</code>	231
10.16	Invoquer <code>guix processes</code>	233

<b>11</b>	<b>Architectures externes .....</b>	<b>235</b>
11.1	Compilation croisée .....	235
11.2	Constructions natives .....	236
<b>12</b>	<b>Configuration du système .....</b>	<b>238</b>
12.1	Utiliser le système de configuration .....	238
	Bootloader .....	239
	Paquets visibles sur tout le système .....	240
	Services systèmes .....	240
	Instancier le système .....	245
	L'interface de programmation .....	246
12.2	Référence de <code>operating-system</code> .....	246
12.3	Systèmes de fichiers .....	250
	12.3.1 Système de fichier Btrfs .....	255
12.4	Périphériques mappés .....	256
12.5	Espace d'échange .....	258
12.6	Comptes utilisateurs .....	260
12.7	Disposition du clavier .....	263
12.8	Régionalisation .....	265
	12.8.1 Considérations sur la compatibilité des données linguistiques .....	266
12.9	Services .....	267
	12.9.1 Services de base .....	268
	12.9.2 Exécution de tâches planifiées .....	287
	12.9.3 Rotation des journaux .....	289
	12.9.4 Configuration du réseau .....	292
	12.9.5 Services réseau .....	299
	12.9.6 Mises à jour non surveillées (Unattended Upgrades) .....	322
	12.9.7 Système de fenêtrage X .....	325
	12.9.8 Services d'impression .....	334
	12.9.9 Services de bureaux .....	347
	12.9.10 Services de son .....	364
	12.9.11 Services de bases de données .....	366
	12.9.12 Services de courriels .....	372
	12.9.13 Services de messagerie .....	400
	12.9.14 Services de téléphonie .....	408
	12.9.15 Services de partage de fichiers .....	415
	12.9.16 Services de surveillance .....	427
	12.9.17 Services Kerberos .....	433
	12.9.18 Services LDAP .....	435
	12.9.19 Services web .....	441
	12.9.20 Services de certificats .....	461
	12.9.21 Services DNS .....	464
	12.9.22 Services VNC .....	477
	12.9.23 Services VPN .....	479
	12.9.24 Système de fichiers en réseau .....	485



12.9.25	Services Samba	488
12.9.26	Intégration continue	490
12.9.27	Services de gestion de l'énergie	495
12.9.28	Services audio	503
12.9.29	Services de virtualisation	504
12.9.30	Services de contrôle de version	529
12.9.31	Services de jeu	547
12.9.32	Service PAM de montage	548
12.9.33	Services Guix	549
12.9.34	Services Linux	555
12.9.35	Services Hurd	559
12.9.36	Services divers	559
12.10	Programmes setuid	571
12.11	Certificats X.509	572
12.12	Name Service Switch	573
12.13	Disque de RAM initial	575
12.14	Configuration du chargeur d'amorçage	578
12.15	Invoquer <code>guix system</code>	584
12.16	Invoquer <code>guix deploy</code>	593
12.17	Exécuter Guix sur une machine virtuelle	597
12.17.1	Se connecter par SSH	598
12.17.2	Utiliser <code>virt-viewer</code> avec Spice	598
12.18	Définir des services	599
12.18.1	Composition de services	599
12.18.2	Types service et services	600
12.18.3	Référence de service	602
12.18.4	Services Shepherd	607
12.18.5	Configurations complexes	612
<b>13</b>	<b>Configuration du dossier personnel</b>	<b>619</b>
13.1	Déclarer l'environnement personnel	619
13.2	Configurer le shell	621
13.3	Services du dossier personnel	622
13.3.1	Services personnels essentiels	622
13.3.2	Shells	625
13.3.3	Exécution de tâches planifiées personnelles	628
13.3.4	Services personnels de gestion de l'énergie	629
13.3.5	Gérer les démons personnels	630
13.3.6	Shell sécurisé	631
13.3.7	Services personnels pour ordinateur de bureau	634
13.3.8	Services personnels Guix	635
13.4	Invoquer <code>guix home</code>	636
<b>14</b>	<b>Documentation</b>	<b>641</b>

<b>15</b>	<b>Plateformes</b> .....	<b>642</b>
15.1	Référence de <code>platform</code> .....	642
15.2	Plateformes prises en charge .....	642
<b>16</b>	<b>Créer des images systèmes</b> .....	<b>644</b>
16.1	Référence de <code>image</code> .....	644
16.1.1	Référence de <code>partition</code> .....	645
16.2	Instancier une image .....	646
16.3	référence de <code>image-type</code> .....	649
16.4	Modules des images .....	651
<b>17</b>	<b>Installer les fichiers de débogage</b> .....	<b>652</b>
17.1	Informations de débogage séparées .....	652
17.2	Reconstruire les informations de débogage .....	653
<b>18</b>	<b>Utiliser <code>TeX</code> et <code>LaTeX</code></b> .....	<b>655</b>
<b>19</b>	<b>Mises à jour de sécurité</b> .....	<b>657</b>
<b>20</b>	<b>Bootstrapping</b> .....	<b>659</b>
20.1	Le bootstrap avec les sources binaires réduites .....	659
20.2	Se préparer à utiliser les binaires de bootstrap .....	662
	Construire les outils de construction .....	663
	Construire les binaires de bootstrap .....	665
	Réduire l'ensemble des binaires de bootstrap .....	665
<b>21</b>	<b>Porter vers une nouvelle plateforme</b> .....	<b>667</b>
<b>22</b>	<b>Contribuer</b> .....	<b>668</b>
22.1	Construire depuis Git .....	668
22.2	Lancer Guix avant qu'il ne soit installé .....	670
22.3	La configuration parfaite .....	671
22.4	Consignes d'empaquetage .....	673
22.4.1	Liberté logiciel .....	673
22.4.2	Conventions de nommage .....	674
22.4.3	Numéros de version .....	674
22.4.4	Synopsis et descriptions .....	676
22.4.5	Substituts ou Phases .....	677
22.4.6	Paquets Emacs .....	677
22.4.7	Modules Python .....	678
22.4.7.1	Spécifier les dépendances .....	679
22.4.8	Modules Perl .....	679
22.4.9	Paquets Java .....	680

22.4.10	Paquets Rust.....	680
22.4.11	Paquets elm.....	680
22.4.12	Polices de caractères.....	682
22.5	Style de code.....	682
22.5.1	Paradigme de programmation.....	682
22.5.2	Modules.....	682
22.5.3	Types de données et reconnaissance de motif.....	682
22.5.4	Formatage du code.....	683
22.6	Envoyer des correctifs.....	683
22.6.1	Configurer Git.....	687
22.6.2	Envoyer une série de correctifs.....	687
	Correctifs isolés.....	687
	Notifier les équipes.....	688
	Plusieurs correctifs.....	688
22.6.3	Équipes.....	689
22.7	Suivi des bogues et des correctifs.....	689
22.7.1	L'outil de gestion des défauts.....	689
22.7.2	Interfaces utilisateurs à Debbugs.....	690
22.7.3	Étiquettes personnalisées de Debbugs.....	690
22.8	Accès en commit.....	691
22.8.1	Candidater pour avoir accès en commit.....	691
22.8.2	Politique de commit.....	693
22.8.3	Régler les problèmes.....	693
22.8.4	Révocation des droits en commit.....	694
22.8.5	Aider.....	694
22.9	Mettre à jour Guix.....	695
22.10	Écrire de la documentation.....	695
22.11	Traduire Guix.....	696
<b>23</b>	<b>Remerciements.....</b>	<b>701</b>
	<b>Annexe A La licence GNU Free Documentation ..</b>	<b>702</b>
	<b>Index des concepts.....</b>	<b>710</b>
	<b>Index de programmation.....</b>	<b>719</b>

# 1 Introduction

GNU Guix<sup>1</sup> est un outil de gestion de paquets et une distribution pour le système GNU. Guix facilite pour les utilisateur·rice·s non privilégié·e·s l’installation, la mise à jour et la suppression de paquets, la restauration à un ensemble de paquets précédent, la construction de paquets depuis les sources et plus généralement aide à la création et à la maintenance d’environnements logiciels.

Vous pouvez installer GNU Guix sur un système GNU/Linux existant pour compléter les outils disponibles sans interférence (voir Chapitre 2 [Installation], page 5) ou vous pouvez l’utiliser comme distribution système indépendante, *Guix System*<sup>2</sup>. Voir Section 1.2 [Distribution GNU], page 2.

## 1.1 Gérer ses logiciels avec Guix

Guix fournit une interface de gestion des paquets par la ligne de commande (voir Chapitre 6 [Gestion de paquets], page 43), des outils pour aider au développement logiciel (voir Chapitre 8 [Développement], page 83), des utilitaires en ligne de commande pour des utilisations plus avancées (voir Chapitre 10 [Utilitaires], page 180) ainsi que des interfaces de programmation Scheme (voir Chapitre 9 [Interface de programmation], page 104).

Son *démon de construction* est responsable de la construction des paquets pour les utilisateur·rice·s (voir Section 2.4 [Paramétrer le démon], page 11) et du téléchargement des binaires pré-construits depuis les sources autorisées (voir Section 6.3 [Substituts], page 54).

Guix contient de nombreuses définitions de paquet GNU et non-GNU qui respectent tous les libertés de l’utilisateur ou utilisatrice (<https://www.gnu.org/philosophy/free-sw.fr.html>). Il est *extensible* : chacun·e peut écrire ses propres définitions de paquets (voir Section 9.2 [Définition des paquets], page 105) et les rendre disponibles dans des modules de paquets indépendants (voir Section 9.1 [Modules de paquets], page 104). Il est aussi *personnalisable* : on peut *dériver* des définitions de paquets spécialisées à partir de définitions existantes, même depuis la ligne de commande (voir Section 10.1.2 [Options de transformation de paquets], page 183).

Sous le capot, Guix implémente la discipline de *gestion de paquet fonctionnel* inventé par Nix (voir Chapitre 23 [Remerciements], page 701). Dans Guix le processus de construction et d’installation des paquets est vu comme une *fonction* dans le sens mathématique du terme. Cette fonction a des entrées (comme des scripts de construction, un compilateur et des bibliothèques) et renvoie un paquet installé. En tant que fonction pure, son résultat ne dépend que de ses entrées. Par exemple, il ne peut pas faire référence à des logiciels ou des scripts qui n’ont pas été explicitement passés en entrée. Une fonction de construction produit toujours le même résultat quand on lui donne le même ensemble d’entrée. Elle ne peut pas modifier l’environnement du système en cours d’exécution d’aucune manière ; par exemple elle ne peut pas créer, modifier ou supprimer des fichiers en dehors de ses

<sup>1</sup> « Guix » se prononce comme « geeks » (en prononçant le « s »), ou « iks » dans l’alphabet phonétique international (API).

<sup>2</sup> Nous appelions par le passé le système Guix « la distribution système Guix » ou « GuixSD ». Nous considérons maintenant qu’il est plus pertinent de tout regrouper sous la bannière de « Guix » comme, après tout, Guix System est directement disponible sous la commande `guix system`, même si vous utilisez une autre distro en dessous !

répertoires de construction et d'installation. Ce résultat s'obtient en lançant les processus de construction dans des environnements isolés (ou des *conteneurs*) où seules les entrées explicites sont visibles.

Le résultat des fonctions de construction de paquets est mis en *cache* dans le système de fichier, dans répertoire spécial appelé le *dépôt* (voir Section 9.9 [Le dépôt], page 157). Chaque paquet est installé dans son répertoire propre dans le dépôt — par défaut dans `/gnu/store`. Le nom du répertoire contient un hash de toutes les entrées utilisées pour construire le paquet ; ainsi, changer une entrée donnera un nom de répertoire différent.

Cette approche est le fondement des fonctionnalités les plus importante de Guix : le support des mises à jour des paquets et des retours en arrière transactionnels, l'installation différenciée par utilisateur-riche et le ramassage de miettes pour les paquets (voir Section 6.1 [Fonctionnalités], page 43).

## 1.2 Distribution GNU

Guix fournit aussi une distribution du système GNU contenant uniquement des logiciels libres<sup>3</sup>. On peut installer la distribution elle-même (voir Chapitre 3 [Installation du système], page 26), mais on peut aussi installer Guix comme gestionnaire de paquets par dessus un système GNU/Linux déjà installé (voir Chapitre 2 [Installation], page 5). Pour distinguer ces deux cas, on appelle la distribution autonome le « système Guix » ou Guix System.

La distribution fournit les paquets cœur de GNU comme la GNU libc, GCC et Binutils, ainsi que de nombreuses applications GNU et non-GNU. La liste complète des paquets disponibles se trouve en ligne (<http://www.gnu.org/software/guix/packages>) ou en lançant `guix package` (voir Section 6.2 [Invoquer guix package], page 44) :

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

Notre but est de fournir une distribution logicielle entièrement libre de GNU/Linux et d'autres variantes de GNU, en se concentrant sur la promotion et l'intégration étroite des composants GNU en insistant sur les programmes et les outils qui aident l'utilisateur-riche à exercer ses libertés.

Les paquets sont actuellement disponibles pour les plateformes suivantes :

`x86_64-linux`

Intel/AMD x86\_64 avec le noyau Linux-libre.

`i686-linux`

Architecture Intel 32 bits (IA32) avec le noyau Linux-libre.

`armhf-linux`

L'architecture ARMv7-A avec gestion des flottants matérielle, Thumb-2 et NEON, avec l'interface binaire applicative (ABI) EABI hard-float et le noyau Linux-libre.

`aarch64-linux`

les processeurs 64 bits ARMv8-A en little-endian, avec le noyau Linux-libre.

---

<sup>3</sup> Le terme « libre » se réfère ici bien sûr à la liberté offerte à la personne qui utilise ces logiciels (<http://www.gnu.org/philosophy/free-sw.fr.html>).

**i586-gnu** GNU/Hurd (<https://hurd.gnu.org>) sur l'architecture Intel 32-bit (IA32). Cette configuration en cours de développement est expérimentale. La manière la plus facile pour vous de l'essayer est de créer une instance `hurd-vm-service-type` sur votre machine GNU/Linux (voir [transparent-emulation-qemu], page 513). Voir Chapitre 22 [Contribuer], page 668, sur la façon d'aider !

**mips64el-linux** (non pris en charge)

les processeurs MIPS 64 bits little-endian, en particulier la série Loongson, l'ABI n32 et le noyau Linux-Libre. Cette configuration n'est plus entièrement prise en charge ; en particulier, il n'y a pas de travaux en cours pour s'assurer que cette architecture fonctionne encore. Si quelqu'un décide de faire revivre cette architecture, le code est toujours disponible.

**powerpc-linux** (non pris en charge)

les processeurs PowerPC 32 bits big-endian, en particulier PowerPC G4 avec la prise en charge d'AltVec et le noyau Linux-Libre. Cette configuration n'est pas entièrement prise en charge et il n'y a pas de travaux en cours pour s'assurer que cette architecture fonctionne.

**powerpc64le-linux**

Les processeurs little-endian et le noyau linux-Libre. Cela couvre les systèmes POWER9 comme la carte mère Talos II certifiée RYF (<https://www.fsf.org/news/talos-ii-mainboard-and-talos-ii-lite-mainboard-now-fsf-certified-to-respect->). Cette plateforme est disponible en tant que « démonstrateur technique » : bien qu'elle soit prise en charge, les substituts ne sont pas encore disponibles dans notre ferme de construction (voir Section 6.3 [Substituts], page 54) et certains paquets peuvent ne pas construire (voir Section 22.7 [Suivi des bogues et des correctifs], page 689). Cela dit, la communauté Guix travaille activement à améliorer cette prise en charge et c'est donc maintenant le bon moment pour l'essayer et participer !

**riscv64-linux**

Les processeurs RISC-V petit-boutistes, en particulier RV64GC, et le noyau linux-Libre. Cette plateforme est disponible en tant que « démonstrateur technique » : bien qu'elle soit prise en charge, les substituts ne sont pas encore disponibles dans notre ferme de construction (voir Section 6.3 [Substituts], page 54) et certains paquets peuvent ne pas construire (voir Section 22.7 [Suivi des bogues et des correctifs], page 689). Cela dit, la communauté Guix travaille activement à améliorer cette prise en charge et c'est donc maintenant le bon moment pour l'essayer et participer !

Avec Guix System, vous *déclarez* tous les aspects de la configuration du système d'exploitation et guix s'occupe d'instancier la configuration de manière transactionnelle, reproductible et sans état (voir Chapitre 12 [Configuration du système], page 238). Guix System utilise le noyau Linux-libre, le système d'initialisation Shepherd (voir Section "Introduction" dans *The GNU Shepherd Manual*), les outils GNU et la chaîne d'outils familière ainsi que l'environnement graphique et les services systèmes de votre choix.

Guix System est disponible sur toutes les plateformes ci-dessus à part `mips64el-linux`, `powerpc-linux`, `powerpc64le-linux` et `riscv64-linux`.

Pour des informations sur comment porter vers d'autres architectures et d'autres noyau, voir Chapitre 21 [Porter], page 667.

La construction de cette distribution est un effort collaboratif et nous vous invitons à nous rejoindre ! Voir Chapitre 22 [Contribuer], page 668, pour des informations sur la manière de nous aider.

## 2 Installation

**Remarque:** Nous vous recommandons d'utiliser ce script shell d'installation (<https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh>) pour installer Guix sur un système GNU/Linux fonctionnel, que nous appelons une *distro externe*<sup>1</sup>. Le script automatise le téléchargement, l'installation et la configuration initiale de Guix. Vous devez l'exécuter en tant que root.

Lorsqu'il est installé sur une distro externe, GNU Guix complète les outils disponibles sans interférence. Ses données se trouvent exclusivement dans deux répertoires, typiquement `/gnu/store` et `/var/guix`; les autres fichiers de votre système comme `/etc` sont laissés intacts.

Une fois installé, Guix peut être mis à jour en lançant `guix pull` (voir Section 6.6 [Invoquer `guix pull`], page 63).

Si vous préférez effectuer les étapes d'installation manuellement ou si vous voulez les personnaliser, vous trouverez les sections suivantes utiles. Elles décrivent les prérequis logiciels pour Guix, ainsi que la manière de l'installer manuellement et de se préparer à l'utiliser.

### 2.1 Installation binaire

Cette section décrit comment installer Guix sur un système quelconque depuis une archive autonome qui fournit les binaires pour Guix et toutes ses dépendances. C'est souvent plus rapide que d'installer depuis les sources, ce qui est décrit dans les sections suivantes. Le seul prérequis est d'avoir GNU tar et Xz.

**Remarque:** Nous recommandons l'utilisation de ce script d'installation du shell (<https://git.savannah.gnu.org/cgit/guix.git/plain/etc/guix-install.sh>). Ce script automatise les étapes de téléchargement, d'installation et de configuration initiale décrites ci-dessous. Il doit être exécuté sous l'utilisateur root. Vous pouvez donc l'exécuter en tant que root :

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

Si vous utilisez Debian ou une dérivée comme Ubuntu, vous pouvez plutôt installer le paquet (il se peut qu'il soit plus vieux que 1.4.0 mais vous pourrez le mettre à jour en lançant '`guix pull`') :

```
sudo apt install guix
```

De même sur openSUSE :

```
sudo zypper install guix
```

Lorsque vous aurez terminé, voir Section 2.6 [Réglages applicatifs], page 22, pour la configuration supplémentaire dont vous pourriez avoir besoin, et Chapitre 5 [Pour démarrer], page 40, pour vos premiers pas !

<sup>1</sup> Cette section s'occupe de l'installation du gestionnaire de paquet, ce qui peut se faire sur un système GNU/Linux existant. Si vous voulez plutôt installer le système d'exploitation GNU complet, voir Chapitre 3 [Installation du système], page 26.



L'installation se déroule comme ceci :

1. Télécharger le tarball binaire à partir de `'https://ftp.gnu.org/gnu/guix/guix-binary-1.4.0.x86_64-` où `x86_64-linux` peut être remplacé par `i686-linux` pour une i686 (32-bits) machine fonctionnant déjà avec le noyau Linux, et autres (voir Section 1.2 [Distribution GNU], page 2).

Assurez-vous de télécharger le fichier `.sig` associé et de vérifier l'authenticité de l'archive avec, comme ceci :

```
$ 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
```

Si cette commande échoue parce que vous n'avez pas la clef publique requise, lancez cette commande pour l'importer :

```
$ wget 'https://sv.gnu.org/people/viewgpg.php?user_id=15145' \
-q0 - | gpg --import -
```

et relancez la commande `gpg --verify`.

Remarquez qu'un avertissement du type « Cette clef n'est pas certifiée par une signature de confiance ! » est normal.

2. Maintenant, vous devez devenir `root`. En fonction de votre distribution, vous devrez lancer `su -` ou `sudo -i`. En tant que `root`, lancez :

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

Cela crée `/gnu/store` (voir Section 9.9 [Le dépôt], page 157) and `/var/guix`. Ce dernier contient un profil prêt à être utilisé pour `root` (voir les étapes suivantes).

Ne décompressez *pas* l'archive sur un système Guix lancé car cela écraserait ses propres fichiers essentiels.

L'option `--warning=no-timestamp` permet de s'assurer que GNU tar n'émet pas d'avertissements concernant des « horodatages manifestement anciens » (de tels avertissements ont été déclenchés par GNU tar 1.26 et plus anciens ; les versions récentes conviennent). Ils proviennent du fait que tous les fichiers de l'archive ont leur heure de modification fixée à un (ce qui signifie le 1er janvier 1970). Ceci est fait exprès pour s'assurer que le contenu de l'archive est indépendant de son temps de création, la rendant ainsi reproductible.

3. Rendez le profil disponible sous `~root/.config/guix/current`, qui est l'emplacement où `guix pull` installera les mises à jour (voir Section 6.6 [Invoquer guix pull], page 63) :

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

Agrémentez `etc/profile` pour augmenter `PATH` et les autres variables d'environnement nécessaires :

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

4. Créez le groupe et les comptes utilisés pour la construction comme expliqué plus loin (voir Section 2.4.1 [Réglages de l’environnement de construction], page 11).
5. Lancez le démon et paramétrez-le pour démarrer automatiquement au démarrage.

Si votre distribution hôte utilise le système d’initialisation `systemd`, cela peut se faire avec ces commandes :

```
# 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
```

Vous pourriez aussi vouloir faire en sorte que `guix gc` soit lancé périodiquement :

```
# 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
```

Vous voudrez peut-être éditer `guix-gc.service` pour ajuster la ligne de commande à vos besoins (voir Section 6.5 [Invoquer `guix gc`], page 60).

Si votre distribution hôte utilise le système d’initialisation `Upstart` :

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

Sinon, vous pouvez toujours démarrer le démon manuellement avec :

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

6. Rendez la commande `guix` disponible pour les autres personnes sur la machine, par exemple avec :

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

C’est aussi une bonne idée de rendre la version Info de ce manuel disponible ici :

```
# 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
```

Ainsi, en supposant que `/usr/local/share/info` se trouve dans le chemin de recherche, l’exécution de `info guix` ouvrira ce manuel (voir Section “`texinfo`” dans GNU `Texinfo`, pour plus de détails sur la modification du chemin de recherche Info).

7. Pour utiliser les substituts de `ci.guix.gnu.org`, `bordeaux.guix.gnu.org` ou l’un de leurs miroirs (voir Section 6.3 [Substituts], page 54), autorisez-les :

```
# 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
```

**Remarque:** Si vous n’activez pas les substituts, Guix va *tout* compiler depuis les sources sur votre machine, rendant très coûteuse chaque installation et chaque mise à jour. Voir Section 6.3.7 [De la confiance en des binaires], page 58, pour une discussion des raisons pour lesquelles on pourrait vouloir désactiver les substituts.

8. On peut avoir besoin d’effectuer des étapes supplémentaires pour que son environnement Guix soit prêt à être utilisé, voir Section 2.6 [Réglages applicatifs], page 22.

Voilà, l’installation est terminée !

Vous pouvez confirmer que Guix fonctionne en installant un paquet d’exemple dans le profil de root :

```
# guix install hello
```

L’archive d’installation binaire peut être (re)produite et vérifiée simplement en lançant la commande suivante dans l’arborescence des sources de Guix :

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

... ce qui à son tour lance :

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

Voir Section 8.3 [Invoquer guix pack], page 96, pour plus d’info sur cet outil pratique.

## 2.2 Prérequis

Cette section dresse la liste des prérequis pour la construction de Guix depuis les sources. La procédure de construction pour Guix est la même que pour les autres logiciels GNU, et n’est pas expliquée ici. Regardez les fichiers README et INSTALL dans l’arborescence des sources de Guix pour plus de détails.

GNU Guix est disponible au téléchargement depuis son site web sur <http://www.gnu.org/software/guix/>.

GNU Guix dépend des paquets suivants :

- GNU Guile (<https://gnu.org/software/guile/>), version 3.0.x ; version 3.0.3 ou supérieure ;
- Guile-Gcrypt (<https://notabug.org/cwebber/guile-gcrypt>), version 0.1.0 ou supérieure ;
- Guile-GnuTLS (<https://gitlab.com/gnutls/guile/>) (voir Section “Guile Preparations” dans *GnuTLS-Guile*)<sup>2</sup>;
- Guile-SQLite3 (<https://notabug.org/guile-sqlite3/guile-sqlite3>), version 0.1.0 ou supérieure ;
- Guile-zlib (<https://notabug.org/guile-zlib/guile-zlib>), version 0.1.0 ou supérieure ;
- Guile-lzlib (<https://notabug.org/guile-lzlib/guile-lzlib>) ;
- Guile-Avahi (<https://www.nongnu.org/guile-avahi/>) ;

---

<sup>2</sup> Les liaisons Guile de GnuTLS (<https://gnutls.org/>) faisaient partie de GnuTLS jusqu’à la version 3.7.8 incluse.

- Guile-Git (<https://gitlab.com/guile-git/guile-git>), version 0.5.0 ou supérieure ;
- Guile-JSON (<https://savannah.nongnu.org/projects/guile-json/>) 4.3.0 ou plus récent ;
- GNU Make (<https://www.gnu.org/software/make/>).

Les dépendances suivantes sont facultatives :

- Traitement du téléchargement des builds (voir Section 2.4.2 [Réglages du téléchargement du démon], page 12) et `guix copy` (voir Section 10.13 [Invoquer `guix copy`], page 229) dépend de Guile-SSH (<https://github.com/artiom-poptsov/guile-ssh>), version 0.13.0 ou ultérieure.
- Guile-zstd (<https://notabug.org/guile-zstd/guile-zstd>), pour la compression et la décompression `zstd` dans `guix publish` et pour les substituts (voir Section 10.11 [Invoquer `guix publish`], page 222).
- Guile-Semver (<https://ngyro.com/software/guile-semver.html>) pour l'importateur `crate` (voir Section 10.5 [Invoquer `guix import`], page 197).
- Guile-Lib (<https://www.nongnu.org/guile-lib/doc/ref/htmlprag/>) pour l'importateur `go` (voir Section 10.5 [Invoquer `guix import`], page 197) et pour certains programmes de mise à jour (voir Section 10.6 [Invoquer `guix refresh`], page 205).
- Lorsque `libbz2` (<http://www.bzip.org>) est disponible, `guix-daemon` peut l'utiliser pour compresser les journaux de construction.

Sauf si `--disable-daemon` a été passé à `configure`, les paquets suivants sont également nécessaires :

- GNU `libcrypt` (<https://gnupg.org/>) ;
- SQLite 3 (<https://sqlite.org/>) ;
- GCC's `g++` (<https://gcc.gnu.org>), avec le support pour le Standard C++11.

Lorsque vous configurez Guix sur un système qui a déjà une installation de Guix, assurez-vous de spécifier le même répertoire d'état que l'installation existante avec l'option `--localstatedir` du script `configure` (voir Section “Directory Variables” dans *GNU Coding Standards*). Habituellement cette option `localstatedir` est initialisée à `/var`. Le script `configure` vous protège des configurations involontaires de `localstatedir` pour éviter que vous ne corrompiez votre dépôt (voir Section 9.9 [Le dépôt], page 157).

## 2.3 Lancer la suite de tests

Après avoir lancé `configure` et `make` correctement, c'est une bonne idée de lancer la suite de tests. Elle peut aider à trouver des erreurs avec la configuration ou l'environnement, ou des bogues dans Guix lui-même — et vraiment, rapporter des échecs de tests est une bonne manière d'aider à améliorer le logiciel. Pour lancer la suite de tests, tapez :

```
make check
```

Les cas de tests peuvent être lancés en parallèle : vous pouvez utiliser l'option `-j` de GNU `make` pour accélérer les choses. Le premier lancement peut prendre plusieurs minutes sur une machine récente ; les lancements suivants seront plus rapides car le dépôt créé pour les tests aura déjà plusieurs choses en cache.

Il est aussi possible de lancer un sous-ensemble des tests en définissant la variable makefile `TESTS` comme dans cet exemple :

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

Par défaut, les résultats des tests sont affichés au niveau du fichier. Pour voir les détails de chaque cas de test individuel, il est possible de définir la variable makefile `SCM_LOG_DRIVER_FLAGS` comme dans cet exemple :

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

Le pilote de tests Automake personnalisé avec SRFI 64 utilisé pour la suite de tests « `check` » (située dans `build-aux/test-driver.scm`) permet aussi de choisir les cas de test à lancer à plus fine granularité, via ses options `--select` et `--exclude`. Voici un exemple, pour lancer tous les cas de tests du fichier de tests `tests/packages.scm` dont les noms commencent par « `transaction-upgrade-entry` » :

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

Pour celles et ceux qui veulent inspecter les résultats des tests échoués directement depuis la ligne de commande, il est possible d’ajouter l’option `--errors-only=yes` à la variable `SCM_LOG_DRIVER_FLAGS` du Makefile et d’initialiser la variable Automake du Makefile `VERBOSE`, de cette manière :

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

L’option `--show-duration=yes` peut être utilisée pour afficher la durée des cas de test individuels, en combinaison avec `--brief=no` :

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

Voir Section “Parallel Test Harness” dans *GNU Automake* pour plus d’information sur le banc de tests parallèle d’Automake.

Après un échec, envoyez un courriel à [bug-guix@gnu.org](mailto:bug-guix@gnu.org) et attachez le fichier `test-suite.log`. Précisez la version de Guix utilisée ainsi que les numéros de version de ses dépendances (voir Section 2.2 [Prérequis], page 8) dans votre message.

Guix possède aussi une suite de tests de systèmes complets qui test des instances complètes du système Guix. Elle ne peut être lancée que sur un système où Guix est déjà installé, avec :

```
make check-system
```

ou, de nouveau, en définissant `TESTS` pour choisir un sous-ensemble des tests à lancer :

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

Ces tests systèmes sont définis dans les modules (`gnu tests ...`). Ils fonctionnent en lançant les systèmes d’exploitation sous test avec une instrumentation légère dans une machine virtuelle (VM). Ils peuvent être intenses en terme de calculs ou plutôt rapides en fonction de la disponibilité des substituts de leurs dépendances (voir Section 6.3 [Substituts], page 54). Certains requièrent beaucoup d’espace disque pour contenir les images des VM.

De nouveau, en cas d’échec, envoyez tous les détails à [bug-guix@gnu.org](mailto:bug-guix@gnu.org).

## 2.4 Paramétrer le démon

Les opérations comme la construction d'un paquet ou le lancement du ramasse-miettes sont toutes effectuées par un processus spécialisé, le *démon de construction*, pour le compte des clients. Seul le démon peut accéder au dépôt et à sa base de données associée. Ainsi, toute opération manipulant le dépôt passe par le démon. Par exemple, les outils en ligne de commande comme `guix package` et `guix build` communiquent avec le démon (*via* des appels de procédures distantes) pour lui dire quoi faire.

Les sections suivantes expliquent comment préparer l'environnement du démon de construction. Voir aussi Section 6.3 [Substituts], page 54, pour apprendre comment permettre le téléchargement de binaires pré-construits.

### 2.4.1 Réglages de l'environnement de construction

Dans une installation standard multi-utilisateur-*rice-s*, Guix et son démon — le programme `guix-daemon` — sont installés par la personne qui administre le système ; `/gnu/store` appartient à `root` et `guix-daemon` est lancé en `root`. Les utilisateur-*rice-s* non-privilegié-e-s peuvent utiliser les outils Guix pour construire des paquets ou accéder au dépôt et le démon le fera pour leur compte en s'assurant que le dépôt garde un état cohérent et permet le partage des paquets déjà construits entre les utilisateur-*rice-s*.

Alors que `guix-daemon` tourne en `root`, vous n'avez pas forcément envie que les processus de construction de paquets tournent aussi en `root`, pour des raisons de sécurité évidentes. Pour éviter cela, vous devriez créer une réserve spéciale de *comptes de construction* que les processus de construction démarrés par le démon utiliseront. Ces comptes de construction n'ont pas besoin d'un shell ou d'un répertoire personnel ; ils seront seulement utilisés quand le démon délaissera ses privilèges `root` dans les processus de construction. En ayant plusieurs de ces comptes, vous permettez au démon de lancer des processus de construction distincts sous des UID différent, ce qui garanti qu'aucune interférence n'ait lieu entre les uns et les autres — une fonctionnalité essentielle puisque les constructions sont supposées être des fonctions pures (voir Chapitre 1 [Introduction], page 1).

Sur un système GNU/Linux, on peut créer une réserve de comptes de construction comme ceci (avec la syntaxe Bash et les commandes `shadow`) :

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

Le nombre de comptes de construction détermine le nombre de tâches de constructions qui peuvent tourner en parallèle, tel que spécifié par l'option `--max-jobs` (voir Section 2.5 [Invoquer `guix-daemon`], page 17). Pour utiliser `guix system vm` et les commandes liées, vous devrez ajouter les comptes de construction au groupe `kvm` pour qu'ils puissent accéder à `/dev/kvm` avec `-G guixbuild,kvm` plutôt que `-G guixbuild` (voir Section 12.15 [Invoquer `guix system`], page 584).

Le programme `guix-daemon` peut ensuite être lancé en `root` avec la commande suivante<sup>3</sup> :

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

De cette façon, le démon démarre les processus de construction dans un chroot, sous un des comptes `guixbuilder`. Sur GNU/Linux par défaut, l'environnement chroot ne contient rien d'autre que :

- un répertoire `/dev` minimal, créé presque indépendamment du `/dev` de l'hôte<sup>4</sup> ;
- le répertoire `/proc` ; il ne montre que les processus du conteneur car on utilise une espace de nom séparé pour les PID ;
- `/etc/passwd` avec une entrée pour le compte actuel et une entrée pour le compte `nobody` ;
- `/etc/group` avec une entrée pour le groupe de ce compte ;
- `/etc/hosts` avec une entrée qui fait correspondre `localhost` à `127.0.0.1` ;
- un répertoire `/tmp` inscriptible.

Le chroot ne contient pas de dossier `/home`, et la variable d'environnement `HOME` est initialisée au répertoire `/homeless-shelter` inexistant. Cela permet de mettre en valeur les utilisations inappropriées de `HOME` dans les scripts de construction des paquets.

Vous pouvez influencer le répertoire où le démon stocke les arbres de construction *via* la variable d'environnement `TMPDIR`. Cependant, l'arbre de construction dans le chroot sera toujours appelé `/tmp/guix-build-nom.drv-0`, où *nom* est le nom de la dérivation — p.ex., `coreutils-8.24`. De cette façon, la valeur de `TMPDIR` ne fuit pas à l'intérieur des environnements de construction, ce qui évite des différences lorsque le processus de construction retient le nom de leur répertoire de construction.

Le démon prend aussi en compte la variable d'environnement `https_proxy` pour ses téléchargements HTTP et HTTPS, que ce soit pour les dérivations à sortie fixes (voir Section 9.10 [Dérivations], page 159) ou pour les substituts (voir Section 6.3 [Substituts], page 54).

Si vous installez Guix en tant qu'utilisateur-*rice* non privilégié-ée, il est toujours possible d'exécuter `guix-daemon` à condition de passer `--disable-chroot`. Cependant, les processus de compilation ne seront pas isolés les uns des autres, ni du reste du système. Ainsi, les processus de compilation peuvent interférer les uns avec les autres, et peuvent accéder à des programmes, des bibliothèques et d'autres fichiers disponibles sur le système - ce qui rend beaucoup plus difficile de les considérer comme des fonctions *pures*.

## 2.4.2 Utiliser le dispositif de téléchargement

Si vous le souhaitez, le démon de construction peut *décharger* des constructions de dérivation sur d'autres machines Guix avec le *crochet de construction offload*<sup>5</sup>. Lorsque

<sup>3</sup> Si votre machine utilise le système d'initialisation `systemd`, copiez le fichier `prefix/lib/systemd/system/guix-daemon.service` dans `/etc/systemd/system` pour vous assurer que `guix-daemon` est démarré automatiquement. De même, si votre machine utilise le système d'initialisation `Upstart`, copiez le fichier `prefix/lib/upstart/system/guix-daemon.conf` dans `/etc/init`.

<sup>4</sup> « presque », parce que même si l'ensemble des fichiers qui apparaissent dans le `/dev` du chroot sont déterminés à l'avance, la plupart de ces fichiers ne peut pas être créée si l'hôte ne les a pas.

<sup>5</sup> Cette fonctionnalité n'est disponible que si `Guile-SSH` (<https://github.com/artiom-poptsov/guile-ssh>) est présent.

cette fonctionnalité est activée, Guix lit une liste de machines de constructions spécifiée par l'utilisateur-riche dans `/etc/guix/machines.scm` ; à chaque fois qu'une construction est demandée, par exemple par `guix build`, le démon essaie de la télécharger sur une des machines qui satisfont les contraintes de la dérivation, en particulier le type de système, p. ex. `x86_64-linux`. Une même machine peut avoir plusieurs types de systèmes, soit parce que son architecture le supporte nativement, soit par émulation (voir [transparent-emulation-qemu], page 513), soit les deux. Les prérequis manquants pour la construction sont copiés par SSH sur la machine de construction qui procède ensuite à la construction ; si elle réussit, les sorties de la construction sont copiés vers la machine de départ. Le dispositif de téléchargement est dotée d'un scheduler de base qui tente de sélectionner la meilleure machine. La meilleure machine est choisie parmi les machines disponibles sur la base de critères tels que :

1. La disponibilité d'un créneau de construction. Une machine de construction peut avoir autant de slots de construction (connexions) que la valeur du champ `parallel-builds` de son objet `build-machine`.
2. Sa vitesse relative, telle que définie dans le champ `speed` de son objet `build-machine`.
3. Sa charge. La charge normalisée de la machine doit être inférieure à une valeur seuil, configurable via le champ `overload-threshold` de son objet `build-machine`.
4. Disponibilité de l'espace disque. Plus de 100 Mio doivent être disponibles.

Le fichier `/etc/guix/machines.scm` ressemble typiquement à cela :

```
(list (build-machine
      (name "eightysix.example.org")
      (system "x86_64-linux")
      (host-key "ssh-ed25519 AAAAC3Nza...")
      (user "bob")
      (speed 2.))      ;incroyablement rapide !

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

      ;; Rappelez-vous que « guix offload » est démarré par
      ;; « guix-daemon » en root.
      (private-key "/root/.ssh/identité-pour-guix"))))
```

Dans l'exemple ci-dessus nous spécifions une liste de deux machines de construction, une pour l'architecture `x86_64` et `i686` et une pour l'architecture `aarch64`.

En fait, ce fichier est — et ça ne devrait pas vous surprendre ! — un fichier Scheme qui est évalué au démarrage du crochet `offload`. Sa valeur de retour doit être une liste d'objets `build-machine`. Même si cet exemple montre une liste fixée de machines de construction, on pourrait imaginer par exemple utiliser DNS-SD pour renvoyer une liste de machines de constructions potentielles découvertes sur le réseau local (voir Section "Introduction" dans *Using Avahi in Guile Scheme Programs*). Le type de données `build-machine` est détaillé plus bas.



**build-machine** [Type de données]

Ce type de données représente les machines de construction sur lesquelles le démon peut télécharger des constructions. Les champs importants sont :

- name** Le nom d'hôte de la machine distante.
- systèmes** Le type de système de la machine distante, p. ex., (`list "x86_64-linux" "i686-linux"`).
- user** Le compte à utiliser lors de la connexion à la machine distante par SSH. Remarquez que la paire de clef SSH ne doit *pas* être protégée par mot de passe pour permettre des connexions non-interactives.
- host-key** Cela doit être la *clef d'hôte SSH publique* de la machine au format OpenSSH. Elle est utilisée pour authentifier la machine lors de la connexion. C'est une longue chaîne qui ressemble à cela :

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

Si la machine utilise le démon OpenSSH, `sshd`, la clef d'hôte se trouve dans un fichier comme `/etc/ssh/ssh_host_ed25519_key.pub`.

Si la machine utilise le démon SSH de GNU `lsh`, la clef d'hôte est dans `/etc/lsh/host-key.pub` ou un fichier similaire. Elle peut être convertie au format OpenSSH avec `lsh-export-key` (voir Section "Converting keys" dans *LSH Manual*) :

```
$ lsh-export-key --openssh < /etc/lsh/host-key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAAE0p8FoQAAAEAs1eB46LV...
```

Il y a un certain nombre de champs facultatifs que vous pouvez remplir :

**port** (par défaut : 22)

Numéro de port du serveur SSH sur la machine.

**private-key** (par défaut : `~root/.ssh/id_rsa`)

Le fichier de clef privée SSH à utiliser lors de la connexion à la machine, au format OpenSSH. Cette clef ne doit pas être protégée par phrase de passe.

Remarquez que la valeur par défaut est la clef privée *du compte root*. Assurez-vous qu'elle existe si vous utilisez la valeur par défaut.

**compression** (par défaut : `"zlib@openssh.com,zlib"`)

**compression-level** (par défaut : 3)

Les méthodes de compression au niveau SSH et le niveau de compression demandé.

Remarquez que le téléchargement utilise la compression SSH pour réduire la bande passante utilisée lors du transfert vers et depuis les machines de construction.

**daemon-socket** (par défaut : `"/var/guix/daemon-socket/socket"`)

Le nom de fichier du socket Unix-domain sur lequel `guix-daemon` écoute sur cette machine.

`overload-threshold` (par défaut : 0.8)

Le seuil de charge au-dessus duquel une machine de déchargement potentielle est ignorée par le programme de déchargement. Cette valeur se traduit approximativement par l'utilisation totale du processeur de la machine de construction, allant de 0,0 (0%) à 1,0 (100%). Elle peut également être désactivée en réglant `overload-threshold` sur `#f`.

`parallel-builds` (par défaut : 1)

Le nombre de constructions qui peuvent tourner simultanément sur la machine.

`speed` (par défaut : 1.0)

Un « facteur de vitesse relatif ». L'ordonnanceur des constructions tendra à préférer les machines avec un plus grand facteur de vitesse.

`features` (par défaut : '()')

Une liste de chaînes qui contient les fonctionnalités spécifiques supportées par la machine. Un exemple est "kvm" pour les machines qui ont le module Linux KVM et le support matériel correspondant. Les dérivations peuvent demander des fonctionnalités par leur nom et seront orchestrées sur les machines de construction correspondantes.

La commande `guix` doit être dans le chemin de recherche des machines de construction. Vous pouvez vérifier si c'est le cas en lançant :

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

Il reste une dernière chose à faire maintenant que `machines.scm` est en place. Comme expliqué ci-dessus, lors du déchargement les fichiers sont transférés entre les dépôts des machines. Pour que cela fonctionne, vous devez d'abord générer une paire de clef sur chaque machine pour permettre au démon d'exporter des archives signées des fichiers de son dépôt (voir Section 6.10 [Invoquer `guix archive`], page 71) :

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

Chaque machine de construction doit autoriser la clef de la machine maîtresse pour qu'ils acceptent les éléments de dépôt de celle-ci :

```
# guix archive --authorize < master-public-key.txt
```

De même, la machine maîtresse doit autoriser les clefs de chaque machine de construction.

Toute cette histoire de clefs permet d'exprimer la confiance mutuelle deux-à-deux entre le maître et les machines de construction. Concrètement, lorsque le maître reçoit des fichiers d'une machine de construction (et vice-versa), son démon de construction s'assure qu'ils sont authentiques, n'ont pas été modifiés par un tiers et qu'il sont signés par un clef autorisée.

Pour tester que votre paramétrage fonctionne, lancez cette commande sur le nœud maître :

```
# guix offload test
```

Cela essaiera de se connecter à toutes les machines de construction spécifiées dans `/etc/guix/machines.scm`, s'assurera que Guix est disponible sur toutes les machines et tentera d'exporter vers la machine et d'importer depuis elle, et rapportera toute erreur survenu pendant le processus.

Si vous souhaitez tester un fichier de machines différent, spécifiez-le sur la ligne de commande :

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

Enfin, vous pouvez tester un sous-ensemble de machines dont le nom correspond à une expression rationnelle comme ceci :

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

Pour afficher la charge actuelle de tous les hôtes de construction, lancez cette commande sur le nœud principal :

```
# guix offload status
```

### 2.4.3 Support de SELinux

Guix inclus un fichier de politique SELinux dans `etc/guix-daemon.cil` qui peut être installé sur un système où SELinux est activé pour que les fichiers Guix soient étiquetés et pour spécifier le comportement attendu du démon. Comme Guix System ne fournit pas de politique SELinux de base, la politique du démon ne peut pas être utilisée sur le système Guix.

#### 2.4.3.1 Installer la politique SELinux

Pour installer la politique, lancez cette commande en root :

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

Puis ré-étiquetez le système de fichier avec `restorecon` ou par un mécanisme différent fournit par votre système.

Une fois la politique installée, le système de fichier ré-étiqueté et le démon redémarré, il devrait être lancé dans le contexte `guix_daemon_t`. Vous pouvez le confirmer avec la commande suivante :

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

Surveillez les fichiers journaux de SELinux pendant que vous lancez une commande comme `guix build hello` pour vous convaincre que SELinux permet toutes les opérations nécessaires.

#### 2.4.3.2 Limitations

La politique n'est pas parfaite. Voici une liste de limitations et de bizarreries qui vous devriez prendre en compte avant de déployer la politique SELinux fournie pour le démon Guix.

1. `guix_daemon_socket_t` n'est pas vraiment utilisé. Aucune des opérations sur les sockets n'impliquent de contextes qui ont quoi que ce soit à voir avec `guix_daemon_socket_t`. Ça ne fait pas de mal d'avoir une étiquette inutilisée, mais il serait préférable de définir des règles sur les sockets uniquement pour cette étiquette.
2. `guix gc` ne peut pas accéder à n'importe quel lien vers les profils. Par conception, l'étiquette de fichier de la destination d'un lien symbolique est indépendant de l'étiquette du lien lui-même. Bien que tous les profils sous `$localstatedir` aient une étiquette, les liens vers ces profils héritent de l'étiquette du répertoire dans lequel ils se trouvent. Pour les liens dans le répertoire personnel cela sera `user_home_t`. Mais pour les liens du répertoire personnel de root, ou `/tmp`, ou du répertoire de travail du

serveur HTTP, etc, cela ne fonctionnera pas. SELinux empêcherait `guix gc` de lire et de suivre ces liens.

3. La fonctionnalité du démon d'écouter des connexions TCP pourrait ne plus fonctionner. Cela demande des règles supplémentaires car SELinux traite les sockets réseau différemment des fichiers.
4. Actuellement tous les fichiers qui correspondent à l'expression rationnelle `/gnu/store/.+-(guix-+|profile)/bin/guix-daemon` reçoivent l'étiquette `guix_daemon_exec_t` ; cela signifie que *tout* fichier avec ce nom dans n'importe quel profil serait autorisé à se lancer dans le domaine `guix_daemon_t`. Ce n'est pas idéal. Un attaquant pourrait construire un paquet qui fournit cet exécutable et convaincre un-e utilisateur-riche de l'installer et de le lancer, ce qui l'élève dans le domaine `guix_daemon_t`. À ce moment SELinux ne pourrait pas l'empêcher d'accéder à des fichiers autorisés pour les processus de ce domaine.

Vous devrez renommer le répertoire du dépôt après chaque mise à jour de `guix-daemon`, par exemple après avoir lancé `guix pull`. En supposant que le dépôt est dans `/gnu`, vous pouvez le faire avec `restorecon -vR /gnu`, ou par d'autres moyens fournis par votre système d'exploitation.

Nous pourrions générer une politique bien plus restrictive à l'installation, pour que seuls les noms de fichiers *exacts* de l'exécutable `guix-daemon` actuellement installé soit étiqueté avec `guix_daemon_exec_t`, plutôt que d'utiliser une expression rationnelle plus large. L'inconvénient c'est que root devrait installer ou mettre à jour la politique à l'installation à chaque fois que le paquet Guix qui fournit l'exécutable `guix-daemon` effectivement exécuté est mis à jour.

## 2.5 Invoquer `guix-daemon`

Le programme `guix-daemon` implémente toutes les fonctionnalités d'accès au dépôt. Cela inclus le lancement des processus de construction, le lancement du ramasse-miettes, la demande de disponibilité des résultats de construction, etc. Il tourne normalement en `root` comme ceci :

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

Ce démon peut aussi être démarré avec le protocole d'« activation par socket » de `systemd` (voir Section “Service De- and Constructors” dans *le manuel de GNU Shepherd*).

Pour des détails sur son paramétrage, voir Section 2.4 [Paramétrer le démon], page 11.

Par défaut, `guix-daemon` lance les processus de construction sous différents UIDs récupérés depuis le groupe de construction spécifié avec `--build-users-group`. En plus, chaque processus de construction est lancé dans un environnement `chroot` qui ne contient que le sous-ensemble du dépôt dont le processus de construction dépend, tel que spécifié par sa dérivation (voir Chapitre 9 [Interface de programmation], page 104), plus un ensemble de répertoires systèmes spécifiques. Par défaut ce dernier contient `/dev` et `/dev/pts`. De plus, sous GNU/Linux, l'environnement de construction est un *conteneur* : en plus d'avoir sa propre arborescence du système de fichier, il a un espace de nom de montage séparé, son propre espace de nom PID, son espace de nom de réseau, etc. Cela aide à obtenir des constructions reproductibles (voir Section 6.1 [Fonctionnalités], page 43).

Lorsque le démon effectue une construction pour le compte de l'utilisateur-riche, il crée un répertoire de construction sous `/tmp` ou sous le répertoire spécifié par sa variable

d'environnement `TMPDIR`. Ce répertoire est partagé avec le conteneur pendant toute la durée de la construction, bien que dans le conteneur, l'arbre de compilation soit toujours appelé `/tmp/guix-build-name.drv-0`.

Le répertoire de construction est automatiquement supprimé à la fin, à moins que la construction n'ait échoué et que le client ait spécifié `--keep-failed` (voir Section 10.1.1 [Options de construction communes], page 180).

Le démon écoute les connexions et démarre un sous-processus pour chaque session démarrée par un client (l'une des sous-commandes de `guix`). La commande `guix processes` vous permet d'obtenir un aperçu de l'activité sur votre système en affichant chaque session et client actifs. Voir Section 10.16 [Invoquer `guix processes`], page 233, pour plus d'informations.

Les options en ligne de commande suivantes sont disponibles :

**`--build-users-group=groupe`**

Utiliser les comptes du *groupe* pour lancer les processus de construction (voir Section 2.4 [Paramétrer le démon], page 11).

**`--no-substitutes`**

Ne pas utiliser de substitut pour les résultats de la construction. C'est-à-dire, toujours construire localement plutôt que de permettre le téléchargement de binaires pré-construits (voir Section 6.3 [Substituts], page 54).

Lorsque le démon est lancé avec `--no-substitutes`, les clients peuvent toujours activer explicitement la substitution *via* l'appel de procédure distante `set-build-options` (voir Section 9.9 [Le dépôt], page 157).

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

Considérer *urls* comme la liste séparée par des espaces des URL des sources de substituts par défaut. Lorsque cette option est omise, `'https://ci.guix.gnu.org https://bordeaux.guix.gnu.org'` est utilisé.

Cela signifie que les substituts sont téléchargés depuis les *urls*, tant qu'ils sont signés par une signature de confiance (voir Section 6.3 [Substituts], page 54).

Voir Section 6.3.3 [Récupérer des substituts d'autres serveurs], page 56, pour plus d'information sur la configuration du démon pour récupérer des substituts d'autres serveurs.

**`--no-offload`**

N'essaye pas de décharger les constructions vers d'autres machines (voir Section 2.4.2 [Réglages du déchargement du démon], page 12). C'est-à-dire que tout sera construit localement au lieu de décharger les constructions à une machine distante.

**`--cache-failures`**

Mettre les échecs de construction en cache. Par défaut, seules les constructions réussies sont mises en cache.

Lorsque cette option est utilisée, `guix gc --list-failures` peut être utilisé pour demander l'ensemble des éléments du dépôt marqués comme échoués ; `guix gc --clear-failures` vide la liste des éléments aillant échoué. Voir Section 6.5 [Invoquer `guix gc`], page 60.

- cores=*n***  
**-c *n*** Utiliser *n* cœurs CPU pour construire chaque dérivation ; 0 signifie autant que possible.  
La valeur par défaut est 0, mais elle peut être annulée par les clients, comme avec l'option **--cores** de **guix build** (voir Section 10.1 [Invoquer guix build], page 180).  
L'effet est de définir la variable d'environnement `NIX_BUILD_CORES` dans le processus de construction, qui peut ensuite l'utiliser pour exploiter le parallélisme en interne — par exemple en lançant **make -j\$NIX\_BUILD\_CORES**.
- max-jobs=*n***  
**-M *n*** Permettre au plus *n* travaux de construction en parallèle. La valeur par défaut est 1. La mettre à 0 signifie qu'aucune construction ne sera effectuée localement ; à la place, le démon déchargera les constructions (voir Section 2.4.2 [Réglages du déchargement du démon], page 12) ou échouera.
- max-silent-time=secondes**  
Lorsque le processus de construction ou de substitution restent silencieux pendant plus de *secondes*, le terminer et rapporter une erreur de construction.  
La valeur par défaut est 0, ce qui désactive le délai.  
La valeur spécifiée ici peut être annulée par les clients (voir Section 10.1.1 [Options de construction communes], page 180).
- timeout=secondes**  
De même, lorsque le processus de construction ou de substitution dure plus de *secondes*, le terminer et rapporter une erreur de construction.  
La valeur par défaut est 0, ce qui désactive le délai.  
La valeur spécifiée ici peut être annulée par les clients (voir Section 10.1.1 [Options de construction communes], page 180).
- rounds=*N***  
Construire chaque dérivation *N* fois à la suite, et lever une erreur si les résultats de construction consécutifs ne sont pas identiques bit-à-bit. Remarquez que ce paramètre peut être modifié par les clients comme **guix build** (voir Section 10.1 [Invoquer guix build], page 180).  
Lorsqu'utilisé avec **--keep-failed**, la sortie différente est gardée dans le dépôt sous `/gnu/store/...-check`. Cela rend plus facile l'étude des différences entre les deux résultats.
- debug** Produire une sortie de débogage.  
Cela est utile pour déboguer des problèmes de démarrage du démon, mais ensuite elle peut être annulée par les clients, par exemple par l'option **--verbosity** de **guix build** (voir Section 10.1 [Invoquer guix build], page 180).
- chroot-directory=rép**  
Ajouter *rép* au chroot de construction.  
Cela peut changer le résultat d'un processus de construction — par exemple s'il utilise une dépendance facultative trouvée dans *rép* lorsqu'elle est disponible ou

pas sinon. Pour cette raison, il n'est pas recommandé d'utiliser cette option. À la place, assurez-vous que chaque dérivation déclare toutes les entrées dont elle a besoin.

**--disable-chroot**

Désactive les constructions dans un chroot.

Utiliser cette option n'est pas recommandé car, de nouveau, elle permet aux processus de construction d'accéder à des dépendances non déclarées. Elle est nécessaire cependant lorsque `guix-daemon` tourne sans privilèges.

**--log-compression=type**

Compresser les journaux de construction suivant le *type*, parmi `gzip`, `bzip2` ou `none`.

À moins que `--lose-logs` ne soit utilisé, tous les journaux de construction sont gardés dans `localstatedir`. Pour gagner de la place, le démon les compresse automatiquement avec `gzip` par défaut.

**--discover[=yes|no]**

Indique s'il faut découvrir les serveurs de substitut sur le réseau local avec mDNS et DNS-SD.

Cette fonction est encore expérimentale. Cependant, voici quelques réflexions sur le sujet.

1. Cela peut être plus rapide ou moins cher que la récupération depuis des serveurs distants ;
2. Il n'y a pas de risque de sécurité, seuls des substituts authentiques seront utilisés (voir Section 6.3.4 [Authentification des substituts], page 57) ;
3. Un·e attaquant·e qui publierait `guix publish` sur votre LAN ne peut pas vous proposer de binaire malveillants, mais il ou elle pourrait apprendre quels logiciels vous installez ;
4. Les serveurs peuvent servir des substituts en HTTP, sans chiffrement, donc n'importe qui sur votre LAN peut voir quels logiciels vous installez.

Il est aussi possible d'activer ou de désactiver la découverte de serveurs de substituts à l'exécution en lançant :

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

**--disable-deduplication**

Désactiver la « déduplication » automatique des fichiers dans le dépôt.

Par défaut, les fichiers ajoutés au dépôt sont automatiquement « dédupliqués » : si un nouveau fichier est identique à un autre fichier trouvé dans le dépôt, le démon en fait un lien en dur vers l'autre fichier. Cela réduit considérablement l'utilisation de l'espace disque au prix d'une charge en entrée/sortie plus grande à la fin d'un processus de construction. Cette option désactive cette optimisation.

**--gc-keep-outputs[=yes|no]**

Dire si le ramasse-miettes (GC) doit garder les sorties des dérivations utilisées.

Lorsqu'il est réglé sur **yes**, le GC conservera les sorties de toute dérivation active disponibles dans le dépôt—les fichiers `.drv`. La valeur par défaut est **no**, ce qui signifie que les sorties des dérivations ne sont conservées que si elles sont accessibles à partir d'une racine GC. Voir Section 6.5 [Invoquer guix gc], page 60, pour en savoir plus sur les racines GC.

`--gc-keep-derivations[=yes|no]`

Dire si le ramasse-miettes (GC) doit garder les dérivations correspondant à des sorties utilisées.

Lorsqu'il est réglé à « **yes** », comme c'est le cas par défaut, le GC garde les dérivations — c.-à-d. les fichiers `.drv` — tant qu'au moins une de leurs sorties est utilisée. Cela permet de garder une trace de l'origine des éléments du dépôt. Le mettre à **no** préserve un peu d'espace disque.

De cette manière, le réglage de l'option `--gc-keep-derivations` sur **yes** étend le résultat des sorties aux dérivations, et le réglage de l'option `--gc-keep-outputs` sur **yes** étend le résultat des dérivations aux sorties. Lorsque les deux sont réglés sur **yes**, l'effet est de conserver tous les prérequis de construction (les sources, le compilateur, les bibliothèques et autres outils de construction) des objets actifs dans le dépôt, que ces prérequis soient accessibles ou non depuis une racine GC. Cela est pratique pour les développeurs car cela permet d'éviter les reconstructions ou les téléchargements.

`--impersonate-linux-2.6`

Sur les systèmes basés sur Linux, imiter Linux 2.6. Cela signifie que l'appel système `uname` du noyau indiquera 2.6 comme numéro de version.

Cela peut être utile pour construire des programmes qui dépendent (généralement sans fondement) du numéro de version du noyau.

`--lose-logs`

Ne pas garder les journaux de construction. Par défaut ils sont gardés dans `localstatedir/guix/log`.

`--system=système`

Supposer que `système` est le type de système actuel. Par défaut c'est la paire architecture-noyau trouvée à la configuration, comme `x86_64-linux`.

`--listen=extrémité`

Écouter les connexions sur `extrémité`. `extrémité` est interprété comme un nom de fichier d'un socket Unix-domain s'il commence par `/` (barre oblique). Sinon, `extrémité` est interprété comme un nom de domaine ou d'hôte et un port sur lequel écouter. Voici quelques exemples :

`--listen=/gnu/var/daemon`

Écouter les connexions sur le socket Unix-domain `/gnu/var/daemon` en le créant si besoin.

`--listen=localhost`

Écouter les connexions TCP sur l'interface réseau correspondant à `localhost` sur le port 44146.



```
--listen=128.0.0.42:1234
```

Écouter les connexions TCP sur l'interface réseau correspondant à 128.0.0.42 sur le port 1234.

Cette option peut être répétée plusieurs fois, auquel cas `guix-daemon` accepte des connexions sur tous les paramètres spécifiés. On peut indiquer aux commandes clientes à quoi se connecter en paramétrant la variable d'environnement `GUIX_DAEMON_SOCKET` (voir Section 9.9 [Le dépôt], page 157).

**Remarque:** Le protocole du démon est *non authentifié et non chiffré*. Utiliser `--listen=host` est adapté sur des réseaux locaux, comme pour des grappes de serveurs, où seuls des nœuds de confiance peuvent se connecter au démon de construction. Dans les autres cas où l'accès à distance au démon est requis, nous conseillons d'utiliser un socket Unix-domain avec SSH.

Lorsque `--listen` est omis, `guix-daemon` écoute les connexions sur le socket Unix-domain situé à `localstatedir/guix/daemon-socket/socket`.

## 2.6 Réglages applicatifs

Lorsque vous utilisez Guix par dessus une distribution GNU/Linux qui n'est pas Guix System — ce qu'on appelle une *distro externe* — quelques étapes supplémentaires sont requises pour que tout soit en place. En voici certaines.

### 2.6.1 Régionalisation

Les paquets installés *via* Guix n'utiliseront pas les données de régionalisation du système hôte. À la place, vous devrez d'abord installer l'un des paquets linguistiques disponibles dans Guix puis définir la variable d'environnement `GUIX_LOCPATH` :

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

Remarquez que le paquet `glibc-locales` contient les données de tous les paramètres linguistiques pris en charge par la GNU libc et pèse environ 930 Mio<sup>6</sup>. Si vous n'avez besoin que de quelques paramètres, vous pouvez définir un paquet personnalisé avec la procédure `make-glibc-utf8-locales` du module (`gnu packages base`). L'exemple suivant définit un paquet contenant plusieurs paramètres linguistiques canadiens connus de GNU libc, qui pèse environ 14 Mio :

```
(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-canadian-utf8-locales"))
```

<sup>6</sup> La taille du paquet `glibc-locales` est réduite à environ 213 Mio avec la déduplication du dépôt et encore à 67 Mio si vous utilisez le système de fichiers Btrfs compressé avec `zstd`

La variable `GUIX_LOCPATH` joue un rôle similaire à `LOCPATH` (voir Section “Locale Names” dans *The GNU C Library Reference Manual*). Il y a deux différences importantes cependant :

1. `GUIX_LOCPATH` n’est pris en compte que par la `libc` dans Guix et pas par la `libc` fournie par les distros externes. Ainsi, utiliser `GUIX_LOCPATH` vous permet de vous assurer que les programmes de la distro externe ne chargeront pas de données linguistiques incompatibles.
2. la `libc` ajoute un suffixe `/X.Y` à chaque entrée de `GUIX_LOCPATH`, où `X.Y` est la version de la `libc` — p. ex. `2.22`. Cela signifie que, si votre profile Guix contient un mélange de programmes liés avec des versions différentes de la `libc`, chaque version de la `libc` essaiera de charger les environnements linguistiques dans le bon format.

Cela est important car le format des données linguistiques utilisés par différentes version de la `libc` peuvent être incompatibles.

## 2.6.2 Name Service Switch

Lorsque vous utilisez Guix sur une distro externe, nous *recommandons fortement* que ce système fasse tourner le *démon de cache de service de noms* de la bibliothèque C de GNU, `nscd`, qui devrait écouter sur le socket `/var/run/nscd/socket`. Sans cela, les applications installées avec Guix peuvent échouer à résoudre des noms d’hôtes ou de comptes, ou même planter. Les paragraphes suivants expliquent pourquoi.

La bibliothèque C de GNU implémente un *name service switch* (NSS), qui est un mécanisme d’extension pour les « résolutions de noms » en général : résolution de nom d’hôte, de compte utilisateur·rice et plus (voir Section “Name Service Switch” dans *The GNU C Library Reference Manual*).

Comme il est extensible, NSS supporte des *greffons* qui fournissent une nouvelle implémentation de résolution de nom : par exemple le greffon `nss-mdns` permet la résolution de noms d’hôtes en `.local`, le greffon `nis` permet la résolution de comptes avec le Network Information Service (NIS), etc. Ces « services de recherches » supplémentaires sont configurés au niveau du système dans `/etc/nsswitch.conf`, et tous les programmes qui tournent sur ce système prennent en compte ces paramètres (voir Section “NSS Configuration File” dans *The GNU C Reference Manual*).

Lorsqu’ils essayent d’effectuer une résolution de nom — par exemple en appelant la fonction `getaddrinfo` en C — les applications essayent d’abord de se connecter au `nscd` ; en cas de réussite, `nscd` effectue la résolution de nom pour eux. Si le `nscd` ne tourne pas, alors ils effectuent la résolution eux-mêmes, en changeant les service de résolution dans leur propre espace d’adressage et en le lançant. Ces services de résolution de noms — les fichiers `libnss_*.so` — sont `dlopenés` mais ils peuvent provenir de la bibliothèque C du système, plutôt que de la bibliothèque C à laquelle l’application est liée (la bibliothèque C de Guix).

Et c’est là que se trouve le problème : si votre application est liée à la bibliothèque C de Guix (disons, `glibc-2.24`) et essaye de charger les greffons NSS d’une autre bibliothèque C (disons, `libnss_mdns.so` pour `glibc-2.22`), il est très probable qu’elle plante ou que sa résolution de nom échoue de manière inattendue.

Lancer `nscd` sur le système, entre autres avantages, élimine ce problème d’incompatibilité binaire car ces fichiers `libnss_*.so` sont chargés par le processus `nscd`, pas par l’application elle-même.

### 2.6.3 Polices X11

La majorité des applications graphiques utilisent `fontconfig` pour trouver et charger les polices et effectuer le rendu côté client X11. Le paquet `fontconfig` dans Guix cherche les polices dans `$HOME/.guix-profile` par défaut. Ainsi, pour permettre aux applications graphiques installées avec Guix d’afficher des polices, vous devez aussi installer des polices avec Guix. Les paquets de polices essentiels sont `font-ghostscript`, `font-dejavu` et `font-gnu-freefont`.

Lorsque vous installez ou supprimez des polices, ou lorsque vous remarquez qu’une application ne trouve pas les polices, vous pouvez avoir besoin d’installer `Fontconfig` et de forcer un rafraîchissement de son cache de police avec :

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

Pour afficher des textes écrits en chinois, en japonais ou en coréen dans les applications graphiques, installez `font-adobe-source-han-sans` ou `font-wqy-zenhei`. Le premier a plusieurs sorties, une par famille de langue (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59). Par exemple, la commande suivante installe les polices pour le chinois :

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

Les vieux programmes comme `xterm` n’utilisent pas `fontconfig` et s’appuient sur le rendu du côté du serveur. Ces programmes ont besoin de spécifier le nom complet de la police en utilisant XLF D (X Logical Font Description), comme ceci :

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

Pour pouvoir utiliser ces noms complets avec les polices TrueType installées dans votre profil Guix, vous devez étendre le chemin des polices du serveur X :

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

Ensuite, vous pouvez lancer `xlsfonts` (du paquet `xlsfonts`) pour vous assurer que vos polices TrueType y sont listées.

### 2.6.4 Certificats X.509

Le paquet `nss-certs` fournit les certificats X.509 qui permettent aux programmes d’authentifier les serveurs web par HTTPS.

Lorsque vous utilisez Guix sur une distribution externe, vous pouvez installer ce paquet et définir les variables d’environnement adéquates pour que les paquets sachent où trouver les certificats. Voir Section 12.11 [Certificats X.509], page 572, pour des informations détaillées.

### 2.6.5 Paquets Emacs

Quand vous installez des paquets Emacs avec Guix, les fichiers Elisp sont placés dans le répertoire `share/emacs/site-lisp/` du profil dans lequel ils sont installés. Les bibliothèques Elisp sont rendues disponibles dans Emacs avec la variable d’environnement `EMACSLLOADPATH`, qui est initialisée à l’installation d’Emacs lui-même.

En plus, des définitions `autoload` sont automatiquement évaluées au démarrage d’Emacs, par la procédure `guix-emacs-autoload-packages` de Guix. Si, pour quelque raison que ce soit, vous souhaitez éviter de charger automatiquement les paquets Emacs installés avec Guix, vous pouvez le faire en lançant Emacs avec l’option `--no-site-file` (voir Section “Init File” dans *The GNU Emacs Manual*).

**Remarque:** Emacs peut maintenant compiler des paquets nativement. Dans la configuration par défaut, cela signifie que les paquets Emacs seront maintenant compilés à la volée (JIT) quand vous les utilisez, et les résultats seront stockés dans un sous-répertoire de votre `user-emacs-directory`.

De plus, le système de construction des paquets Emacs prend en charge la compilation native de manière transparente, mais remarquez que `emacs-minimal` — l’Emacs par défaut pour construire les paquets — n’a pas été configuré avec la compilation native. Pour compiler nativement vos paquets emacs en avance, utilisez une transformation comme `--with-input=emacs-minimal=emacs`.

## 2.7 Mettre à niveau Guix

Pour mettre Guix à niveau, lancez :

```
guix pull
```

Voir Section 6.6 [Invoquer guix pull], page 63, pour plus d’informations.

Sur une distribution externe, vous pouvez mettre à jour le démon de construction en lançant :

```
sudo -i guix pull
```

suivi de (dans le cas où votre distribution utilise l’outil de gestion de services Systemd) :

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

Sur Guix System, la mise à jour du démon est effectuée par la reconfiguration du système (voir Section 12.15 [Invoquer guix system], page 584).

## 3 Installation du système

Cette section explique comment installer Guix System sur une machine. Guix, en tant que gestionnaire de paquets, peut aussi être installé sur un système GNU/Linux déjà installé, voir Chapitre 2 [Installation], page 5.

### 3.1 Limitations

Nous considérons Guix System comme prêt pour une grande variété de cas d'utilisation pour le « bureau » et le serveur. Les garanties de fiabilité qu'il fournit — les mises à jour transactionnelles, les retours en arrière et la reproductibilité — en font une solide fondation.

Néanmoins, avant de procéder à l'installation, soyez conscient de ces limitations les plus importantes qui s'appliquent à la version 1.4.0 :

- De plus en plus de services systèmes sont fournis (voir Section 12.9 [Services], page 267) mais certains manquent toujours cruellement.
- GNOME, Xfce, LXDE et Enlightenment sont disponibles (voir Section 12.9.9 [Services de bureaux], page 347), ainsi qu'un certain nombre de gestionnaires de fenêtres X11. Cependant, il manque actuellement KDE.

Plus qu'un avertissement, c'est une invitation à rapporter les problèmes (et vos succès !) et à nous rejoindre pour améliorer la distribution. Voir Chapitre 22 [Contribuer], page 668, pour plus d'info.

### 3.2 Considérations matérielles

GNU Guix se concentre sur le respect des libertés de ses utilisateurs et utilisatrices. Il est construit autour du noyau Linux-libre, ce qui signifie que seuls les matériels pour lesquels des pilotes logiciels et des micrologiciels libres sont disponibles sont pris en charge. De nos jours, une grande gamme de matériel qu'on peut acheter est prise en charge par GNU/Linux-libre — des claviers aux cartes graphiques en passant par les scanners et les contrôleurs Ethernet. Malheureusement, il reste des produits dont les fabricants refusent de laisser le contrôle aux utilisateur·rice·s sur leur propre utilisation de l'ordinateur, et ces matériels ne sont pas pris en charge par Guix System.

L'un des types de matériels où les pilotes ou les micrologiciels sont le moins disponibles sont les appareils WiFi. Les appareils WiFi connus pour fonctionner sont ceux qui utilisent des puces Atheros (AR9271 et AR7010) qui correspondent au pilote `ath9k` de Linux-libre, et ceux qui utilisent des puces Broadcom/AirForce (BCM43xx avec la révision Wireless-Core 5), qui correspondent au pilote `b43-open` de Linux-libre. Des micrologiciels libres existent pour les deux et sont disponibles directement sur Guix System, dans `%base-firmware` (voir Section 12.2 [référence de operating-system], page 246).

L'installateur vous averti rapidement s'il détecte des périphériques connus pour ne *pas* marcher à cause du manque de micrologiciel ou de pilote libre.

La Free Software Foundation (<https://www.fsf.org/>) a un programme de certification nommé *Respects Your Freedom* (<https://www.fsf.org/ryf>) (RYF), pour les produits matériels qui respectent votre liberté et votre vie privée en s'assurant que vous avez le contrôle sur l'appareil. Nous vous encourageons à vérifier la liste des appareils certifiés par RYF.

Une autre ressource utile est le site web H-Node (<https://www.h-node.org/>). Il contient un catalogue d'appareils avec des informations sur leur support dans GNU/Linux.

### 3.3 Installation depuis une clef USB ou un DVD

Une image d'installation ISO-9660 qui peut être écrite sur une clé USB ou être gravée sur un DVD est téléchargeable à partir de '[https://ftp.gnu.org/gnu/guix/guix-system-install-1.4.0.x86\\_64-](https://ftp.gnu.org/gnu/guix/guix-system-install-1.4.0.x86_64-)' où vous pouvez remplacer `x86_64-linux` par l'un des éléments suivants :

`x86_64-linux`

pour un système GNU/Linux sur un CPU compatible Intel/AMD 64-bits ;

`i686-linux`

pour un système GNU/Linux sur un CPU compatible Intel 32-bits.

Assurez-vous de télécharger les fichiers `.sig` associés et de vérifier l'authenticité de l'image avec, de cette manière :

```
$ 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
```

Si cette commande échoue parce que vous n'avez pas la clef publique requise, lancez cette commande pour l'importer :

```
$ wget https://sv.gnu.org/people/viewgpg.php?user_id=15145 \
-q0 - | gpg --import -
```

et relancez la commande `gpg --verify`.

Remarquez qu'un avertissement du type « Cette clef n'est pas certifiée par une signature de confiance ! » est normal.

Cette image contient les outils nécessaires à l'installation. Elle est faite pour être copiée *telle quelle* sur une clef USB assez grosse ou un DVD.

#### Copie sur une clef USB

Insérez la clef USB de 1 Gio ou plus dans votre machine et déterminez son nom d'appareil. En supposant que la clef usb est connue sous le nom de `/dev/sdX`, copiez l'image avec :

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

Accéder à `/dev/sdX` requiert généralement les privilèges d'administration.

#### Graver sur un DVD

Insérez un DVD vierge dans votre machine et déterminez son nom d'appareil. En supposant que le DVD soit connu sous le nom de `/dev/srX`, copiez l'image avec :

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

Accéder à `/dev/srX` requiert généralement les privilèges root.

#### Démarrage

Une fois que c'est fait, vous devriez pouvoir redémarrer le système et démarrer depuis la clé USB ou le DVD. Pour cela, vous devrez généralement entrer dans le menu de démarrage BIOS ou UEFI, où vous pourrez choisir de démarrer sur la clé USB. Pour démarrer depuis

Libreboot, passez en mode de commande en appuyant sur la touche `c` et en tapant `search grub usb`.

Voir Section 3.8 [Installer Guix dans une VM], page 36, si, à la place, vous souhaitez installer Guix System dans une machine virtuelle (VM).

### 3.4 Préparer l'installation

Une fois que vous avez démarré, vous pouvez utiliser l'installateur graphique, qui rend facile la prise en main (voir Section 3.5 [Installation graphique guidée], page 28). Sinon, si vous êtes déjà familier avec GNU/Linux et que vous voulez plus de contrôle que ce que l'installateur graphique propose, vous pouvez choisir le processus d'installation « manuel » (voir Section 3.6 [Installation manuelle], page 30).

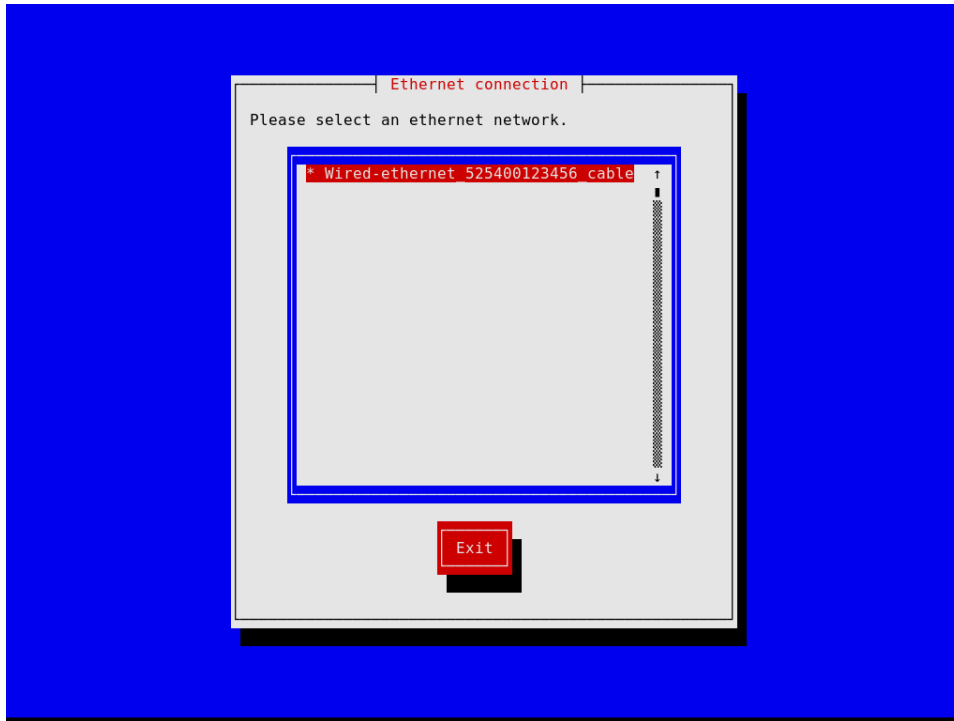
L'installateur graphique est disponible sur le TTY1. Vous pouvez obtenir des shells root sur les TTY 3 à 6 en tapant `ctrl-alt-f3`, `ctrl-alt-f4` etc. Le TTY2 affiche cette documentation que vous pouvez atteindre avec `ctrl-alt-f2`. On peut naviguer dans la documentation avec les commandes du lecteur Info (voir *Stand-alone GNU Info*). Le démon de souris GPM tourne sur le système d'installation, ce qui vous permet de sélectionner du texte avec le bouton gauche de la souris et de le coller en appuyant sur la molette.

**Remarque:** L'installation nécessite un accès au réseau pour que les dépendances manquantes de votre configuration système puissent être téléchargées. Voyez la section « réseau » plus bas.

### 3.5 Installation graphique guidée

L'installateur graphique est une interface utilisateur en mode texte. Il vous guidera, avec des boîtes de dialogue, le long des étapes requises pour installer GNU Guix System.

La première boîte de dialogue vous permet de paramétrer le système comme vous le souhaitez pendant l'installation : vous pouvez choisir la langue, la disposition du clavier et paramétrer le réseau, qui sera utilisé pendant l'installation. L'image ci-dessous montre le dialogue pour le réseau.

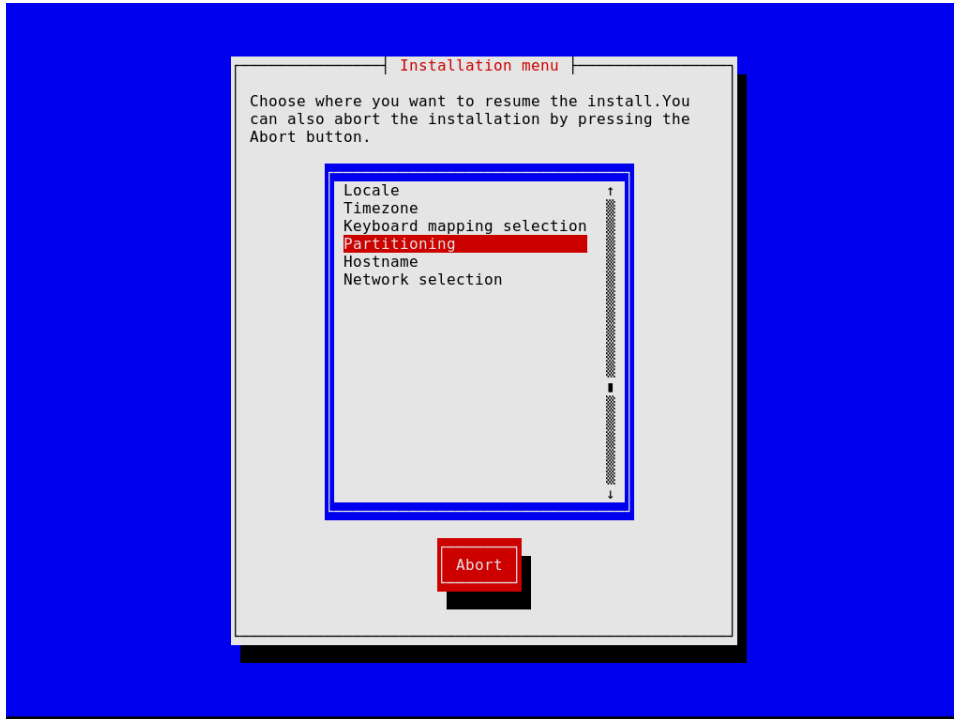


Les étapes suivantes vous permettent de partitionner votre disque dur, comme le montre l'image ci-dessous, de choisir si vous voulez ou non utiliser des systèmes de fichiers chiffrés, de saisir le nom d'hôte et le mot de passe root et de créer un compte supplémentaire, entre autres choses.





Remarquez que, à tout moment, l'installateur vous permet de sortir de l'étape d'installation actuelle et de recommencer une étape précédente, comme le montre l'image ci-dessous.



Une fois que vous avez fini, l'installateur produit une configuration de système d'exploitation et vous la montre (voir Section 12.1 [Utiliser le système de configuration], page 238). À ce moment, vous pouvez appuyer sur « OK » et l'installation continuera. Lorsqu'elle aura réussi, vous pourrez redémarrer sur le nouveau système et vous amuser. Voir Section 3.7 [Après l'installation du système], page 35, pour la suite des festivités !

## 3.6 Installation manuelle

Cette section décrit comme vous pourriez installer « manuellement » GNU Guix System sur votre machine. Cette option nécessite que vous soyez familier avec GNU/Linux, le shell et avec les outils d'administration usuels. Si vous pensez que ce n'est pas pour vous, pensez à utiliser l'installateur graphique (voir Section 3.5 [Installation graphique guidée], page 28).

Le système d'installation fournit des shells root sur les TTY 3 à 6 ; appuyez sur `ctrl-alt-f3`, `ctrl-alt-f4` etc pour y accéder. Il inclut plusieurs outils usuels pour requis pour cette tâche. Mais c'est aussi un système Guix complet. Cela signifie que vous pouvez installer des paquets supplémentaires si vous en avez besoin, avec `guix package` (voir Section 6.2 [Invoquer guix package], page 44).

### 3.6.1 Disposition du clavier réseau et partitionnement

Avant que vous ne puissiez installer le système, vous voudrez sans doute ajuster la disposition du clavier, paramétrer le réseau et partitionner le disque dur cible. Cette section vous guidera à travers tout cela.

### 3.6.1.1 Disposition du clavier

L'image d'installation utilise la disposition clavier qwerty (US). Si vous voulez la changer, vous pouvez utiliser la commande `loadkeys`. Par exemple, la commande suivante sélectionne la disposition Dvorak :

```
loadkeys dvorak
```

Consultez les fichiers dans `/run/current-system/profile/share/keymaps` pour trouver une liste des dispositions disponibles. Lancez `man loadkey` pour plus d'informations.

### 3.6.1.2 Réseau

Lancez la commande suivante pour voir comment vos interfaces réseau sont appelées :

```
ifconfig -a
```

... ou, avec la commande spécifique à GNU/Linux `ip` :

```
ip address
```

Les interfaces filaires ont un nom qui commence par 'e' ; par exemple, l'interface qui correspond au premier contrôleur Ethernet sur la carte mère est appelé 'eno1'. Les interfaces sans-fil ont un nom qui commence par 'w', comme 'w1p2s0'.

Connexion filaire

Pour configurer une connexion filaire, lancez la commande suivante, en remplaçant *interface* par le nom de l'interface filaire que vous voulez utiliser.

```
ifconfig interface up
```

... ou, avec la commande spécifique à GNU/Linux `ip` :

```
ip link set interface up
```

Connexion sans-fil

Pour configurer le réseau sans-fil, vous pouvez créer un fichier de configuration pour l'outil de configuration `wpa_supplicant` (son emplacement importe peu) avec l'un des éditeurs de texte disponibles comme `nano` :

```
nano wpa_supplicant.conf
```

Par exemple, la déclaration qui suit peut aller dans ce fichier et fonctionnera pour plusieurs réseaux sans-fil, si vous donnez le vrai SSID et la phrase de passe pour le réseau auquel vous vous connectez :

```
network={
    ssid="mon-ssid"
    key_mgmt=WPA-PSK
    psk="la phrase de passe secrète du réseau"
}
```

Démarrez le service sans-fil et lancez-le en tâche de fond avec la commande suivante (en remplaçant *interface* par le nom de l'interface réseau que vous voulez utiliser) :

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

Lancez `man wpa_supplicant` pour plus d'informations.

À partir de ce moment, vous avez besoin d'une adresse IP. Sur les réseaux où les IP sont automatiquement attribuée par DHCP, vous pouvez lancer :

```
dhclient -v interface
```

Essayez de pinger un serveur pour voir si le réseau fonctionne :

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

Mettre en place un accès réseau est presque toujours une nécessité parce que l'image ne contient pas tous les logiciels et les outils dont vous pourriez avoir besoin.

Si vous avez besoin d'un accès HTTP et HTTPS pour passer à travers un proxy, lancez la commande suivante :

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

où *URL* est l'URL du proxy, par exemple `http://example.org:8118`.

Si vous le souhaitez, vous pouvez continuer l'installation à distance en démarrant un serveur SSH :

```
herd start ssh-daemon
```

Assurez-vous soit de définir un mot de passe avec `passwd`, soit de configurer l'authentification par clef OpenSSH avant de vous connecter.

### 3.6.1.3 Partitionnement

À moins que vous ne l'ayez déjà fait, l'étape suivante consiste à partitionner le disque puis à formater les partitions cibles.

L'image d'installation inclus plusieurs outils de partitionnement, dont `Parted` (voir Section “Overview” dans *GNU Parted User Manual*), `fdisk`, et `cgdisk`. Lancez-en un et paramétrez votre disque avec le partitionnement qui vous convient :

```
cgdisk
```

Si votre disque utilise le format des tables de partitions GUID (GPT) et que vous souhaitez installer un GRUB pour système BIOS (c'est le cas par défaut), assurez-vous de créer qu'une partition de démarrage BIOS soit bien disponible (voir Section “BIOS installation” dans *GNU GRUB manual*).

Si vous souhaitez à la place utilise GRUB pour système EFI, vous devrez avoir une *partition système EFI* (ESP) en FAT32. Cette partition peut être montée dans `/boot/efi` par exemple et doit avoir le drapeau `esp`. P. ex. pour `parted` :

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

#### Remarque:

Vous n'êtes pas sûr de savoir si vous devez utiliser un GRUB EFI ou BIOS ? Si le répertoire `/sys/firmware/efi` existe sur l'image d'installation, vous devriez probablement effectuer une installation EFI, avec `grub-efi-bootloader`. Sinon, vous devriez utiliser le GRUB en BIOS, `grub-bootloader`. Voir Section 12.14 [Configuration du chargeur d'amorçage], page 578, pour plus d'information sur le chargeur d'amorçage.

Une fois que vous avez fini le partitionnement du disque dur cible, vous devez créer un système de fichier sur les partitions<sup>1</sup>. Pour l'ESP, si vous en avez une et en supposant que ce soit `/dev/sda1`, lancez :

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

<sup>1</sup> Actuellement Guix System ne prend en charge que les systèmes de fichiers `ext4`, `btrfs`, `JFS`, `F2FS` et `XFS`. En particulier, le code qui lit les UUID des systèmes de fichiers et les étiquettes ne fonctionne que pour ces types de systèmes de fichiers.

Concernant le système de fichier root, ext4 est le format le plus largement utilisé. D'autres systèmes de fichiers, comme Btrfs, supportent la compression, laquelle est reportée pour compléter agréablement la déduplication que le démon réalise indépendamment du système de fichier (voir Section 2.5 [Invoquer guix-daemon], page 17).

Préférez assigner une étiquette au système de fichier pour que vous puissiez vous y référer de manière fiable dans la déclaration `file-system` (voir Section 12.3 [Systèmes de fichiers], page 250). On le fait habituellement avec l'option `-L` de `mkfs.ext4` et des commandes liées. Donc, en supposant que la partition racine soit sur `/dev/sda2`, on peut créer un système de fichier avec pour étiquette `my-root` avec :

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

Si vous voulez plutôt chiffrer la partition root, vous pouvez utiliser les utilitaires `Cryptsetup` et `LUKS` pour cela (voir `man cryptsetup` pour plus d'informations).

**Attention:** Remarquez que GRUB peut déverrouiller des périphériques LUKS2 depuis la version 2.06, mais il prend en charge uniquement la fonction de dérivation de clé PBKDF2, qui n'est pas celle par défaut de `cryptsetup luksFormat`. Vous pouvez vérifier quelle fonction de dérivation de clé est utilisée par un périphérique avec `cryptsetup luksDump périphérique`, et en regardant le champ PBKDF de vos emplacements de clés.

En supposant que vous voulez stocker la partition racine sur `/dev/sda2`, voici la séquence de commandes pour la formater en partition LUKS2 :

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

Une fois cela effectué, montez le système de fichier cible dans `/mnt` avec une commande comme (de nouveau, en supposant que `my-root` est l'étiquette du système de fichiers racine) :

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

Montez aussi tous les systèmes de fichiers que vous voudriez utiliser sur le système cible relativement à ce chemin. Si vous avez choisi d'avoir un `/boot/efi` comme point de montage EFI par exemple, montez-la sur `/mnt/boot/efi` maintenant pour qu'elle puisse être trouvée par `guix system init` ensuite.

Enfin, si vous souhaitez utiliser une ou plusieurs partitions de swap (voir Section 12.5 [Espace d'échange], page 258), assurez-vous de les initialiser avec `mkswap`. En supposant que vous avez une partition de swap sur `/dev/sda3`, vous pouvez lancer :

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

Autrement, vous pouvez utiliser un fichier de swap. Par exemple, en supposant que dans le nouveau système vous voulez utiliser le fichier `/swapfile` comme fichier de swap, vous lanceriez<sup>2</sup> :

```
# Cela représente 10 Gio d'espace d'échange. Ajustez « count » pour changer la taille.
```

<sup>2</sup> Cet exemple fonctionnera sur plusieurs types de systèmes de fichiers (p. ex. ext4). Cependant, pour les systèmes de fichiers qui utilisent la copie sur écriture (COW) comme btrfs, les étapes requises peuvent varier. Pour plus de détails, regardez les pages de manuel de `mkswap` et `swapon`.

```
dd if=/dev/zero of=/mnt/swapfile bs=1MiB count=10240
# Par sécurité, laissez le fichier en lecture et en écriture uniquement pour root.
chmod 600 /mnt/swapfile
mkswap /mnt/swapfile
swapon /mnt/swapfile
```

Remarquez que si vous avez chiffré la partition racine et créé un fichier d'échange dans son système de fichier comme décrit ci-dessus, alors le chiffrement protégera aussi le fichier d'échange, comme n'importe quel fichier de ce système de fichiers.

### 3.6.2 Effectuer l'installation

Lorsque la partition cible est prête et que les autres partitions sont montées, on est prêt à commencer l'installation. Commencez par :

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

Cela rend `/gnu/store` capable de faire de la copie sur écriture, de sorte que les paquets ajoutés pendant l'installation sont écrits sur le disque cible sur `/mnt` plutôt que gardés en mémoire. Cela est nécessaire parce que la première phase de la commande `guix system init` (voir plus bas) implique de télécharger ou de construire des éléments de `/gnu/store` qui est initialement un système de fichiers en mémoire.

Ensuite, vous devrez modifier un fichier et fournir la déclaration du système à installer. Pour cela, le système d'installation propose trois éditeurs de texte. Nous recommandons GNU nano (voir *GNU nano Manual*), qui supporte la coloration syntaxique la correspondance de parenthèses ; les autres éditeurs sont `mg` (un clone d'Emacs) et `nvi` (un clone de l'éditeur `vi` original de BSD). Nous recommandons vivement de stocker ce fichier sur le système de fichier racine cible, disons en tant que `/mnt/etc/config.scm`. Sinon, vous perdrez votre fichier de configuration une fois que vous aurez redémarré sur votre nouveau système.

Voir Section 12.1 [Utiliser le système de configuration], page 238, pour un aperçu de comment créer votre fichier de configuration. Les exemples de configuration dont on parle dans cette section sont disponibles dans `/etc/configuration` sur l'image d'installation. Ainsi, pour commencer avec une configuration du système qui fournit un serveur d'affichage graphique (un système de « bureau »), vous pouvez lancer ce qui suit :

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

Vous devriez faire attention à ce que contient votre fichier de configuration, en particulier :

- Assurez-vous que la forme `bootloader-configuration` se réfère aux cibles où vous voulez installer GRUB. Elle devrait aussi mentionner `grub-bootloader` si vous installez GRUB en mode BIOS (ou « legacy ») ou `grub-efi-bootloader` pour les systèmes UEFI plus récents. Pour les anciens systèmes, le champ `targets` contient le nom des périphériques comme `(list "/dev/sda")` ; pour les systèmes UEFI il contient les chemins vers les partitions EFI montées, comme `(list "/boot/efi")` ; assurez-vous bien que ces chemins sont montés et qu'il y a une entrée `file-system` dans votre configuration.

- Assurez-vous que les étiquettes de vos systèmes de fichiers correspondent aux valeurs de leur champs `device` dans votre configuration `file-system`, en supposant que la configuration `file-system` utilise la procédure `file-system-label` dans son champ `device`.
- Si vous avez des partitions RAID ou chiffrées, assurez-vous d'ajouter un champ `mapped-device` pour les décrire (voir Section 12.4 [Périphériques mappés], page 256).

Une fois que vous avez fini les préparatifs sur le fichier de configuration, le nouveau système peut être initialisé (rappelez-vous que le système de fichiers racine cible est dans `/mnt`) :

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

Cela copie tous les fichiers nécessaires et installe GRUB sur `/dev/sdX` à moins que vous ne passiez l'option `--no-bootloader`. Pour plus d'informations, voir Section 12.15 [Invoquer `guix system`], page 584. Cette commande peut engendrer des téléchargements ou des constructions pour les paquets manquants, ce qui peut prendre du temps.

Une fois que cette commande a terminé — et on l'espère réussi ! — vous pouvez lancer `reboot` et démarrer sur votre nouveau système. Le mot de passe `root` est d'abord vide ; les mots de passe des autres comptes doivent être initialisés avec la commande `passwd` en tant que `root`, à moins que votre configuration ne spécifie autre chose (voir [user-account-password], page 262). Voir Section 3.7 [Après l'installation du système], page 35, pour la suite !

### 3.7 Après l'installation du système

Bravo ! Vous avez maintenant redémarré sur votre système Guix ! À partir de maintenant, vous pouvez mettre à jour le système quand vous voudrez, avec :

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

Cela crée une nouvelle génération du système avec les derniers paquets et services (voir Section 12.15 [Invoquer `guix system`], page 584). Nous vous recommandons de le faire régulièrement pour que votre système inclue les dernières mises à jour de sécurité (voir Chapitre 19 [Mises à jour de sécurité], page 657).

#### Remarque:

Remarquez que `sudo guix` exécute la commande `guix` de votre compte et *non* celle de `root`, parce que `sudo` ne change pas `PATH`. Pour utiliser explicitement le `guix` de `root`, tapez `sudo -i guix . . .`

La différence est importante ici, car `guix pull` met à jour la commande `guix` et les définitions de paquets uniquement pour l'utilisateur sous lequel elle est exécutée. Cela signifie que si vous choisissez d'utiliser la commande `guix system reconfigure` dans le shell de connexion de `root`, vous devrez utiliser séparément la commande `guix pull`.

Maintenant, voir Chapitre 5 [Pour démarrer], page 40, et rejoignez-nous sur `#guix` sur le réseau IRC Libera Chat ou sur `guix-devel@gnu.org` pour partager votre expérience !

### 3.8 Installer Guix sur une machine virtuelle

Si vous souhaitez installer Guix System sur une machine virtuelle (VM) ou un serveur privé virtuel (VPS) plutôt que sur votre machine chérie, cette section est faite pour vous.

Pour démarrer une VM QEMU (<https://qemu.org/>) pour installer Guix System sur une image disque, suivez ces étapes :

1. Tout d'abord récupérez et décompressez l'image d'installation du système Guix comme décrit précédemment (voir Section 3.3 [Installation depuis une clef USB ou un DVD], page 27).
2. Créez une image disque qui contiendra le système installé. Pour créer une image qcow2, utilisez la commande `qemu-img` :

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

Le fichier qui en résulte sera bien plus petit que les 50 Go (habituellement moins de 1 Mo) mais il grossira au fur et à mesure que le stockage virtuel grossira.

3. Démarrez l'image d'installation USB dans une VM :

```
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.système.iso
```

`-enable-kvm` est facultatif, mais améliore nettement les performances, voir Section 12.17 [Lancer Guix dans une VM], page 597.

4. Vous êtes maintenant root dans la VM, continuez en suivant la procédure d'installation. Voir Section 3.4 [Préparer l'installation], page 28, et suivez les instructions.

Une fois l'installation terminée, vous pouvez démarrer le système dans votre image `guix-system.img`. Voir Section 12.17 [Lancer Guix dans une VM], page 597, pour une manière de faire.

### 3.9 Construire l'image d'installation

L'image d'installation décrite plus haut a été construite avec la commande `guix system`, plus précisément :

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

Regardez le fichier `gnu/system/install.scm` dans l'arborescence des sources et regardez aussi Section 12.15 [Invoquer `guix system`], page 584, pour plus d'informations sur l'image d'installation.

### 3.10 Construire l'image d'installation pour les cartes ARM

De nombreuses cartes ARM requièrent une variante spécifique du chargeur d'amorçage U-Boot (<https://www.denx.de/wiki/U-Boot/>).

Si vous construisez une image disque et que le chargeur d'amorçage n'est pas disponible autrement (sur un autre périphérique d'amorçage etc), il est recommandé de construire une image qui inclut le chargeur d'amorçage, plus précisément :

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

A20-0Linuxino-Lime2 est le nom de la carte. Si vous spécifiez une carte invalide, une liste de cartes possibles sera affichée.



## 4 Corriger les problèmes du système

Le système Guix permet de redémarrer sur une génération précédente si la dernière ne fonctionne pas, ce qui le rend assez robuste contre les défaillances irréversibles. Cette fonctionnalité dépend du bon fonctionnement de GRUB cependant, ce qui signifie que si pour quelque raison que ce soit, votre installation de GRUB est corrompue pendant une reconfiguration du système, vous pourriez ne pas pouvoir facilement démarrer une génération précédente. Une technique qui peut être utilisée dans ce cas est de *chrooter* dans votre système endommagé et de le reconfigurer à partir de là. Cette technique est expliquée plus bas.

### 4.1 Chrooter dans un système existant

Cette section explique comment *chrooter* sur un système Guix déjà installé dans le but de le reconfigurer, par exemple pour corriger une installation défaillante de GRUB. Le processus est semblable à la manière de faire pour les autres systèmes GNU/Linux, mais il y a quelques particularités du système Guix comme le démon et les profils qui le rend utile à décrire ici.

1. Récupérez une image amorçable du système Guix. Nous vous recommandons d'utiliser le dernier instantané de développement pour que le noyau et les outils utilisés soient au moins aussi récents que le système installé ; vous pouvez le récupérer à partir de l'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](https://ci.guix.gnu.org/search/latest/ISO-9660?query=spec:images+status:success+system:x86_64-linux+image.iso)). Suivez la section voir Section 3.3 [Installation depuis une clef USB ou un DVD], page 27, pour apprendre à le copier sur un média amorçable.
2. Démarrez l'image et suivez l'installateur graphique jusqu'à avoir configuré le réseau. Autrement, vous pouvez configurer le réseau manuellement en suivant la section [manual-installation-networking], page 31. Si vous avez une erreur du type 'RTNETLINK answers: Operation not possible due to RF-kill', essayez 'rfkill list' suivi de 'rfkill unblock 0', où '0' est l'identifiant de votre périphérique (ID).
3. Passez sur une console virtuelle (tty) si vous ne l'avez pas déjà fait en appuyant simultanément sur **Control + Alt + F4**. Montez votre système de fichier sur `/mnt`. En supposant que votre partition racine est `/dev/sda2`, vous feriez :

```
mount /dev/sda2 /mnt
```

4. Montez les périphériques blocs spéciaux et les répertoires spécifiques à Linux :

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

Si votre système est basé sur EFI, vous devez aussi monter la partition ESP. En supposant qu'elle est sur `/dev/sda1`, vous pouvez le faire avec :

```
mount /dev/sda1 /mnt/boot/efi
```

5. Entrez dans votre système avec `chroot` :

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

6. Sourcez le profil système ainsi que votre profil *utilisateur* pour mettre en place l'environnement, où *utilisateur* est le nom d'utilisateur utilisé par le système Guix que vous essayez de réparer :

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

Pour vous assurer que vous travaillez avec la révision de Guix que vous utiliseriez en tant qu'utilisateur normal, sourcez aussi votre profil actuel de Guix :

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

7. Démarrez un `guix-daemon` minimal en tâche de fond :

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

8. Modifiez votre configuration système comme vous le souhaitez, puis reconfigurez avec :

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

9. Enfin, vous devriez pouvoir redémarrer le système pour tester votre correction.

## 5 Pour démarrer

Vous êtes certainement arrivé-e à cette section parce que vous avez installé Guix sur une autre distribution (voir Chapitre 2 [Installation], page 5), ou bien vous avez installé Guix System (voir Chapitre 3 [Installation du système], page 26). Il est temps pour vous de commencer à utiliser Guix et cette section est là pour vous aider à le faire et vous donner une idée de ce que c'est.

Guix est, entre autres, un programme d'installation de logiciels, donc la première chose que vous voudrez probablement faire est de chercher un logiciel. Disons que vous cherchez un éditeur de texte, vous pouvez lancer :

```
guix search text editor
```

Cette commande vous montre un certain nombre de *paquets* correspondants, en indiquant à chaque fois le nom du paquet, sa version, une description et des informations supplémentaires. Une fois que vous avez trouvé celui que vous voulez utiliser, disons Emacs (ah ha !), vous pouvez l'installer (lancez cette commande en tant qu'utilisateur-riche, *pas besoin des privilèges d'administration* !) :

```
guix install emacs
```

Vous avez installé votre premier paquet, félicitations ! Le paquet est maintenant visible dans votre *profil* par défaut, `$HOME/.guix-profile` — un profil est un répertoire contenant les paquets installés. Vous avez probablement remarqué que Guix a téléchargé des binaires pré-compilés ; ou bien, si vous vous êtes dit : *non*, pas de binaires pré-compilés, alors Guix est probablement encore en train de construire un logiciel (voir Section 6.3 [Substituts], page 54, pour plus d'informations).

À moins que vous utilisiez Guix System, la commande `guix install` doit avoir affiché cet indice :

conseil : pensez à définir les variables d'environnement nécessaires en exécutant : ■

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

Vous pouvez également consulter ``guix package --search-paths -p "$HOME/.guix-profile"`

En effet, vous devez maintenant indiquer à votre *shell* où se trouvent `emacs` et les autres programmes installés avec Guix. En collant les deux lignes ci-dessus, c'est exactement ce que vous ferez : vous ajouterez `$HOME/.guix-profile/bin` — qui est l'endroit où se trouve le paquet installé — à la variable d'environnement `PATH`. Vous pouvez coller ces deux lignes dans votre shell pour qu'elles prennent effet immédiatement, mais surtout vous devez les ajouter à `~/.bash_profile` (ou un fichier équivalent si vous n'utilisez pas Bash) afin que les variables d'environnement soient définies la prochaine fois que vous lancerez un shell. Vous n'avez besoin de le faire qu'une seule fois et les autres variables d'environnement des chemins de recherche seront traitées de la même manière — par exemple, si vous installez les bibliothèques `python` et `Python`, `GUIX_PYTHONPATH` sera définie.

Vous pouvez continuer à installer des paquets à votre guise. Pour lister les paquets installés, lancez :

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

Pour supprimer un paquet, sans surprise, lancez `guix remove`. Une caractéristique distinctive est la possibilité de faire *revenir en arrière* toute opération que vous avez effectuée — installation, suppression, mise à niveau — en tapant simplement :

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

C'est parce que chaque opération est en fait une *transaction* qui crée une nouvelle *génération*. Les générations et leurs différences entre elles peuvent être affichées en lançant :

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

Vous connaissez maintenant les bases de la gestion des paquets !

**Pour aller plus loin:** Voir Chapitre 6 [Gestion de paquets], page 43, pour en savoir plus sur la gestion des paquets. Vous pouvez aimer la gestion *declarative* des paquets avec `guix package --manifest`, la gestion de *profils* séparés avec `--profil`, la suppression des anciennes générations, le ramasse-miettes et d'autres fonctionnalités astucieuses qui vous seront utiles à mesure que vous vous familiariserez avec Guix. Si vous développez du code, voir Chapitre 8 [Développement], page 83, pour des outils supplémentaires. Et si vous êtes curieux-euse, voir Section 6.1 [Fonctionnalités], page 43, pour jeter un coup d'œil sous le capot.

Une fois que vous avez installé un ensemble de paquets, vous voudrez périodiquement faire une *mise à jour* à la dernière version flambant neuve. Pour cela, vous devez d'abord récupérer la dernière révision de Guix et de sa collection de paquets :

```
guix pull
```

Le résultat final est une nouvelle commande `guix`, sous `~/ .config/guix/current/bin`. A moins que vous ne soyez sous Guix System, la première fois que vous lancez `guix pull`, assurez-vous de suivre le conseil que la commande affiche et, comme nous l'avons vu ci-dessus, collez ces deux lignes dans votre terminal et dans `.bash_profile` :

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

Vous devez aussi informer votre shell de pointer sur ce nouveau `guix` :

```
hash guix
```

A ce stade, vous pilotez un Guix tout neuf. Vous pouvez donc aller de l'avant et mettre effectivement à jour tous les paquets que vous avez installés précédemment :

```
guix upgrade
```

En exécutant cette commande, vous verrez que des binaires sont téléchargés (ou peut-être que certains paquets sont construits), et vous finirez par obtenir les paquets mis à jour. Si l'un de ces paquets n'est pas à votre goût, n'oubliez pas que vous pouvez toujours revenir en arrière !

Vous pouvez afficher la révision exacte de Guix actuellement en cours d'exécution en lançant :

```
guix describe
```

L'information affichée est *tout ce qu'il faut pour reproduire exactement le même Guix*, que ce soit à un moment différent ou sur une machine différente.

**Pour aller plus loin:** Voir Section 6.6 [Invoquer guix pull], page 63, pour plus d'informations. Voir Chapitre 7 [Canaux], page 75, sur la façon de spécifier des *canaux* supplémentaires pour extraire des paquets, sur la façon de répliquer Guix, et plus encore. Vous pouvez également trouver *time-machine* pratique (voir Section 6.7 [Invoquer guix time-machine], page 66).

Si vous avez installé Guix System, une des premières choses que vous voudrez faire est de le mettre à jour. Une fois que vous avez lancé `guix pull` pour obtenir le dernier Guix, vous pouvez mettre à jour le système comme ceci :

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

Ceci terminé, le système exécute les dernières versions de ses logiciels. Lorsque vous redémarrez, vous remarquerez un sous-menu dans le chargeur d'amorçage qui indique « Old system generations » : c'est ce qui vous permet de démarrer *une ancienne génération de votre système*, si la dernière génération est « cassée » ou insatisfaisante. Tout comme pour les paquets, vous pouvez toujours *revenir* à une génération précédente *de l'ensemble du système* :

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

Il y a beaucoup de choses que vous voudrez probablement modifier sur votre système : ajouter de nouveaux comptes utilisateur·rice·s, ajouter de nouveaux services système, modifier la configuration de ces services, etc. La configuration du système est *entièrement* décrite dans le fichier `/etc/config.scm`. Voir Section 12.1 [Utiliser le système de configuration], page 238, pour apprendre à le modifier.

Maintenant, vous en savez assez pour commencer !

**Ressources:** Le reste de ce manuel constitue une référence pour tout ce qui concerne Guix. Voici quelques ressources supplémentaires que vous pourriez trouver utiles :

- Voir *The GNU Guix Cookbook*, pour une liste de recettes de style "how-to" pour une variété d'applications.
- La GNU Guix Reference Card (<https://guix.gnu.org/guix-refcard.pdf>) énumère en deux pages la plupart des commandes et options dont vous aurez besoin.
- Le site web contient des vidéos instructives (<https://guix.gnu.org/en/videos/>) couvrant des sujets tels que l'utilisation quotidienne de Guix, comment obtenir de l'aide et comment devenir un·e contributeur·rice.
- Voir Chapitre 14 [Documentation], page 641, pour savoir comment accéder à la documentation sur votre ordinateur.

Nous espérons que vous apprécierez Guix autant que la communauté a de plaisir à le construire !

## 6 Gestion de paquets

Le but de GNU Guix est de permettre à ses utilisatrices et utilisateurs d’installer, mettre à jour et supprimer facilement des paquets logiciels sans devoir connaître leur procédure de construction ou leurs dépendances. Guix va aussi plus loin que ces fonctionnalités évidentes.

Ce chapitre décrit les principales fonctionnalités de Guix, ainsi que des outils de gestion des paquets qu’il fournit. En plus de l’interface en ligne de commande décrite en dessous de (voir Section 6.2 [Invoquer guix package], page 44), vous pouvez aussi utiliser l’interface Emacs-Guix (voir *Le manuel de référence de emacs-guix*), après avoir installé le paquet `emacs-guix` (lancez la commande `M-x guix-help` pour le démarrer) :

```
guix install emacs-guix
```

### 6.1 Fonctionnalités

Ici, nous supposons que vous avez déjà fait vos premiers pas avec Guix voir Chapitre 5 [Pour démarrer], page 40) et que vous voulez avoir un aperçu de ce qui se passe sous le capot.

Lorsque vous utilisez Guix, chaque paquet arrive dans *dépôt des paquets*, dans son propre répertoire — quelque chose comme `/gnu/store/xxx-paquet-1.2`, où `xxx` est une chaîne en base32.

Plutôt que de se rapporter à ces répertoires, les utilisateur·rice·s ont leur propre *profil* qui pointe vers les paquets qu’ils ou elles veulent vraiment utiliser. Ces profils sont stockés dans le répertoire personnel de chacun·e dans `$HOME/.guix-profile`.

Par exemple, `alice` installe GCC 4.7.2. Il en résulte que `/home/alice/.guix-profile/bin/gcc` pointe vers `/gnu/store/...-gcc-4.7.2/bin/gcc`. Maintenant, sur la même machine, `bob` a déjà installé GCC 4.8.0. Le profil de `bob` continue simplement de pointer vers `/gnu/store/...-gcc-4.8.0/bin/gcc` — c.-à-d. les deux versions de GCC coexistent sur le même système sans aucune interférence.

La commande `guix package` est l’outil central pour gérer les paquets (voir Section 6.2 [Invoquer guix package], page 44). Il opère sur les profils de chaque utilisateur·rice et peut être utilisé avec les *privilèges normaux*.

La commande fournit les opérations évidentes d’installation, de suppression et de mise à jour. Chaque invocation est en fait une *transaction* : soit l’opération demandée réussit, soit rien ne se passe. Ainsi, si le processus `guix package` est terminé pendant la transaction ou si une panne de courant arrive pendant la transaction, le profil reste dans son état précédent et reste utilisable.

En plus, il est possible *d’annuler* toute transaction sur les paquets. Donc si par exemple un mise à jour installe une nouvelle version d’un paquet qui révèle un bogue sérieux, vous pouvez revenir en arrière à l’instance précédente de votre profil que vous saviez bien fonctionner. De même, la configuration globale du système dans Guix est sujette aux mises à jour transactionnelles et aux annulations (voir Section 12.1 [Utiliser le système de configuration], page 238).

Tout paquet du dépôt des paquets peut être *glané*. Guix peut déterminer quels paquets sont toujours référencés par les profils des utilisateur·rice·s et supprimer ceux qui ne sont de tout évidence plus référencés (voir Section 6.5 [Invoquer guix gc], page 60). Les

utilisateur·rice·s peuvent toujours explicitement supprimer les anciennes générations de leur profil pour que les paquets auxquels elles faisaient référence puissent être glanés.

Guix prend une approche *purement fonctionnelle* de la gestion de paquets, telle que décrite dans l'introduction (voir Chapitre 1 [Introduction], page 1). Chaque nom de répertoire de paquet dans `/gnu/store` contient un hash de toutes les entrées qui ont été utilisées pendant la construction de ce paquet — le compilateur, les bibliothèques, les scripts de construction, etc. Cette correspondance directe permet aux utilisateur·rice·s de s'assurer que l'installation d'un paquet donné correspond à l'état actuel de leur distribution. Elle aide aussi à maximiser la *reproductibilité* : grâce aux environnements de construction utilisés, une construction donnée a de fortes chances de donner des fichiers identiques bit-à-bit lorsqu'elle est effectuée sur des machines différentes (voir Section 2.5 [Invoquer guix-daemon], page 17).

Ce fondement permet à Guix de supporter le *déploiement transparent de binaire ou source*. Lorsqu'une binaire pré-construit pour une entrée de `/gnu/store` est disponible depuis une source externe (un *substitut*), Guix le télécharge simplement et le décompresse ; sinon, il construit le paquet depuis les sources localement (voir Section 6.3 [Substituts], page 54). Comme les résultats des constructions sont généralement reproductibles au bit près, si vous n'avez pas besoin de faire confiance aux serveurs qui fournissent les substituts : vous pouvez forcer une construction locale et *défier* les fournisseurs (voir Section 10.12 [Invoquer guix challenge], page 226).

Le contrôle de l'environnement de construction est aussi une fonctionnalité utile pour les développeurs. La commande `guix shell` permet aux développeurs d'un paquet de mettre en place rapidement le bon environnement de développement pour leur paquet, sans avoir à installer manuellement les dépendances du paquet dans leur profil (voir Section 8.1 [Invoquer guix shell], page 83).

La totalité de Guix et des définitions de paquets sont placés sous contrôle de version, et `guix pull` vous permet de « voyager dans le temps » de l'historique de Guix lui-même (voir Section 6.6 [Invoquer guix pull], page 63). Cela est rend possible la réplication d'une instance Guix sur une machine différente ou plus tard, ce qui vous permet de *répliquer des environnements logiciels complets*, tout en garantissant un *suivi de provenance* précis des logiciels.

## 6.2 Invoquer guix package

La commande `guix package` est l'outil qui permet d'installer, mettre à jour et supprimer les paquets ainsi que de revenir à une configuration précédente. Ces opérations fonctionnent sur un *profil* utilisateur—un répertoire avec les paquets installés. Chaque utilisateur a un profile par défaut dans `$HOME/.guix-profile`. La commande n'opère que dans le profil de l'utilisateur·rice et fonctionne avec les privilèges normaux (voir Section 6.1 [Fonctionnalités], page 43). Sa syntaxe est :

```
guix package options
```

*options* spécifie d'abord les opérations à effectuer pendant la transaction. À la fin, une nouvelle génération du profil est créée mais les *générations* précédentes du profil restent disponibles si l'utilisateur·rice souhaite y revenir.

Par exemple, pour supprimer `lua` et installer `guile` et `guile-cairo` en une seule transaction :

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

Parce que c'est pratique, nous fournissons aussi les alias suivants :

- `guix search` est un alias de `guix package -s`,
- `guix install` est un alias de `guix package -i`,
- `guix remove` est un alias de `guix package -r`,
- `guix upgrade` est un alias de `guix package -u`,
- et `guix search` est un alias de `guix package -s`.

Ces alias sont moins expressifs que `guix package` et fournissent moins d'options, donc dans certains cas vous devrez probablement utiliser `guix package` directement.

`guix package` supporte aussi une *approche déclarative* où on spécifie l'ensemble exact des paquets qui doivent être disponibles que l'on passe *via* l'option `--manifest` (voir [profile-manifest], page 48).

Pour chaque utilisateur-*rice*, un lien symbolique vers son profil par défaut est automatiquement créé dans `$HOME/.guix-profile`. Ce lien symbolique pointe toujours vers la génération actuelle du profil par défaut du compte. Ainsi, on peut ajouter `$HOME/.guix-profile/bin` à sa variable d'environnement `PATH` etc.

Si vous n'utilisez pas la distribution système Guix, vous devriez ajouter les lignes suivantes à votre `~/.bash_profile` (voir Section "Bash Startup Files" dans *The GNU Bash Reference Manual*) pour que les shells créés ensuite aient tous les bonnes définitions des variables d'environnement :

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

Dans un environnement multi-utilisateur-*rice*, les profils des utilisateur-*rice*-s sont stockés dans un emplacement enregistré comme *garbage-collector root*, vers lequel `$HOME/.guix-profile` pointe (voir Section 6.5 [Invoquer `guix gc`], page 60). Ce répertoire est normalement `localstatedir/guix/profiles/per-user/user`, où `localstatedir` est la valeur passée à `configure` comme `--localstatedir`, et `user` est le nom d'utilisateur-*rice*. Le répertoire `per-user` est créé au démarrage de `guix-daemon`, et le sous-répertoire `user` est créé par `guix package`.

Les *options* peuvent être les suivante :

```
--install=paquet ...
-i paquet ...
```

Installer les *paquets* spécifiés.

Chaque *paquet* peut spécifier un simple nom de paquet, tel que `guile`, éventuellement suivi d'un arobase et d'un numéro de version, tel que `guile@3.0.7` ou simplement `guile@3.0`. (dans ce dernier cas, la version la plus récente préfixée par 3.0 est sélectionnée).

Si aucun numéro de version n'est spécifié, la version la plus récente disponible est choisie. En plus, une telle spécification de *paquet* peut contenir un deux-points, suivi du nom d'une des sorties du paquet, comme dans `gcc:doc` ou `binutils@2.22:lib` (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59).



Des paquets avec un nom correspondant (et éventuellement une version) sont recherchés dans les modules de la distribution GNU (voir Section 9.1 [Modules de paquets], page 104).

Autrement, un *paquet* peut directement spécifier un fichier du dépôt comme `/gnu/store/...-guile-3.0.7`, produit par exemple par `guix build`.

Parfois les paquets ont des *entrées propagées* : ce sont des dépendances qui sont installées automatiquement avec le paquet demandé (voir [package-propagated-inputs], page 110, pour plus d'informations sur les entrées propagées dans les définitions des paquets).

Un exemple est la bibliothèque MPC de GNU : ses fichiers d'en-tête C se réfèrent à ceux de la bibliothèque MPFR de GNU, qui se réfèrent en retour à ceux de la bibliothèque GMP. Ainsi, lorsqu'on installe MPC, les bibliothèques MPFR et GMP sont aussi installées dans le profil ; supprimer MPC supprimera aussi MPFR et GMP — à moins qu'ils n'aient été aussi installés explicitement par l'utilisateur-riche.

En outre, les paquets s'appuient parfois sur la définition de variables d'environnement pour leurs chemins de recherche (voir l'explication de l'`--chemins de recherche` ci-dessous). Toute définition de variable d'environnement manquante ou éventuellement incorrecte est reportée ici.

`--install-from-expression=exp`

`-e exp` Installer le paquet évalué par *exp*.

*exp* doit être une expression Scheme qui s'évalue en un objet `<package>`. Cette option est notamment utile pour distinguer les variantes d'un paquet avec le même nom, avec des expressions comme `(@ (gnu packages base) guile-final)`.

Remarquez que cette option installe la première sortie du paquet, ce qui peut être insuffisant lorsque vous avez besoin d'une sortie spécifique d'un paquet à plusieurs sorties.

`--install-from-file=fichier`

`-f fichier`

Installer le paquet évalué par le code dans le *fichier*.

Par exemple, *fichier* peut contenir une définition comme celle-ci (voir Section 9.2 [Définition des paquets], page 105) :

```
(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
            "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kz17c9lng89ndqi"))))
      (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+))

```

Lorsqu'on développe, on peut trouver utile d'inclure un tel fichier `guix.scm` à la racine de l'arborescence des sources de son projet qui pourrait être utilisé pour tester des versions de développement et créer des environnements de développement reproductibles (voir Section 8.1 [Invoquer guix shell], page 83).

Le *fichier* peut aussi contenir une représentation JSON d'une ou de plusieurs définitions de paquets. L'exécution de `guix package -f` sur `hello.json` avec le contenu suivant entraînerait l'installation du paquet `greeter` après la construction de `myhello` :

```

[
  {
    "name": "myhello",
    "version": "2.10",
    "source": "mirror://gnu/hello/hello-2.10.tar.gz",
    "build-system": "gnu",
    "arguments": {
      "tests?": false
    }
  },
  {
    "home-page": "https://www.gnu.org/software/hello/",
    "synopsis": "Hello, GNU world: An example GNU package",
    "description": "GNU Hello prints a greeting.",
    "license": "GPL-3.0+",
    "native-inputs": ["gettext"]
  },
  {
    "name": "greeter",
    "version": "1.0",
    "source": "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=paquet ...``-r paquet ...`

Supprimer les *paquets* spécifiés.

Comme pour `--install`, chaque *paquet* peut spécifier un numéro de version ou un nom de sortie en plus du nom du paquet. Par exemple, `'-r glibc:debug'` supprimerait la sortie `debug` de `glibc`.

`--upgrade[=regex ...]``-u [regex ...]`

Mettre à jour tous les paquets installés. Si un ou plusieurs *regex* sont spécifiés, la mise à jour n'installera que les paquets dont le nom correspond à *regex*. Voir aussi l'option `--do-not-upgrade` ci-dessous.

Remarquez que cela met à jour vers la dernière version des paquets trouvée dans la distribution actuellement installée. Pour mettre à jour votre distribution, vous devriez lancer régulièrement `guix pull` (voir Section 6.6 [Invoquer `guix pull`], page 63).

Quand une mise à jour est en cours d'exécution, les transformations de paquets initialement appliquées lors de la création du profil sont automatiquement réappliquées (voir Section 10.1.2 [Options de transformation de paquets], page 183). Par exemple, supposons que vous avez d'abord installé Emacs depuis l'extrémité de sa branche de développement avec :

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

La prochaine fois que vous lancerez `guix upgrade`, Guix va à nouveau extraire l'extrémité de la branche de développement Emacs et construira `emacs-next` à partir de ce checkout.

Notez que les options de transformation comme `--with-branch` et `--with-source` dépendent de l'état extérieur ; c'est à vous de vous assurer qu'elles fonctionnent comme souhaité. Vous pouvez également écarter une transformation qui s'applique à un paquet en lançant :

```
$ guix install paquet
```

`--do-not-upgrade[=regex ...]`

Lors d'une utilisation conjointe avec l'option `--upgrade`, ne *pas* mettre à jour les paquets dont le nom correspond à *regex*. Par exemple, pour mettre à jour tous les paquets du profil actuel à l'exception de ceux qui contiennent la chaîne « emacs » :

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

`--manifest=fichier``-m fichier`

Créer une nouvelle génération du profil depuis l'objet manifeste renvoyé par le code Scheme dans *fichier*. Cette option peut être répétée plusieurs fois, auquel cas les manifestes sont concaténés.

Cela vous permet de *déclarer* le contenu du profil plutôt que de le construire avec une série de `--install` et de commandes similaires. L'avantage étant que

le *fichier* peut être placé sous contrôle de version, copié vers d'autres machines pour reproduire le même profil, etc.

*fichier* doit renvoyer un objet *manifest* qui est en gros une liste de paquets :

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

(packages->manifest
 (list emacs
        guile-2.0
        ;; Utiliser une sortie spécifique d'un paquet.
        (list guile-2.0 "debug")))
```

Voir Section 9.4 [Écrire un manifeste], page 120, pour des informations sur la manière d'écrire un manifeste. Voir [export-manifest], page 53, pour apprendre à obtenir un fichier manifeste depuis un profil existant.

#### `--roll-back`

Revenir à la *génération* précédente du profil c.-à-d. défaire la dernière transaction.

Lorsqu'elle est combinée avec des options comme `--install`, Le retour en arrière se produit avant toute autre action.

Lorsque vous revenez de la première génération qui contient des fichiers, le profil pointera vers la *zéroième génération* qui ne contient aucun fichier en dehors de ses propres métadonnées.

Après être revenu en arrière, l'installation, la suppression et la mise à jour de paquets réécrit les futures générations précédentes. Ainsi, l'historique des générations dans un profil est toujours linéaire.

#### `--switch-generation=motif`

`-S motif` Basculer vers une génération particulière définie par le *motif*.

Le *motif* peut être soit un numéro de génération soit un nombre précédé de « + » ou « - ». Ce dernier signifie : se déplacer en avant ou en arrière d'un nombre donné de générations. Par exemple, si vous voulez retourner à la dernière génération après `--roll-back`, utilisez `--switch-generation=+1`.

La différence entre `--roll-back` et `--switch-generation=-1` est que `--switch-generation` ne vous amènera pas à la génération initiale, donc si la génération demandée n'existe pas la génération actuelle ne changera pas.

#### `--search-paths [=genre]`

Rapporter les définitions des variables d'environnement dans la syntaxe Bash qui peuvent être requises pour utiliser l'ensemble des paquets installés. Ces variables d'environnement sont utilisées pour spécifier les *chemins de recherche* de fichiers utilisés par les paquets installés.

Par exemple, GCC a besoin des variables d'environnement `CPATH` et `LIBRARY_PATH` pour trouver les en-têtes et les bibliothèques dans le profil de l'utilisateur-riche (voir Section "Environment Variables" dans *Using the GNU Compiler Collection (GCC)*). Si GCC et, disons, la bibliothèque C sont installés dans le profil, alors `--search-paths` suggérera d'initialiser ces variables à `profil/include` et `profil/lib`, respectivement (voir Section 9.8

[Chemins de recherche], page 154, pour des infos sur les spécifications de chemins de recherche associées aux paquets).

Le cas d'utilisation typique est de définir ces variables d'environnement dans le shell :

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

*genre* peut être l'une des valeurs **exact**, **prefix** ou **suffix**, ce qui signifie que les définitions des variables d'environnement renvoyées seront soit les paramètres exacts, soit placés avant ou après la valeur actuelle de ces paramètres. Lorsqu'il est omis, *genre* a pour valeur par défaut **exact**.

Cette option peut aussi être utilisé pour calculer les chemins de recherche *combinés* de plusieurs profils. Regardez cet exemple :

```
$ guix package -p toto -i guile
$ guix package -p titi -i guile-json
$ guix package -p toto -p titi --search-paths
```

La dernière commande ci-dessus montre la variable `GUILE_LOAD_PATH` bien que, pris individuellement, ni `toto` ni `titi` n'auraient donné cette recommandation.

**--profile=profil**

**-p profil** Utiliser le *profil* à la place du profil par défaut du compte.

*profil* doit être le nom d'un fichier qui sera créé une fois terminé. Concrètement, *profil* sera un simple lien symbolique (« symlink ») pointant vers le profil réel où les paquets sont installés :

```
$ guix install hello -p ~/code/my-profile
...
$ ~/code/my-profile/bin/hello
Hello, world!
```

Tout ce qu'il faut pour se débarrasser du profil est de supprimer ce lien symbolique et ses frères et sœurs qui pointent vers des générations spécifiques :

```
$ rm ~/code/my-profile ~/code/my-profile-*-link
```

**--list-profiles**

Liste tous les profils utilisateur·rice :

```
$ guix package --list-profiles
/home/charlie/.guix-profile
/home/charlie/code/my-profile
/home/charlie/code/devel-profile
/home/charlie/tmp/test
```

En cours d'exécution sous `root`, liste tous les profils de tou·te·s les utilisateur·rice·s.

**--allow-collisions**

Permettre des collisions de paquets dans le nouveau profil. À utiliser à vos risques et périls !

Par défaut, `guix package` rapporte les *collisions* dans le profil comme des erreurs. Les collisions ont lieu quand deux version ou variantes d'un paquet donné se retrouvent dans le profil.

**--bootstrap**

Utiliser le programme d'amorçage Guile pour compiler le profil. Cette option n'est utile qu'aux personnes qui développent la distribution.

En plus de ces actions, `guix package` supporte les options suivantes pour demander l'état actuel d'un profil ou la disponibilité des paquets :

**--search=regex**

**-s regex** Lister les paquets disponibles dont le nom, le synopsis ou la description correspondent à la *regex* (en étant insensible à la casse), triés par pertinence. Afficher toutes les métadonnées des paquets correspondants au format `recutils` (voir *GNU recutils manual*).

Cela permet à des champs spécifiques d'être extraits avec la commande `recsel`, par exemple :

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

name: glibc
version: 2.25
relevance: 1

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

De manière similaire, pour montrer le nom de tous les paquets disponibles sous licence GNU LGPL version 3 :

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

name: gmp
...
```

Il est aussi possible de raffiner les résultats de la recherche en donnant plusieurs options `-s` à `guix package`, ou plusieurs arguments à `guix search`. Par exemple, la commande suivante renvoie la liste des jeux de plateau (cette fois-ci avec l'alias `guix search`) :

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

Si on avait oublié `-s game`, on aurait aussi eu les paquets logiciels qui s'occupent de circuits imprimés (en anglais : circuit board) ; supprimer les chevrons autour de `board` aurait aussi ajouté les paquets qui parlent de clavier (en anglais : *keyboard*).

Et maintenant un exemple plus élaboré. La commande suivante recherche les bibliothèques cryptographiques, retire les bibliothèques Haskell, Perl, Python et Ruby et affiche le nom et le synopsis des paquets correspondants :

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

Voir Section “Selection Expressions” dans *GNU recutils manual* pour plus d’information sur les *expressions de sélection* pour `recsel -e`.

`--show=paquet`

Afficher les détails du *paquet* dans la liste des paquets disponibles, au format `recutils` (voir *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
...
```

Vous pouvez aussi spécifier le nom complet d’un paquet pour n’avoir que les détails concernant une version spécifique (cette fois-ci avec l’alias `guix show`) :

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

`--list-installed[=regex]`

`-I [regex]`

Liste les paquets actuellement installés dans le profil spécifié, avec les paquets les plus récemment installés en dernier. Lorsque *regex* est spécifié, liste uniquement les paquets installés dont le nom correspond à *regex*.

Pour chaque paquet installé, affiche les éléments suivants, séparés par des tabulations : le nom du paquet, sa version, la partie du paquet qui est installé (par exemple, `out` pour la sortie par défaut, `include` pour ses en-têtes, etc) et le chemin du paquet dans le dépôt.

`--list-available[=regex]`

`-A [regex]`

Lister les paquets actuellement disponibles dans la distribution pour ce système (voir Section 1.2 [Distribution GNU], page 2). Lorsque *regex* est spécifié, lister uniquement les paquets dont le nom correspond à *regex*.

Pour chaque paquet, affiche les éléments suivants séparés par des tabulations : son nom, sa version, les parties du paquet (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59), et l’emplacement de sa définition.

`--list-generations[=motif]`

`-l [motif]`

Renvoyer la liste des générations avec leur date de création ; pour chaque génération, montre les paquets installés avec les paquets installés les plus

récemment en dernier. Remarquez que la zéroième génération n'est jamais montrée.

Pour chaque paquet installé, afficher les éléments suivants, séparés par des tabulations : le nom du paquet, sa version, la partie du paquet qui a été installée (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59), et l'emplacement du paquet dans le dépôt.

Lorsque *motif* est utilisé, la commande ne renvoie que les générations correspondantes. Les motifs valides sont :

- *Des entiers et des entiers séparés par des virgules.* Les deux motifs correspondent numéros de génération. Par exemple, `--list-generations=1` renvoie la première.

Et `--list-generations=1,8,2` renvoie les trois générations dans l'ordre spécifié. Aucun espace ni virgule surnuméraire ne sont permis.

- *Ranges.* `--list-generations=2..9` affiche les générations demandées et tout ce qui se trouve entre elles. Remarquez que le début d'un intervalle doit être plus petit que sa fin.

Il est aussi possible d'omettre le numéro final. Par exemple, `--list-generations=2..` renvoie toutes les générations à partir de la deuxième.

- *Des durées.* Vous pouvez aussi récupérer les derniers *N* jours, semaines, ou mois en passant un entier avec la première lettre de la durée (en anglais : d, w ou m). Par exemple `--list-generations=20d` liste les générations qui sont âgées d'au plus 20 jours.

`--delete-generations[=motif]`

`-d [motif]`

Lorsque *motif* est omis, supprimer toutes les générations en dehors de l'actuelle.

Cette commande accepte les mêmes motifs que `--list-generations`. Lorsque *motif* est spécifié, supprime les générations correspondantes. Lorsque *motif* spécifie une durée, les générations *plus anciennes* que la durée spécifiée correspondent. Par exemple `--delete-generations=1m` supprime les générations vieilles de plus d'un mois.

Si la génération actuelle correspond, elle n'est *pas* supprimée. La zéroième génération n'est elle non plus jamais supprimée.

Remarquez que supprimer des générations empêche de revenir en arrière vers elles. Ainsi, cette commande doit être utilisée avec précaution.

`--export-manifest`

Écrire un manifeste vers la sortie standard utilisable avec `--manifest` et qui correspond à un ou plusieurs profils choisis.

Cette option est conçue pour vous aider à migrer du mode de fonctionnement « impératif » — lancer `guix install`, `guix upgrade`, etc — au mode déclaratif offert par `--manifest`.

Soyez conscient que le manifeste qui est résulte est une *approximation* de ce que votre profil contient vraiment ; par exemple, en fonction de la manière dont



votre profil a été créé, il peut se référer à des paquets ou des versions de paquets qui ne sont pas exactement ce que vous avez spécifié.

Gardez en tête qu'un manifeste est purement symbolique : il ne contient que des noms de paquets et éventuellement des versions, et leur signification varie avec le temps. Si vous voulez « épinglez » des canaux aux révisions qui ont été utilisées pour construire les profils, voir `--export-channels` plus bas.

#### `--export-channels`

Écrit sur la sortie standard la liste des canaux utilisés par les profils choisis, dans un format convenable pour `guix pull --channels` ou `guix time-machine --channels` (voir Chapitre 7 [Canaux], page 75).

Avec `--export-manifest`, cette option fournit les informations qui vous permettent de répliquer le profil actuel (voir Section 7.3 [Répliquer Guix], page 76).

Cependant, remarquez que la sortie de cette commande est une *approximation* de ce qui a vraiment été utilisé pour construire le profil. En particulier, un profil unique peut avoir été construit à partir de plusieurs révisions du même canal. Dans ce cas, `--export-manifest` choisit la dernière et écrit la liste des autres révisions dans un commentaire. Si vous avez vraiment besoin de choisir des paquets à partir d'autres révisions des canaux, vous pouvez utiliser des inférieurs dans votre manifeste pour cela (voir Section 6.8 [Inférieurs], page 68).

Avec `--export-manifest`, c'est un bon point de départ si vous voulez migrer du modèle « impératif » au modèle complètement déclaratif qui consiste en un fichier manifeste et un fichier de canaux qui épingle les révisions exactes des canaux que vous voulez.

Enfin, comme `guix package` peut démarrer des processus de construction, il supporte les options de construction communes (voir Section 10.1.1 [Options de construction communes], page 180). Il prend aussi en charge les options de transformation de paquets comme `--with-source`, et les préserve à travers les mises à jour (voir Section 10.1.2 [Options de transformation de paquets], page 183).

## 6.3 Substituts

Guix gère le déploiement depuis des binaires ou des sources de manière transparente ce qui signifie qu'il peut aussi bien construire localement que télécharger des éléments pré-construits depuis un serveur ou les deux. Nous appelons ces éléments pré-construits des *substituts* — ils se substituent aux résultats des constructions locales. Dans la plupart des cas, télécharger un substitut est bien plus rapide que de construire les choses localement.

Les substituts peuvent être tout ce qui résulte d'une construction de dérivation (voir Section 9.10 [Dérivations], page 159). Bien sûr dans le cas général, il s'agit de paquets binaires pré-construits, mais les archives des sources par exemple résultent aussi de la construction d'une dérivation qui peut aussi être disponible en tant que substitut.

### 6.3.1 Serveur de substituts officiel

`ci.guix.gnu.org` et `bordeaux.guix.gnu.org` sont tous deux des interfaces aux fermes de construction officielles qui construisent des paquets pour Guix continuellement pour certaines architectures et les rend disponibles en tant que substituts. Ce sont les sources

par défaut des substituts qui peuvent être modifiées en passant l'option `--substitute-urls` soit à `guix-daemon` (voir [guix-daemon --substitute-urls], page 18) soit aux outils clients comme `guix package` (voir [client --substitute-urls option], page 181).

Les URL des substituts peuvent être soit en HTTP soit en HTTPS. Le HTTPS est recommandé parce que les communications sont chiffrées ; à l'inverse HTTP rend les communications visibles pour un espion qui peut utiliser les informations accumulées sur vous pour déterminer par exemple si votre système a des vulnérabilités de sécurité non corrigées.

Les substituts des fermes de construction officielles sont activés par défaut dans la distribution système Guix (voir Section 1.2 [Distribution GNU], page 2). Cependant, ils sont désactivés par défaut lorsque vous utilisez Guix sur une distribution externe, à moins que vous ne les ayez explicitement activés via l'une des étapes d'installation recommandées (voir Chapitre 2 [Installation], page 5). Les paragraphes suivants décrivent comment activer ou désactiver les substituts pour la ferme de construction officielle ; la même procédure peut être utilisée pour activer les substituts de n'importe quel autre serveur de substituts.

### 6.3.2 Autoriser un serveur de substituts

Pour permettre à Guix de télécharger les substituts depuis `ci.guix.gnu.org`, `bordeaux.guix.gnu.org` ou un miroir, vous devez ajouter la clé publique correspondante à la liste de contrôle d'accès (ACL) des imports d'archives, avec la commande `guix archive` (voir Section 6.10 [Invoquer guix archive], page 71). Cela implique que vous faites confiance au serveur de substituts pour ne pas être compromis et vous servir des substituts authentiques.

**Remarque:** Si vous utilisez le système Guix, vous pouvez sauter cette section : le système Guix autorise les substituts de `ci.guix.gnu.org` et `bordeaux.guix.gnu.org` par défaut.

Les clés publiques de chaque serveur de substituts maintenu par le projet sont installées avec Guix, dans `préfixe/share/guix/`, où `préfixe` est le préfixe d'installation de Guix. Si vous avez installé Guix depuis les sources, assurez-vous d'avoir vérifié la signature GPG de `guix-1.4.0.tar.gz` qui contient ce fichier de clé publique. Ensuite vous pouvez lancer quelque chose comme ceci :

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

Une fois que cela est en place, la sortie d'une commande comme `guix build` devrait changer de quelque chose comme :

```
$ guix build emacs --dry-run
```

Les dérivations suivantes seraient construites :

```
/gnu/store/yr7bnx8xwcayd6j95r2clmkdl1qh688w-emacs-24.3.drv
/gnu/store/x8qsh1hlhgjx6cwsjyvbnfv2i37z23w-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
```

...

à quelque chose comme :

```
$ guix build emacs --dry-run
```

112.3 Mo seraient téléchargés :

```
/gnu/store/pk3n22lbq6ydamyymqkz7i69wiwjiwi-emacs-24.3
```



Cela suppose que le fichier `key.pub` contient la clé de signature de `guix.example.org`. Avec ce changement dans le fichier de configuration de votre système d'exploitation (disons `/etc/config.scm`), vous pouvez reconfigurer et redémarrer le service `guix-daemon` ou redémarrer pour que les changements prennent effet :

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

Si vous utilisez Guix sur une « distro externe », vous suivrez plutôt les étapes suivantes pour avoir des substituts de serveurs supplémentaires :

1. Modifier le fichier de configuration du service `guix-daemon` ; pour `systemd`, c'est normalement `/etc/systemd/system/guix-daemon.service`. Ajouter l'option `--substitute-urls` à la ligne de command `guix-daemon` et listez les URL qui vous intéressent (voir [daemon-substitute-urls], page 18) :

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

2. Redémarrez le démon. Pour `systemd`, cela ressemble à :

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

3. Autorisez la clé du nouveau serveur (voir Section 6.10 [Invoquer guix archive], page 71) :

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

De nouveau, cela suppose que `key.pub` contient la clé publique que `guix.example.org` utilise pour signer les substituts.

Maintenant vous êtes paré ! Les substituts seront téléchargés de préférence à partir de `https://guix.example.org`, avec `ci.guix.gnu.org` puis `bordeaux.guix.gnu.org` en réserve. Évidemment vous pouvez lister autant de serveurs de substituts que vous voulez, avec l'inconvénient que la recherche de substituts sera ralentie si vous devez contacter trop de serveurs.

Remarquez qu'il y a des situations où vous pourriez vouloir ajouter l'URL d'un serveur de substitut *sans* autoriser sa clé. Voir Section 6.3.4 [Authentification des substituts], page 57, pour comprendre ce point délicat.

### 6.3.4 Authentification des substituts

Guix détecte et lève une erreur lorsqu'il essaye d'utiliser un substitut qui a été modifié. De même, il ignore les substituts qui ne sont pas signés ou qui ne sont pas signés par l'une des clefs listées dans l'ACL.

Il y a une exception cependant : si un serveur non autorisé fournit des substituts qui sont *identiques bit-à-bit* à ceux fournis par un serveur autorisé, alors le serveur non autorisé devient disponible pour les téléchargements. Par exemple en supposant qu'on a choisi deux serveurs de substituts avec cette option :

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

Si l'ACL contient uniquement la clef de `'b.example.org'`, et si `'a.example.org'` sert *exactement les mêmes* substituts, alors Guix téléchargera les substituts de `'a.example.org'` parce qu'il vient en premier dans la liste et peut être considéré comme un miroir de

'`b.example.org`'. En pratique, les machines de construction indépendantes produisent généralement les mêmes binaires, grâce à des constructions reproductibles au bit près (voir ci-dessous).

Lorsque vous utilisez HTTPS, le certificat X.509 du serveur n'est *pas* validé (en d'autres termes, le serveur n'est pas authentifié), contrairement à ce que des clients HTTPS comme des navigateurs web font habituellement. Cela est dû au fait que Guix authentifie les informations sur les substituts eux-mêmes, comme expliqué plus haut, ce dont on se soucie réellement (alors que les certificats X.509 authentifient la relation entre nom de domaine et clef publique).

### 6.3.5 Paramètres de serveur mandataire

Les substituts sont téléchargés par HTTP ou HTTPS. Les variables d'environnement `http_proxy` et `https_proxy` peuvent être initialisées dans l'environnement de `guix-daemon` et sont prises en compte pour le téléchargement des substituts. Remarquez que la valeur de ces variables d'environnement dans l'environnement où sont exécutés `guix build`, `guix package`, et d'autres commandes client n'a *absolument aucun effet*.

### 6.3.6 Échec de substitution

Même lorsqu'un substitut pour une dérivation est disponible, la substitution échoue parfois. Cela peut arriver pour plusieurs raisons : le serveur de substitut peut être hors ligne, le substitut a récemment été supprimé du serveur, la connexion peut avoir été interrompue, etc.

Lorsque les substituts sont activés et qu'un substitut pour une dérivation est disponible, mais que la tentative de substitution échoue, Guix essaiera de construire la dérivation localement si `--fallback` a été passé en argument (voir [common build option `--fallback`], page 181). Plus spécifiquement, si cet option n'a pas été passée en argument, alors aucune construction locale n'est effectuée et la dérivation est considérée comme étant en échec. Cependant, si `--fallback` est passé en argument, alors Guix essaiera de construire la dérivation localement et l'échec ou le succès de la dérivation dépend de l'échec ou du succès de la construction locale. Remarquez que lorsque les substituts sont désactivés ou qu'aucun substitut n'est disponible pour la dérivation en question, une construction locale sera *toujours* effectuée, indépendamment du fait que l'argument `--fallback` ait été ou non passé.

Pour se donner une idée du nombre de substituts disponibles maintenant, vous pouvez essayer de lancer la commande `guix weather` (voir Section 10.15 [Invoquer `guix weather`], page 231). Cette commande fournit des statistiques sur les substituts fournis par un serveur.

### 6.3.7 De la confiance en des binaires

De nos jours, le contrôle individuel sur son utilisation propre de l'informatique est à la merci d'institutions, de sociétés et de groupes avec assez de pouvoir et de détermination pour contourner les infrastructures informatiques et exploiter leurs faiblesses. Bien qu'utiliser les substituts soit pratique, nous encourageons chacun·e à construire aussi par soi-même, voire à faire tourner sa propre ferme de construction, pour que les serveurs de substituts du projet deviennent une cible moins intéressante. Une façon d'aider est de publier les logiciels que vous construisez avec `guix publish` pour que les autres aient plus de choix de serveurs où télécharger les substituts (voir Section 10.11 [Invoquer `guix publish`], page 222).

Guix possède les fondations pour maximiser la reproductibilité logicielle (voir Section 6.1 [Fonctionnalités], page 43). Dans la plupart des cas, des constructions indépendantes d'un paquet donnée ou d'une dérivation devrait donner des résultats identiques au bit près. Ainsi, à travers un ensemble de constructions de paquets indépendantes il est possible de renforcer l'intégrité du système. La commande `guix challenge` a pour but d'aider les utilisateur·rices à tester les serveurs de substituts et à aider les développeur·euses à trouver les constructions de paquets non-déterministes (voir Section 10.12 [Invoquer `guix challenge`], page 226). De même, l'option `--check` de `guix build` permet à chaque personne de vérifier si les substituts précédemment installés sont authentiques en les reconstruisant localement (voir [vérification de la construction], page 192).

À l'avenir, nous aimerions que Guix puisse publier et recevoir des binaires d'autres personnes, de manière pair-à-pair. Si vous voulez discuter de ce projet, rejoignez-nous sur `guix-devel@gnu.org`.

## 6.4 Des paquets avec plusieurs résultats

Souvent, les paquets définis dans Guix ont une seule *sortie* — c.-à-d. que le paquet source conduit à exactement un répertoire dans le dépôt. Lorsque vous lancez `guix install glibc`, vous installez la sortie par défaut du paquet GNU lib c ; la sortie par défaut est appelée `out` mais son nom peut être omis comme le montre cette commande. Dans ce cas particulier, la sortie par défaut de `glibc` contient tous les fichiers d'en-tête C, les bibliothèques partagées, les bibliothèques statiques, la documentation Info et les autres fichiers de support.

Parfois il est plus approprié de séparer les divers types de fichiers produits par un même paquet source en plusieurs sorties. Par exemple, la bibliothèque C GLib (utilisée par GTK+ et des paquets associés) installe plus de 20 Mo de documentation de référence dans des pages HTML. Pour préserver l'espace disque des utilisateurs qui n'en ont pas besoin, la documentation va dans une sortie séparée nommée `doc`. Pour installer la sortie principale de GLib, qui contient tout sauf la documentation, on devrait lancer :

```
guix install glib
```

La commande pour installer la documentation est :

```
guix install glib:doc
```

Certains paquets installent des programmes avec des « empreintes dépendances » différentes. Par exemple le paquet WordNet installe à la fois les outils en ligne de commande et les interfaces graphiques (GUI). La première ne dépend que de la bibliothèque C, alors que cette dernière dépend de Tcl/Tk et des bibliothèques X sous-jacentes. Dans ce cas, nous laissons les outils en ligne de commande dans la sortie par défaut et l'interface graphique dans une sortie séparée. Cela permet aux utilisateurs qui n'ont pas besoin d'interface graphique de gagner de la place. La commande `guix size` peut aider à trouver ces situations (voir Section 10.9 [Invoquer `guix size`], page 215). `guix graph` peut aussi être utile (voir Section 10.10 [Invoquer `guix graph`], page 218).

Il y a plusieurs paquets à sorties multiples dans la distribution GNU. D'autres noms de sorties conventionnels sont `lib` pour les bibliothèques et éventuellement les fichiers d'en-tête, `bin` pour les programmes indépendants et `debug` pour les informations de débogage (voir Chapitre 17 [Installer les fichiers de débogage], page 652). Les sorties d'un paquet sont listés dans la troisième colonne de la sortie de `guix package --list-available` (voir Section 6.2 [Invoquer `guix package`], page 44).

## 6.5 Invoquer `guix gc`

Les paquets qui sont installés mais pas utilisés peuvent être *glanés*. La commande `guix gc` permet aux utilisateurs de lancer explicitement le ramasse-miettes pour récupérer de l'espace dans le répertoire `/gnu/store`. C'est la *seule* manière de supprimer des fichiers de `/gnu/store` — supprimer des fichiers ou des répertoires à la main peut le casser de manière impossible à réparer !

Le ramasse-miettes a un ensemble de *racines* connues : tout fichier dans `/gnu/store` atteignable depuis une racine est considéré comme *utilisé* et ne peut pas être supprimé ; tous les autres fichiers sont considérés comme *inutilisés* et peuvent être supprimés. L'ensemble des racines du ramasse-miettes (ou « racines du GC » pour faire court) inclue les profils par défaut des utilisateurs ; par défaut les liens symboliques sous `/var/guix/gcroots` représentent ces racines du GC. De nouvelles racines du GC peuvent être ajoutées avec la commande `guix build --root` par exemple (voir Section 10.1 [Invoquer `guix build`], page 180). La commande `guix gc --list-roots` permet de les lister.

Avant de lancer `guix gc --collect-garbage` pour faire de la place, c'est souvent utile de supprimer les anciennes génération des profils utilisateurs ; de cette façon les anciennes constructions de paquets référencées par ces générations peuvent être glanées. Cela se fait en lançant `guix package --delete-generations` (voir Section 6.2 [Invoquer `guix package`], page 44).

Nous recommandons de lancer le ramasse-miettes régulièrement ou lorsque vous avez besoin d'espace disque. Par exemple pour garantir qu'au moins 5 Go d'espace reste libre sur votre disque, lancez simplement :

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

Il est parfaitement possible de le lancer comme une tâche périodique non-interactive (voir Section 12.9.2 [Exécution de tâches planifiées], page 287, pour apprendre comment paramétrer une telle tâche). Lancer `guix gc` sans argument ramassera autant de miettes que possible mais ça n'est pas le plus pratique : vous pourriez vous retrouver à reconstruire ou re-télécharger des logiciels « inutilisés » du point de vu du GC mais qui sont nécessaires pour construire d'autres logiciels — p. ex. la chaîne de compilation.

La commande `guix gc` a trois modes d'opération : elle peut être utilisée pour glaner des fichiers inutilisés (par défaut), pour supprimer des fichiers spécifiques (l'option `--delete`), pour afficher des informations sur le ramasse-miettes ou pour des requêtes plus avancées. Les options du ramasse-miettes sont :

```
--collect-garbage[=min]
```

```
-C [min] Ramasse les miettes — c.-à-d. les fichiers inaccessibles de /gnu/store et ses sous-répertoires. C'est l'opération par défaut lorsqu'aucune option n'est spécifiée.
```

Lorsque *min* est donné, s'arrêter une fois que *min* octets ont été collectés. *min* peut être un nombre d'octets ou inclure un suffixe d'unité, comme MiB pour mébioctet et GB pour gigaoctet (voir Section “Block size” dans *GNU Coreutils*).

Lorsque *min* est omis, tout glaner.

```
--free-space=libre
```

```
-F libre Glaner jusqu'à ce que libre espace soit disponible dans /gnu/store si possible ; libre est une quantité de stockage comme 500MiB comme décrit ci-dessus.
```

Lorsque *libre* ou plus est disponible dans `/gnu/store` ne rien faire et s'arrêter immédiatement.

`--delete-generations[=durée]`  
`-d [durée]`

Avant de commencer le glanage, supprimer toutes les générations plus vieilles que *durée*, pour tous les profils utilisateurs et générations de l'environnement personnel ; lorsque cela est lancé en root, cela s'applique à tous les profils *de tous les utilisateurs*.

Par exemple, cette commande supprime toutes les générations de tous vos profils plus vieilles que 2 mois (sauf s'il s'agit de la génération actuelle) puis libère de l'espace jusqu'à atteindre au moins 10 Go d'espace libre :

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

`--delete`

`-D` Essayer de supprimer tous les fichiers et les répertoires du dépôt spécifiés en argument. Cela échoue si certains des fichiers ne sont pas dans le dépôt ou s'ils sont toujours utilisés.

`--list-failures`

Lister les éléments du dépôt qui correspondent à des échecs de construction.

Cela n'affiche rien à moins que le démon n'ait été démarré avec `--cache-failures` (voir Section 2.5 [Invoquer `guix-daemon`], page 17).

`--list-roots`

Lister les racines du GC appartenant à l'utilisateur ; lorsque la commande est lancée en root, lister *toutes* les racines du GC.

`--list-busy`

Liste des éléments de stockage utilisés par les processus en cours d'exécution. Ces éléments de stockage sont effectivement considérés comme des racines GC : ils ne peuvent pas être supprimés.

`--clear-failures`

Supprimer les éléments du dépôt spécifiés du cache des constructions échouées.

De nouveau, cette option ne fait de sens que lorsque le démon est démarré avec `--cache-failures`. Autrement elle ne fait rien.

`--list-dead`

Montrer la liste des fichiers et des répertoires inutilisés encore présents dans le dépôt — c.-à-d. les fichiers et les répertoires qui ne sont plus atteignables par aucune racine.

`--list-live`

Montrer la liste des fichiers et des répertoires du dépôt utilisés.

En plus, les références entre les fichiers existants du dépôt peuvent être demandés :

`--references`

`--referrers`

Lister les références (respectivement les référents) des fichiers du dépôt en argument.



**--requisites**

**-R** Lister les prérequis des fichiers du dépôt passés en argument. Les prérequis sont le fichier du dépôt lui-même, leur références et les références de ces références, récursivement. En d'autres termes, la liste retournée est la *closure transitive* des fichiers du dépôt.

Voir Section 10.9 [Invoquer guix size], page 215, pour un outil pour surveiller la taille de la closure d'un élément. Voir Section 10.10 [Invoquer guix graph], page 218, pour un outil pour visualiser le graphe des références.

**--derivivers**

Renvoie les dérivations menant aux éléments du dépôt donnés (voir Section 9.10 [Dérivations], page 159).

Par exemple cette commande :

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

renvoie les fichiers `.drv` menant au paquet `emacs` installé dans votre profil.

Remarquez qu'il peut n'y avoir aucun fichier `.drv` par exemple quand ces fichiers ont été glanés. Il peut aussi y avoir plus d'un fichier `.drv` correspondant à cause de dérivations à sortie fixées.

Enfin, les options suivantes vous permettent de vérifier l'intégrité du dépôt et de contrôler l'utilisation du disque.

**--verify[=*options*]**

Vérifier l'intégrité du dépôt.

Par défaut, s'assurer que tous les éléments du dépôt marqués comme valides dans la base de données du démon existent bien dans `/gnu/store`.

Lorsqu'elle est fournie, l'*option* doit être une liste séparée par des virgule de l'un ou plus parmi `contents` et `repair`.

Lorsque vous passez `--verify=contents`, le démon calcul le hash du contenu de chaque élément du dépôt et le compare au hash de sa base de données. Les différences de hash sont rapportées comme des corruptions de données. Comme elle traverse *tous les fichiers du dépôt*, cette commande peut prendre très longtemps pour terminer, surtout sur un système avec un disque lent.

Utiliser `--verify=repair` ou `--verify=contents,repair` fait que le démon essaie de réparer les objets du dépôt corrompus en récupérant leurs substituts (voir Section 6.3 [Substituts], page 54). Comme la réparation n'est pas atomique et donc potentiellement dangereuse, elle n'est disponible que pour l'administrateur système. Une alternative plus légère lorsque vous connaissez exactement quelle entrée est corrompue consiste à lancer `guix build --repair` (voir Section 10.1 [Invoquer guix build], page 180).

**--optimize**

Optimiser le dépôt en liant en dur les fichiers identiques — c'est la *déduplication*.

Le démon effectue une déduplication après chaque import d'archive ou construction réussie, à moins qu'il n'ait été lancé avec `--disable-deduplication` (voir Section 2.5 [Invoquer guix-daemon], page 17). Ainsi, cette option est surtout utile lorsque le démon tourne avec `--disable-deduplication`.

**--vacuum-database**

Guix utilise une base de données `sqlite` pour garder la trace des éléments du dépôt (voir Section 9.9 [Le dépôt], page 157). Avec le temps il est possible que la base de données grossisse et soit fragmentée. En conséquence, vous pourriez vouloir effacer l'espace libéré et recoller les pages partiellement utilisées dans la base de données laissées par la suppression des paquets ou après le lancement du ramasse-miettes. Lancer `sudo guix gc --vacuum-database` verrouillera la base de données et lancera un `VACUUM` du dépôt, pour défragmenter la base de données et libérer les pages vides, et déverrouillera la base une fois l'opération terminée.

## 6.6 Invoquer `guix pull`

Les paquets sont installés ou mis à jour vers la dernière version disponible dans la distribution active sur votre machine locale. Pour mettre à jour cette distribution, en même temps que les outils Guix, vous devez lancer `guix pull` ; la commande télécharge le dernier code source de Guix ainsi que des descriptions de paquets, et le déploie. Le code source est téléchargé depuis un dépôt Git (<https://git-scm.com/book/en/>), par défaut le dépôt officiel de GNU Guix, bien que cela puisse être personnalisé. `guix pull` garantit que le code téléchargé est *authentique* en vérifiant que les commits sont signés par les développeur·euses de Guix.

Spécifiquement, `guix pull` télécharge le code depuis les *canaux* (voir voir Chapitre 7 [Canaux], page 75) spécifiés par un des points suivants, dans cet ordre :

1. l'option `--channels` ;
2. le fichier utilisateur-*rice* `~/.config/guix/channels.scm` ;
3. le fichier de l'ensemble du système `/etc/guix/channels.scm` ;
4. les canaux intégrés par défaut spécifiés dans la variable `%default-channels`.

À la fin, `guix package` utilisera les paquets et les versions des paquets de la copie de Guix tout juste récupérée. Non seulement ça, mais toutes les commandes Guix et les modules Scheme seront aussi récupérés depuis la dernière version. Les nouvelles sous-commandes de `guix` ajoutés par la mise à jour sont aussi maintenant disponibles.

Chaque utilisateur peut mettre à jour sa copie de Guix avec `guix pull` et l'effet est limité à l'utilisateur qui a lancé `guix pull`. Par exemple, lorsque l'utilisateur `root` lance `guix pull`, cela n'a pas d'effet sur la version de Guix que voit `alice` et vice-versa.

Le résultat après avoir lancé `guix pull` est un *profil* disponible sous `~/.config/guix/current` contenant la dernière version de Guix. Ainsi, assurez-vous de l'ajouter au début de votre chemin de recherche pour que vous utilisiez la dernière version. Le même conseil s'applique au manuel Info (voir Chapitre 14 [Documentation], page 641) :

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

L'option `--list-generations` ou `-l` liste les anciennes générations produites par `guix pull`, avec des détails sur leur origine :

```
$ guix pull -l
```

```
Génération 1 10 juin 2018 00:18:18
guix 65956ad
  URL du dépôt : https://git.savannah.gnu.org/git/guix.git
  branche : origin/master
  commit : 65956ad3526ba09e1f7a40722c96c6ef7c0936fe
```

```
Génération 2 11 juin 2018 11:02:49
guix e0cc7f6
  URL du dépôt : https://git.savannah.gnu.org/git/guix.git
  branche : origin/master
  commit : e0cc7f669bec22c37481dd03a7941c7d11a64f1d
```

```
Génération 3 13 juin 2018 23:31:07 (actuelle)
guix 844cc1c
  URL du dépôt : https://git.savannah.gnu.org/git/guix.git
  branche : origin/master
  commit : 844cc1c8f394f03b404c5bb3aee086922373490c
```

Voir Section 6.9 [Invoquer guix describe], page 70, pour d'autres manières de décrire le statut actuel de Guix.

Ce profil `~/.config/guix/current` fonctionne comme les profils créés par `guix package` (voir Section 6.2 [Invoquer guix package], page 44). C'est-à-dire que vous pouvez lister les générations, revenir en arrière à une génération précédente — c.-à-d. la version de Guix précédente — etc. :

```
$ guix pull --roll-back
passé de la génération 3 à 2
$ guix pull --delete-generations=1
suppression de /var/guix/profiles/per-user/charlie/current-guix-1-link
```

Vous pouvez aussi utiliser `guix package` (voir Section 6.2 [Invoquer guix package], page 44) pour contrôler le profil en le nommant explicitement :

```
$ guix package -p ~/.config/guix/current --roll-back
passé de la génération 3 à 2
$ guix package -p ~/.config/guix/current --delete-generations=1
suppression de /var/guix/profiles/per-user/charlie/current-guix-1-link
```

La commande `guix pull` est typiquement invoquée sans arguments mais elle prend en charge les options suivantes :

```
--url=url
--commit=commit
--branch=branche
```

Télécharger le code pour le canal `guix` depuis l'*url* spécifié, au *commit* donné (un commit Git valide représenté par une chaîne hexadécimale ou le nom d'une étiquette) ou à la branche *branch*.

Ces options sont fournies pour votre confort, mais vous pouvez aussi spécifier votre configuration dans le fichier `~/.config/guix/channels.scm` ou en utilisant l'option `--channels` (voir plus bas).

`--channels=file`

`-C file` Lit la liste des canaux dans *file* plutôt que dans `~/.config/guix/channels.scm` ou `/etc/guix/channels.scm`. *file* doit contenir un code Scheme qui s'évalue en une liste d'objets de canaux. Voir Chapitre 7 [Canaux], page 75, pour plus d'informations.

`--news`

`-N` Affiche les nouvelles écrites par les auteurs des canaux pour leurs utilisateurs et utilisatrices depuis la génération précédente (voir Chapitre 7 [Canaux], page 75). Lorsque vous passez `--details`, affiche aussi les nouveaux paquets et les paquets mis à jour.

Vous pouvez consulter ces informations pour les générations précédentes avec `guix pull -l`.

`--list-generations[=motif]`

`-l [motif]`

Liste toutes les générations de `~/.config/guix/current` ou, si *motif* est fourni, le sous-ensemble des générations qui correspondent à *motif*. La syntaxe de *motif* est la même qu'avec `guix package --list-generations` (voir Section 6.2 [Invoquer guix package], page 44).

Par défaut, cela affiche les informations sur les canaux utilisés à chaque révision et les nouvelles associées. Si vous passez l'option `--details`, cela affichera la liste des paquets ajoutés et mis à jour à chaque génération par rapport à la précédente.

`--details`

Demande à `--list-generations` ou `--news` d'afficher plus d'informations sur la différence entre les générations successives, voir plus haut.

`--roll-back`

Revenir à la *génération* précédente de `~/.config/guix/current` c.-à-d. défaire la dernière transaction.

`--switch-generation=motif`

`-S motif` Basculer vers une génération particulière définie par le *motif*.

Le *motif* peut être soit un numéro de génération soit un nombre précédé de « + » ou « - ». Ce dernier signifie : se déplacer en avant ou en arrière d'un nombre donné de générations. Par exemple, si vous voulez retourner à la dernière génération après `--roll-back`, utilisez `--switch-generation=+1`.

`--delete-generations[=motif]`

`-d [motif]`

Lorsque *motif* est omis, supprimer toutes les générations en dehors de l'actuelle. Cette commande accepte les mêmes motifs que `--list-generations`. Lorsque *motif* est spécifié, supprime les générations correspondantes. Lorsque *motif* spécifie une durée, les générations *plus anciennes* que la durée spécifiée correspondent. Par exemple `--delete-generations=1m` supprime les générations vieilles de plus d'un mois.

Si la génération actuelle correspond, elle n'est *pas* supprimée.

Remarquez que supprimer des générations empêche de revenir en arrière vers elles. Ainsi, cette commande doit être utilisée avec précaution.

Voir Section 6.9 [Invoquer `guix describe`], page 70, pour une manière d'afficher des informations sur la génération actuelle uniquement.

`--profile=profil`

`-p profil` Utiliser le *profil* à la place de `~/.config/guix/current`.

`--dry-run`

`-n` Montrer quels commits des canaux seraient utilisés et ce qui serait construit ou substitué mais ne pas le faire vraiment.

`--allow-downgrades`

Permet d'extraire des révisions de canaux plus anciennes ou sans rapport avec celles qui sont actuellement utilisées.

Par défaut, `guix pull` protège contre les attaques dites de "downgrade", par lesquelles le dépôt Git d'un canal serait réinitialisé à une révision antérieure ou sans rapport avec lui-même, ce qui pourrait vous conduire à installer des versions plus anciennes de paquets logiciels, ou connues pour être vulnérables.

**Remarque:** Assurez-vous de comprendre les implications de sa sécurité avant d'utiliser `--allow-downgrades`.

`--disable-authentication`

Permet de "tirer" le code du canal sans l'authentifier.

Par défaut, `guix pull` authentifie le code téléchargé depuis les canaux en vérifiant que ses commits sont signés par les développeur·euses, et renvoie une erreur si ce n'est pas le cas. Cette option lui enjoint de ne pas effectuer une telle vérification.

**Remarque:** Assurez-vous de comprendre les implications de sa sécurité avant d'utiliser `--disable-authentication`.

`--system=système`

`-s système`

Tenter de construire pour le *système* — p. ex. `i686-linux` — plutôt que pour le type de système de l'hôte de construction.

`--bootstrap`

Utiliser le programme d'amorçage Guile pour construire la dernière version de Guix. Cette option n'est utile qu'aux personnes qui développent Guix.

Le mécanisme de *canaux* vous permet de dire à `guix pull` quels répertoires et branches récupérer, ainsi que les dépôts *supplémentaires* contenant des modules de paquets qui devraient être déployés. Voir Chapitre 7 [Canaux], page 75, pour plus d'information.

En plus, `guix pull` supporte toutes les options de construction communes (voir Section 10.1.1 [Options de construction communes], page 180).

## 6.7 Invoquer `guix time-machine`

La commande `guix time-machine` donne accès à d'autres révisions de Guix, par exemple pour installer d'anciennes versions de paquets, ou pour reproduire un calcul dans un environnement identique. La révision de Guix à utiliser est définie par un commit ou par

un fichier de description de canal créé par `guix describe` (voir Section 6.9 [Invoquer `guix describe`], page 70).

Supposons que vous vouliez voyager en ces jours de novembre 2020 où la version 1.2.0 de Guix a été publiée et que, une fois que vous y êtes, vous lanciez le `guile` de cette époque :

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

La commande ci-dessus récupère Guix 1.2.0 et lance sa commande `guix environment` pour démarrer un environnement dans un conteneur qui lance `guile` (`guix environment` a depuis été remplacé par `guix shell` ; voir Section 8.1 [Invoquer `guix shell`], page 83). C'est comme conduire une DeLorean<sup>1</sup> ! La première invocation de `guix time-machine` peut être coûteuse. Elle doit télécharger voire construire un grand nombre de paquets. Heureusement le résultat est mis en cache et les commandes suivantes qui ciblent le même commit sont presque instantanées.

**Remarque:** L'histoire de Guix est immuable et `guix time-machine` fournit exactement les mêmes logiciels que ceux de la révision de Guix spécifique. Naturellement, aucun correctif de sécurité n'est fourni pour les anciennes versions de Guix ou ses canaux. Utiliser `guix time-machine` sans précaution ouvre la porte à des vulnérabilités de sécurité. Voir Section 6.6 [Invoquer `guix pull`], page 63.

La syntaxe générale est :

```
guix time-machine options... -- commande arg...
```

où *command* et *arg...* sont passés sans modification à la commande `guix` de la révision spécifiée. Les *options* qui définissent cette révision sont les mêmes que pour la commande `guix pull` (voir Section 6.6 [Invoquer `guix pull`], page 63) :

`--url=url`

`--commit=commit`

`--branch=branche`

Utiliser le canal `guix` depuis l'*url* spécifiée, au *commit* donné (un commit Git valide représenté par une chaîne hexadécimale ou le nom d'une étiquette) ou à la branche *branch*.

`--channels=file`

`-C file` Lit la liste des canaux dans *file*. *file* doit contenir un code Scheme qui s'évalue en une liste d'objets de canaux. Voir Chapitre 7 [Canaux], page 75, pour plus d'informations.

Quant à `guix pull`, l'absence d'options signifie que le dernier commit sur la branche master sera utilisé. La commande

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

va donc construire le paquet `hello` tel que défini dans la branche master, qui est en général une révision de Guix plus récente que celle que vous avez installée. Le voyage dans le temps fonctionne dans les deux sens !

Remarquez que `guix time-machine` peut lancer des constructions de canaux et de leurs dépendances, et qu'elles peuvent être contrôlées avec les options de construction communes (voir Section 10.1.1 [Options de construction communes], page 180).

<sup>1</sup> Si vous ne savez pas ce qu'est une DeLorean, envisagez un voyage vers les années 1980.

## 6.8 Inférieurs

**Remarque:** La fonctionnalité décrite ici est un « démonstrateur technique » à la version 1.4.0. Ainsi, l'interface est sujette à changements.

Parfois vous pourriez avoir à mélanger des paquets de votre révision de Guix avec des paquets disponibles dans une révision différente de Guix. Les *inférieurs* de Guix vous permettent d'accomplir cette tâche en composant différentes versions de Guix de manière arbitraire.

Techniquement, un « inférieur » est surtout un processus Guix séparé connecté à votre processus Guix principal à travers un REPL (voir Section 9.13 [Invoquer guix repl], page 176). Le module (`guix inferior`) vous permet de créer des inférieurs et de communiquer avec eux. Il fournit aussi une interface de haut-niveau pour naviguer dans les paquets d'un inférieur — *des paquets inférieurs* — et les manipuler.

Lorsqu'on les combine avec des canaux (voir Chapitre 7 [Canaux], page 75), les inférieurs fournissent une manière simple d'interagir avec un révision de Guix séparée. Par exemple, disons que vous souhaitez installer dans votre profil le paquet `guile` actuel, avec le `guile-json` d'une ancienne révision de Guix — peut-être parce que la nouvelle version de `guile-json` a une API incompatible et que vous voulez lancer du code avec l'ancienne API. Pour cela, vous pourriez écrire un manifeste à utiliser avec `guix package --manifest` (voir Section 9.4 [Écrire un manifeste], page 120) ; dans ce manifeste, vous créeriez un inférieur pour l'ancienne révision de Guix qui vous intéresse et vous chercheriez le paquet `guile-json` dans l'inférieur :

```
(use-modules (guix inferior) (guix channels)
             (srfi srfi-1)) ;pour « first »

(define channels
  ;; L'ancienne révision depuis laquelle on veut
  ;; extraire guile-json.
  (list (channel
        (name 'guix)
        (url "https://git.savannah.gnu.org/git/guix.git")
        (commit
         "65956ad3526ba09e1f7a40722c96c6ef7c0936fe"))))

(define inferior
  ;; Un inférieur représentant la révision ci-dessus.
  (inferior-for-channels channels))

;; Maintenant on crée un manifeste avec le paquet « guile » actuel
;; et l'ancien paquet « guile-json ».
(packages->manifest
 (list (first (lookup-inferior-packages inferior "guile-json"))
       (specification->package "guile")))
```

Durant la première exécution, `guix package --manifest` pourrait avoir besoin de construire le canal que vous avez spécifié avant de créer l'inférieur ; les exécutions suivantes seront bien plus rapides parce que la révision de Guix sera déjà en cache.

Le module (`guix inferior`) fournit les procédures suivantes pour ouvrir un inférieur :

`inferior-for-channels channels` [`#:cache-directory`] [Procédure Scheme]  
 [`#:ttl`]

Renvoie un inférieur pour *channels*, une liste de canaux. Elle utilise le cache dans *cache-directory*, où les entrées peuvent être glanées après *ttl* secondes. Cette procédure ouvre une nouvelle connexion au démon de construction.

Elle a pour effet de bord de construire ou de substituer des binaires pour *channels*, ce qui peut prendre du temps.

`open-inferior directory` [`#:command "bin/guix"`] [Procédure Scheme]

Ouvre le Guix inférieur dans *directory* et lance *directory/command repl* ou équivalent. Renvoie `#f` si l'inférieur n'a pas pu être lancé.

Les procédures listées plus bas vous permettent d'obtenir et de manipuler des paquets inférieurs.

`inferior-packages inferior` [Procédure Scheme]

Renvoie la liste des paquets connus de l'inférieur *inferior*.

`lookup-inferior-packages inferior name` [`version`] [Procédure Scheme]

Renvoie la liste triée des paquets inférieurs qui correspondent à *name* dans *inferior*, avec le plus haut numéro de version en premier. Si *version* est vrai, renvoie seulement les paquets avec un numéro de version préfixé par *version*.

`inferior-package? obj` [Procédure Scheme]

Renvoie vrai si *obj* est un paquet inférieur.

`inferior-package-name package` [Procédure Scheme]

`inferior-package-version package` [Procédure Scheme]

`inferior-package-synopsis package` [Procédure Scheme]

`inferior-package-description package` [Procédure Scheme]

`inferior-package-home-page package` [Procédure Scheme]

`inferior-package-location package` [Procédure Scheme]

`inferior-package-inputs package` [Procédure Scheme]

`inferior-package-native-inputs package` [Procédure Scheme]

`inferior-package-propagated-inputs package` [Procédure Scheme]

`inferior-package-transitive-propagated-inputs package` [Procédure Scheme]

`inferior-package-native-search-paths package` [Procédure Scheme]

`inferior-package-transitive-native-search-paths package` [Procédure Scheme]

`inferior-package-search-paths package` [Procédure Scheme]

Ces procédures sont la contrepartie des accesseurs des enregistrements de paquets

(voir Section 9.2.1 [référence de package], page 108). La plupart fonctionne en effec-

tuant des requêtes à l'inférieur dont provient *package*, donc l'inférieur doit toujours

être disponible lorsque vous appelez ces procédures.



Les paquets inférieurs peuvent être utilisés de manière transparente comme tout autre paquet ou objet simili-fichier dans des G-expressions (voir Section 9.12 [G-Expressions], page 166). Ils sont aussi gérés de manière transparente par la procédure `packages->manifest`, qui est typiquement utilisée dans des manifestes (voir Section 6.2 [Invoquer guix package], page 44). Ainsi, vous pouvez insérer un paquet inférieur à peu près n'importe où vous utiliseriez un paquet normal : dans des manifestes, dans le champ `packages` de votre déclaration `operating-system`, etc.

## 6.9 Invoquer guix describe

Souvent vous voudrez répondre à des questions comme « quelle révision de Guix j'utilise ? » ou « quels canaux est-ce que j'utilise ? ». C'est une information utile dans de nombreuses situations : si vous voulez *répliquer* un environnement sur une machine différente ou un compte utilisateur, si vous voulez rapporter un bogue ou pour déterminer quel changement dans les canaux que vous utilisez l'a causé ou si vous voulez enregistrer l'état de votre système pour le reproduire. La commande `guix describe` répond à ces questions.

Lorsqu'elle est lancée depuis un `guix` mis à jour avec `guix pull`, `guix describe` affiche les canaux qui ont été construits, avec l'URL de leur dépôt et l'ID de leur commit (voir Chapitre 7 [Canaux], page 75) :

```
$ guix describe
Generation 10 03 sep. 2018 17:32:44 (actuelle)
guix e0fa68c
  URL du dépôt : https://git.savannah.gnu.org/git/guix.git
  branche : master
  commit : e0fa68c7718fffd33d81af415279d6ddb518f727
```

Si vous connaissez bien le système de contrôle de version Git, cela ressemble en essence à `git describe` ; la sortie est aussi similaire à celle de `guix pull --list-generations`, mais limitée à la génération actuelle (voir Section 6.6 [Invoquer guix pull], page 63). Comme l'ID de commit de Git ci-dessus se réfère sans aucune ambiguïté à un instantané de Guix, cette information est tout ce dont vous avez besoin pour décrire la révision de Guix que vous utilisez et pour la répliquer.

Pour rendre plus facile la réplification de Guix, `guix describe` peut aussi renvoyer une liste de canaux plutôt que la description lisible par un humain au-dessus :

```
$ guix describe -f channels
(list (channel
      (name 'guix)
      (url "https://git.savannah.gnu.org/git/guix.git")
      (commit
        "e0fa68c7718fffd33d81af415279d6ddb518f727")))
      (introduction
        (make-channel-introduction
          "9edb3f66fd807b096b48283debdccdcfea34bad"
          (openpgp-fingerprint
            "BBB0 2DDF 2CEA F6A8 OD1D E643 A2A0 6DF2 A33A 54FA")))))
```

Vous pouvez sauvegarder ceci dans un fichier et le donner à `guix pull -C` sur une autre machine ou plus tard, ce qui instantiera *exactement la même révision de Guix* (voir Sec-

tion 6.6 [Invoquer guix pull], page 63). À partir de là, comme vous pouvez déployer la même révision de Guix, vous pouvez aussi bien *répliquer un environnement logiciel complet*. Nous pensons humblement que c'est *génial*, et nous espérons que vous aimerez ça aussi !

Voici les détails des options supportées par `guix describe` :

`--format=format`

`-f format` Produire la sortie dans le *format* donné, parmi :

`human` produire une sortie lisible par un humain ;

`canaux` produire une liste de spécifications de canaux qui peut être passée à `guix pull -C` ou installée dans `~/.config/guix/channels.scm` (voir Section 6.6 [Invoquer guix pull], page 63) ;

`channels-sans-intro`

comme `canaux`, mais oubliez le champ `introduction` ; utilisez-le pour produire une spécification de canal adaptée à la version 1.1.0 ou antérieure de Guix—le champ `introduction` concerne l'authentification des canaux (voir Chapitre 7 [Canaux], page 75) et n'est pas pris en charge par ces versions antérieures ;

`json` produire une liste de spécifications de canaux dans le format JSON ;

`recutils` produire une liste de spécifications de canaux dans le format Recutils.

`--list-formats`

Affiche les formats disponibles pour l'option `--format`.

`--profile=profil`

`-p profil` Afficher les informations sur le *profil*.

## 6.10 Invoquer guix archive

La commande `guix archive` permet aux utilisateurs d'*exporter* des fichiers du dépôt dans une simple archive puis ensuite de les *importer* sur une machine qui fait tourner Guix. En particulier, elle permet de transférer des fichiers du dépôt d'une machine vers le dépôt d'une autre machine.

**Remarque:** Si vous cherchez une manière de produire des archives dans un format adapté pour des outils autres que Guix, voir Section 8.3 [Invoquer guix pack], page 96.

Pour exporter des fichiers du dépôt comme une archive sur la sortie standard, lancez :

```
guix archive --export options spécifications...
```

*spécifications* peut être soit des noms de fichiers soit des spécifications de paquets, comme pour `guix package` (voir Section 6.2 [Invoquer guix package], page 44). Par exemple, la commande suivante crée une archive contenant la sortie `gui` du paquet `git` et la sortie principale de `emacs` :

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

Si les paquets spécifiés ne sont pas déjà construits, `guix archive` les construit automatiquement. Le processus de construction peut être contrôlé avec les options de construction communes (voir Section 10.1.1 [Options de construction communes], page 180).

Pour transférer le paquet `emacs` vers une machine connectée en SSH, on pourrait lancer :

```
guix archive --export -r emacs | ssh la-machine guix archive --import
```

De même, on peut transférer un profil utilisateur complet d'une machine à une autre comme cela :

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

Toutefois, il faut noter que, dans les deux exemples, tous les `emacs` et le profil, ainsi que toutes leurs dépendances sont transférés (en raison de l'`-r`), indépendamment de ce qui est déjà disponible dans le dépôt sur la machine cible. L'option `--missing` peut aider à déterminer quels sont les éléments manquants dans le dépôt cible. La commande `guix copy` simplifie et optimise tout ce processus, c'est donc probablement ce que vous devriez utiliser dans ce cas (voir Section 10.13 [Invoquer `guix copy`], page 229).

Chaque élément du dépôt est écrit dans le format *archive normalisée* ou *nar* (décrit ci-dessous), et la sortie de `guix archive --export` (et entrée de `guix archive --import`) est un *nar bundle*.

Le format *nar* est comparable dans l'esprit au format "tar", mais avec des différences qui le rendent plus approprié à nos objectifs. Premièrement, plutôt que d'enregistrer toutes les métadonnées Unix pour chaque fichier, le format *nar* ne mentionne que le type de fichier (régulier, répertoire ou lien symbolique) ; les autorisations Unix et le propriétaire/groupe sont rejetés. Deuxièmement, l'ordre dans lequel les entrées de répertoire sont stockées, suit toujours celui des noms de fichiers selon leur vérification de concordance des locales C. Cela rend la production d'archives entièrement déterministe.

Ce format de lot *nar* est surtout la concaténation de zéro, un ou plusieurs *nar* avec des métadonnées pour chaque élément du dépôt qu'il contient : son nom de fichier, ses références, la dérivation correspondante et une signature numérique.

Lors de l'export, le démon signe numériquement le contenu de l'archive et cette signature est ajoutée à la fin du fichier. Lors de l'import, le démon vérifie la signature et rejette l'import en cas de signature invalide ou si la clef de signature n'est pas autorisée.

Les principales options sont :

`--export` Exporter les fichiers ou les paquets spécifiés du dépôt (voir plus bas). Écrire l'archive résultante sur la sortie standard.

Les dépendances ne sont *pas* incluses dans la sortie à moins que `--recursive` ne soit passé.

`-r`

`--recursive`

En combinaison avec `--export`, cette option demande à `guix archive` d'inclure les dépendances des éléments donnés dans l'archive. Ainsi, l'archive résultante est autonome : elle contient la fermeture des éléments de dépôt exportés.

**--import** Lire une archive depuis l'entrée standard et importer les fichiers inclus dans le dépôt. Annuler si l'archive a une signature invalide ou si elle est signée par une clef publique qui ne se trouve pas dans les clefs autorisées (voir **--authorize** plus bas).

**--missing** Liste une liste de noms de fichiers du dépôt sur l'entrée standard, un par ligne, et écrit sur l'entrée standard le sous-ensemble de ces fichiers qui manquent dans le dépôt.

**--generate-key[=paramètres]** Génère une nouvelle paire de clefs pour le démon. C'est un prérequis avant que les archives ne puissent être exportées avec **--export**. Cette opération est habituellement instantanée mais elle peut prendre du temps parce qu'elle doit récupérer suffisamment d'entropie pour générer la paire de clefs. Sur Guix System, **guix-service-type** prend soin de générer cette paire de clés au premier démarrage.

La paire de clés générée est typiquement stockée dans `/etc/guix`, dans **signing-key.pub** (clé publique) et **signing-key.sec** (clé privée, qui doit rester secrète). Lorsque *paramètres* est omis, une clé ECDSA utilisant la courbe Ed25519 est générée, ou pour les versions de libgcrypt avant 1.6.0, une clé RSA de 4096 bits. Autrement, *paramètres* peut spécifier les paramètres **genkey** adaptés pour libgcrypt (voir Section "General public-key related Functions" dans *The Libgcrypt Reference Manual*).

**--authorize** Autoriser les imports signés par la clef publique passée sur l'entrée standard. La clef publique doit être au « format avancé s-expression » — c.-à-d. le même format que le fichier **signing-key.pub**.

La liste des clés autorisées est gardée dans un fichier modifiable par des humains dans `/etc/guix/acl`. Le fichier contient des « s-expressions au format avancé » (<https://people.csail.mit.edu/rivest/Sexp.txt>) et est structuré comme une liste de contrôle d'accès dans l'infrastructure à clefs publiques simple (SPKI) (<https://theworld.com/~cme/spki.txt>).

**--extract=répertoire**

**-x répertoire**

Lit une archive à un seul élément telle que servie par un serveur de substituts (voir Section 6.3 [Substituts], page 54) et l'extrait dans *répertoire*. C'est une opération de bas niveau requise seulement dans de rares cas d'usage ; voir plus loin.

Par exemple, la commande suivante extrait le substitut pour Emacs servi par `ci.guix.gnu.org` dans `/tmp/emacs` :

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

Les archives à un seul élément sont différentes des archives à plusieurs éléments produites par `guix archive --export` ; elles contiennent un seul élément du

dépôt et elles n'embarquent *pas* de signature. Ainsi cette opération ne vérifie *pas* de signature et sa sortie devrait être considérée comme non sûre.

Le but principal de cette opération est de faciliter l'inspection du contenu des archives venant de serveurs auxquels on ne fait potentiellement pas confiance. (voir Section 10.12 [Invoquer guix challenge], page 226).

`--list`

`-t` Lit une archive à un seul élément telle que servie par un serveur de substituts (voir Section 6.3 [Substituts], page 54) et affiche la liste des fichiers qu'elle contient, comme dans cet exemple :

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

## 7 Canaux

Guix et sa collection de paquets sont mis à jour en lançant `guix pull` (voir Section 6.6 [Invoquer `guix pull`], page 63). Par défaut `guix pull` télécharge et déploie Guix lui-même depuis le dépôt officiel de GNU Guix. Cela peut être personnalisé en définissant des *canaux* dans le fichier `~/.config/guix/channels.scm`. Un canal spécifie l'URL et la branche d'un répertoire Git à déployer et on peut demander à `guix pull` de récupérer un ou plusieurs canaux. En d'autres termes, les canaux peuvent être utilisés pour *personnaliser* et pour *étendre* Guix, comme on le verra plus bas. Guix est en mesure de prendre en compte les préoccupations de sécurité et de traiter les mises à jour authentifiées.

### 7.1 Spécifier des canaux supplémentaires

Vous pouvez spécifier des *canaux supplémentaires* à utiliser. Pour utiliser un canal, écrivez dans `~/.config/guix/channels.scm` pour dire à `guix pull` de récupérer votre canal personnel *en plus* des canaux par défaut de Guix :

```
;; Ajouter des variantes de paquets à ceux fournis par Guix.
(cons (channel
      (name 'my-personal-packages)
      (url "https://example.org/personal-packages.git"))
      %default-channels)
```

Remarquez que le bout de code au-dessus est (comme toujours !) du code Scheme ; nous utilisons `cons` pour ajouter un canal à la liste des canaux que la variable `%default-channels` représente (voir Section “Pairs” dans *GNU Guile Reference Manual*). Avec ce fichier en place, `guix pull` construit non seulement Guix mais aussi les modules de paquets de votre propre dépôt. Le résultat dans `~/.config/guix/current` est l'union de Guix et de vos propres modules de paquets :

```
$ guix describe
Génération 19 Aug 27 2018 16:20:48
guix d894ab8
  URL du dépôt : https://git.savannah.gnu.org/git/guix.git
  branche : master
  commit : d894ab8e9bfabcefa6c49d9ba2e834dd5a73a300
variant-packages dd3df5e
  URL du dépôt : https://example.org/personal-packages.git
  branche : master
  commit : dd3df5e2c8818760a8fc0bd699e55d3b69fef2bb
```

La sortie de `guix describe` ci-dessus montre que nous sommes maintenant à la génération 19 et qu'il contient à la fois Guix et les paquets du canal `variant-personal-packages` (voir Section 6.9 [Invoquer `guix describe`], page 70).

### 7.2 Utiliser un canal Guix personnalisé

Le canal nommé `guix` spécifie où Guix lui-même — ses outils en ligne de commande ainsi que sa collection de paquets — seront téléchargés. Par exemple, supposons que vous voulez effectuer les mises à jour depuis votre propre copie du dépôt Guix sur `example.org`, et plus

particulièrement depuis la branche `super-hacks`. Vous pouvez écrire cette spécification dans `~/.config/guix/channels.scm` :

```
;; Dit à « guix pull » d'utiliser un autre dépôt.
(list (channel
      (name 'guix)
      (url "https://example.org/my-guix.git")
      (branch "super-hacks")))
```

Maintenant, `guix pull` récupérera le code depuis la branche `super-hacks` du dépôt sur `example.org`. La question de l'authentification est traitée ci-dessous (voir Section 7.4 [Authentification des canaux], page 77).

### 7.3 Répliquer Guix

La commande `guix describe` ci-dessus montre précisément quels commits ont été utilisés pour construire l'instance de Guix utilisée (voir Section 6.9 [Invoquer `guix describe`], page 70). Nous pouvons répliquer cette instance sur une autre machine ou plus tard, en fournissant une spécification de canal « épinglée » à ces commits et qui ressemble à ceci :

```
;; Déployer des commits précis de mes canaux préférés.
(list (channel
      (name 'guix)
      (url "https://git.savannah.gnu.org/git/guix.git")
      (commit "6298c3ffd9654d3231a6f25390b056483e8f407c"))
      (channel
      (name 'variant-packages)
      (url "https://example.org/variant-packages.git")
      (commit "dd3df5e2c8818760a8fc0bd699e55d3b69fef2bb")))
```

Pour obtenir cette spécification de canaux épinglée, le plus simple est de lancer `guix describe` et de sauvegarder sa sortie au format `channels` dans un fichier, comme ceci :

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

Le fichier `channels.scm` qui en résulte peut être passé à l'option `-C` de `guix pull` (voir Section 6.6 [Invoquer `guix pull`], page 63) ou `guix time-machine` (voir Section 6.7 [Invoquer `guix time-machine`], page 66), comme dans cet exemple :

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

Étant donné le fichier `channels.scm`, la commande ci-dessus récupérera toujours *exactement la même instance de Guix*, puis utilisera cette instance pour lancer exactement le même Python (voir Section 8.1 [Invoquer `guix shell`], page 83). Sur n'importe quelle machine, à n'importe quel moment, elle lance exactement le même binaire, bit à bit.

Les canaux épinglés répondent à un problème similaire à celui des « fichiers de verrouillage » implémentés par certains outils de déploiement — ils vous permettent d'épingler et de reproduire un ensemble de paquets. Dans le cas de Guix cependant, vous épinglez en fait l'entièreté de l'ensemble de paquets défini par les commits donnés des canaux ; en fait, vous épinglez l'ensemble de Guix, avec des modules cœurs et ses outils en ligne de commande. Vous gagnez aussi la garantie forte que vous obtenez, effectivement, exactement le même environnement logiciel.

Cela vous donne des super-pouvoirs, ce qui vous permet de suivre la provenance des artefacts binaires avec un grain très fin et de reproduire les environnements logiciels à volonté — une sorte de capacité de « méta-reproductibilité », si vous voulez. Voir Section 6.8 [Inférieurs], page 68, pour une autre manière d'utiliser ces super-pouvoirs.

## 7.4 Authentification des canaux

Les commandes `guix pull` et `guix time-machine authenticate` téléchargent et contrôlent le code récupéré sur les canaux : elles s'assurent que chaque commit récupéré est signé par un·e développeur·euse autorisé·e. L'objectif est de protéger contre les modifications non autorisées du canal qui conduiraient les utilisateur·rice·s à exécuter du code malveillant.

En tant qu'utilisateur, vous devez fournir une *introduction de canal* dans votre fichier de canaux afin que Guix sache comment authentifier son premier commit. Une spécification de canal, y compris son introduction, ressemble à quelque chose de ce genre :

```
(channel
  (name 'some-channel)
  (url "https://example.org/some-channel.git")
  (introduction
    (make-channel-introduction
      "6f0d8cc0d88abb59c324b2990bfee2876016bb86"
      (openpgp-fingerprint
        "CABB A931 COFF EEC6 900D 0CFB 090B 1199 3D9A EBB5"))))
```

La spécification ci-dessus indique le nom et l'URL du canal. L'appel à `make-channel-introduction` ci-dessus spécifie que l'authentification de ce canal commence au commit `6f0d8cc...`, qui est signé par la clé OpenPGP avec l'empreinte digitale `CABB A931...`

Pour le canal principal, appelé `guix`, vous obtenez automatiquement cette information à partir de votre installation de Guix. Pour les autres canaux, incluez l'introduction du canal fournie par les auteurs du canal dans votre fichier `channels.scm`. Assurez-vous de récupérer l'introduction du canal à partir d'une source fiable, car c'est la base de votre confiance.

Si vous êtes curieux·euse à propos des mécanismes d'authentification, lisez !

## 7.5 Canaux avec des substituts

Lorsque vous lancez `guix pull`, Guix compilera d'abord les définitions de tous les paquets disponibles. C'est une opération coûteuse pour laquelle des substituts (voir Section 6.3 [Substituts], page 54) peuvent être disponibles. L'extrait de code suivant dans `channels.scm` s'assure que `guix pull` utilise le dernier commit pour lequel des substituts sont disponibles pour les définitions des paquets : pour cela, on demande au serveur d'intégration continue `https://ci.guix.gnu.org`.

```
(use-modules (guix ci))

(list (channel-with-substitutes-available
      %default-guix-channel
      "https://ci.guix.gnu.org"))
```

Remarquez que cela ne signifie pas que tous les paquets que vous installerez après avoir lancé `guix pull` auront des substituts. Cela s'assure uniquement que `guix pull` n'essaiera



pas de compiler les définitions des paquets. C'est particulièrement pratique si vous utilisez une machine avec des ressources limitées.

## 7.6 Écrire de nouveaux de canaux

Disons que vous avez un ensemble de paquets personnels ou de variantes personnalisées dont vous pensez qu'il ne vaudrait pas le coup de contribuer au projet Guix, mais que vous voudriez pouvoir utiliser de manière transparente sur la ligne de commande : vous écririez d'abord des modules contenant ces définitions de paquets (voir Section 9.1 [Modules de paquets], page 104), en les maintenant dans un dépôt Git, puis vous ou n'importe qui d'autre pourrait l'utiliser comme un canal supplémentaire où trouver ces paquets. Sympa, non ?

**Attention:** Avant que vous, cher utilisateur, ne vous exclamiez « Oh mais c'est *super génial* ! » et que vous ne publiez vos canaux personnels publiquement, nous voudrions vous donner quelques avertissements :

- Avant de publier un canal, envisagez de contribuer vos définitions de paquets dans Guix (voir Chapitre 22 [Contribuer], page 668). Guix en tant que projet est ouvert à tous les logiciels libres de toutes sortes, et les paquets dans Guix sont déjà disponibles à tous les utilisateurs de Guix et bénéficient des processus d'assurance qualité du projet.
- Lorsque vous maintenez des définitions de paquets en dehors de Guix, nous, les développeur·euses de Guix, considérons que *la charge de la compatibilité vous incombe*. Rappelez-vous que les modules de paquets et les définitions de paquets ne sont que du code Scheme qui utilise diverses interfaces de programmation (API). Nous souhaitons rester libres de changer ces API pour continuer à améliorer Guix, éventuellement d'une manière qui casse votre canal. Nous ne changeons jamais l'API gratuitement, mais nous ne nous engageons *pas* à geler les API non plus.
- Corollaire : si vous utilisez un canal externe et que le canal est cassé, merci de *rapporter le problème à l'auteur du canal*, pas au projet Guix.

Vous avez été prévenus ! Maintenant, nous pensons que des canaux externes sont une manière pratique d'exercer votre liberté pour augmenter la collection de paquets de Guix et de partager vos améliorations, qui sont les principes de bases du logiciel libre (<https://www.gnu.org/philosophy/free-sw.html>). Contactez-nous par courriel sur [guix-devel@gnu.org](mailto:guix-devel@gnu.org) si vous souhaitez discuter à ce propos.

Pour créer un canal, créez un dépôt Git contenant vos propres modules de paquets et rendez-le disponible. Le dépôt peut contenir tout ce que vous voulez, mais un canal utile contiendra des modules Guile qui exportent des paquets. Une fois que vous avez démarré un canal, Guix se comportera comme si le répertoire de la racine du dépôt Git de ce canal était ajouté au chemin de chargement de Guile (voir Section “Load Paths” dans *GNU Guile Reference Manual*). Par exemple, si votre canal contient un fichier `mes-paquets/mes-outils.scm` qui définit un module Guile, le module sera disponible sous le nom de (`mes-paquets mes-outils`) et vous pourrez l'utiliser comme les autres modules (voir Section “Modules” dans *GNU Guile Reference Manual*).

En tant qu'auteur.e d'un canal, envisagez de regrouper le nécessaire d'authentification avec votre canal afin que les utilisatrice-eur-s puissent l'authentifier. Voir Section 7.4 [Authentification des canaux], page 77, et Section 7.9 [Spécifier les autorisations des canaux], page 80, pour savoir comment les utiliser.

## 7.7 Modules de paquets dans un sous-répertoire

En tant qu'auteur.e d'un canal, vous voudriez garder vos modules de canal dans un sous-répertoire. Si vos modules sont dans le sous-répertoire `guix`, vous devez ajouter un fichier de métadonnées `.guix-channel` qui contient :

```
(channel
  (version 0)
  (directory "guix"))
```

## 7.8 Déclarer des dépendances de canaux

Les auteurs de canaux peuvent décider d'augmenter une collection de paquets fournie par d'autres canaux. Ils peuvent déclarer leur canal comme dépendant d'autres canaux dans le fichier de métadonnées `.guix-channel` qui doit être placé à la racine de dépôt du canal.

Le fichier de métadonnées devrait contenir une S-expression simple comme cela :

```
(channel
  (version 0)
  (dependencies
    (channel
      (name some-collection)
      (url "https://example.org/first-collection.git")

      ;; La partie « introduction » ci-dessous est facultative : vous la
      ;; fournissez pour les dépendances qui peuvent être authentifiées.
      (introduction
        (channel-introduction
          (version 0)
          (commit "a8883b58dc82e167c96506cf05095f37c2c2c6cd")
          (signer "CABB A931 COFF EEC6 900D OCFB 090B 1199 3D9A EBB5")))))
    (channel
      (name some-other-collection)
      (url "https://example.org/second-collection.git")
      (branch "testing"))))
```

Dans l'exemple ci-dessus, ce canal est déclaré comme dépendant de deux autres canaux, qui seront récupérés automatiquement. Les modules fournis par le canal seront compilés dans un environnement où les modules de tous les canaux déclarés sont disponibles.

Pour des raisons de fiabilité et de maintenabilité, vous devriez éviter d'avoir des dépendances sur des canaux que vous ne maîtrisez pas et vous devriez ajouter le minimum de dépendances possible.

## 7.9 Spécifier les autorisations des canaux

Comme nous l'avons vu plus haut, Guix garantit le code source qu'il récupère depuis les canaux venant de développeur·euses autorisé·es. En tant qu'auteur·e, vous devez spécifier la liste des développeur·euses autorisé·es dans le fichier `.guix-authorizations` dans le dépôt Git des canaux. Le rôle de l'authentification est simple : chaque commit doit être signé par une clé listée dans le fichier `.guix-authorizations` de son (ses) commit(s) parent(s)<sup>1</sup> Le fichier `.guix-authorizations` ressemble à ceci :

```
;; Fichier d'exemple '.guix-authorizations'.

(authorizations
 (version 0)                ;version actuelle du format de fichier

 ((("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  OCFB 090B 1199 3D9A EBB5"
   (name "charlie")))))
```

Chaque empreinte digitale est suivie de paires de clés/valeurs facultatives, comme dans l'exemple ci-dessus. Actuellement, ces paires clé/valeur sont ignorées.

Ce rôle d'authentification crée un problème de poule et d'œuf : comment authentifions-nous le premier commit ? Par rapport à cela : comment traiter les canaux dont l'historique du dépôt contient des commits non signés et qui n'ont pas de `.guix-authorizations` ? Et comment bifurquer des canaux existants ?

L'introduction des canaux répondent à ces questions en décrivant le premier commit d'un canal qui doit être authentifiée. La première fois qu'un canal est récupéré avec `guix pull` ou `guix time-machine`, la commande recherche le commit d'introduction et vérifie qu'il est signé par la clé OpenPGP spécifiée. Elle authentifie ensuite les commits selon la règle ci-dessus. L'authentification échoue si le commit cible n'est ni un descendant ni un ancêtre du commit d'introduction.

De plus, votre canal doit fournir toutes les clés OpenPGP qui ont été mentionnées dans les fichiers `.guix-authorizations`, stockés dans les fichiers `.key`, qui peuvent être soit binaires soit "blindés ASCII". Par défaut, ces fichiers `.key` sont recherchés dans la branche nommée `keyring`, mais vous pouvez spécifier un nom de branche différent dans `.guix-channel` de cette façon :

```
(channel
 (version 0)
 (keyring-reference "my-keyring-branch"))
```

En résumé, en tant qu'auteur·e d'une chaîne, il y a trois choses que vous devez faire pour permettre aux utilisateur·rice·s d'authentifier votre code :

<sup>1</sup> Les commits de Git forment un *graphe acyclique orienté* (DAG en anglais). Chaque commit peut avoir zéro ou plusieurs parents ; les commits « usuels » ont un seul parent et les commits de fusion ont deux parents. Lisez *Git for Computer Scientists* (<https://eagain.net/articles/git-for-computer-scientists/>) pour une bonne vue d'ensemble.

1. Exporter les clés OpenPGP des committers passés et présents avec `gpg -export` et les stocker dans des fichiers `.key`, par défaut dans une branche nommée `keyring` (nous recommandons d'en faire une *branche orpheline*).
2. Introduisez un fichier `.guix-authorisations` initial dans le dépôt du canal. Faites cela dans un commit signé (voir Section 22.8 [Accès en commit], page 691, pour savoir comment signer les commit Git).
3. Annoncez l'introduction du canal, par exemple sur la page web de votre canal. L'introduction du canal, comme nous l'avons vu ci-dessus, est la paire de clés commit—c'est-à-dire le commit qui a introduit `.guix-authorizations`, et l'empreinte digitale de l'OpenPGP utilisé pour le signer.

Avant de pousser vers votre dépôt Git public, vous pouvez lancer `guix git-authenticate` pour vérifier que vous avez bien signé tous les commits que vous allez pousser avec une clé autorisée :

```
guix git authenticate commit signer
```

où `commit` et `signer` sont votre présentation de canal. Voir Section 8.5 [Invoquer `guix git authenticate`], page 102, pour plus de détails.

Publier un canal signé demande de la discipline : toute faute, comme un commit non signé ou un commit signée par une clé non autorisée, vont empêcher les utilisateur·rice·s de récupérer depuis votre canal—bref, c'est tout l'intérêt de l'authentification ! Faites particulièrement attention aux merges : les merge commits sont considérés comme authentiques si et seulement s'ils sont signés par une clé présente dans le fichier `.guix-authorizations` des branches *both*.

## 7.10 URL primaire

Les auteur.e.s de canaux peuvent indiquer l'URL principale de leur dépôt Git de canal dans le fichier `.guix-channel` comme ceci :

```
(channel
 (version 0)
 (url "https://example.org/guix.git"))
```

Cela permet à `guix pull` de déterminer si elle récupère du code d'un miroir du canal ; lorsque c'est le cas, elle avertit l'utilisateur·rice que le miroir pourrait être périmé et affiche l'URL primaire. De cette façon, les utilisateur·rice·s ne peuvent pas être amené·e·s à extraire du code d'un miroir périmé qui ne reçoit pas les mises à jour de sécurité.

Cette fonctionnalité n'a de sens que pour les dépôts authentifiés, tels que le canal officiel `guix`, pour lequel `guix pull` garantit l'authenticité du code qu'il récupère.

## 7.11 Écrire des nouveautés de canaux

Les auteur.e.s de canaux peuvent occasionnellement vouloir communiquer à leurs utilisateur·rice·s des informations au sujet de changements importants dans le canal. Vous pourriez leur envoyer un courriel, mais ce n'est pas pratique.

Au lieu de cela, les canaux peuvent fournir un *fichier de nouvelles* ; lorsque les utilisateur·rice·s du canal exécutent `guix pull`, ce fichier de nouvelles est automatiquement

lu et `guix pull --news` peut afficher les annonces qui correspondent aux nouveaux commits qui ont été récupérés, le cas échéant.

Pour cela, les auteur-e-s de canaux doivent en premier lieu déclarer le nom du fichier de nouvelles dans leur fichier `.guix-channel` :

```
(channel
  (version 0)
  (news-file "etc/news.txt"))
```

Le fichier de nouvelles lui-même, `etc/news.txt` dans cet exemple, doit ressembler à quelque chose comme ceci :

```
(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!"))))
```

Tandis que le fichier de nouvelles utilise la syntaxe Scheme, évitez de le nommer avec une extension `.scm`, sinon il sera récupéré lors de la construction du canal et produira une erreur puisqu'il ne s'agit pas d'un module valide. Vous pouvez également déplacer le module du canal dans un sous-répertoire et stocker le fichier de nouvelles dans un autre répertoire.

Le fichier consiste en une liste de *news entries*. Chaque entrée est associée à un commit ou un tag : elle décrit les changements faits dans ce commit, éventuellement dans les commits précédents comme il faut. Les utilisateur-ric-e-s ne voient les entrées que la première fois qu'il-elle-s obtiennent le commit auquel l'entrée se réfère.

Le **titre** doit être un résumé d'une ligne tandis que **body** peut être arbitrairement long, et les deux peuvent contenir des balises Texinfo (voir Section "Overview" dans *GNU Texinfo*). Le titre et le corps sont tous deux une liste de tuples de balises/messages de langue, ce qui permet à `guix pull` d'afficher les nouvelles dans la langue qui correspond à la locale de l'utilisateur.

Si vous souhaitez traduire des actualités en utilisant un déroulement basé sur `gettext`, vous pouvez extraire des chaînes traduisibles avec `xgettext` (voir Section "xgettext Invocation" dans *GNU Gettext Utilities*). Par exemple, en supposant que vous écriviez d'abord les entrées de nouvelles en anglais, la commande ci-dessous crée un fichier PO contenant les chaînes à traduire :

```
xgettext -o news.po -l scheme -ken etc/news.txt
```

Pour résumer, oui, vous pourriez utiliser votre chaîne comme un blog. Mais attention, ce n'est *pas tout à fait* ce à quoi vos utilisateur-ric-e-s pourraient s'attendre.

## 8 Développement

Si vous êtes développeur de logiciels, Guix fournit des outils que vous devriez trouver utiles — indépendamment du langage dans lequel vous développez. C’est ce dont parle ce chapitre.

La commande `guix shell` permet de créer des environnements logiciels à usage unique, que ce soit pour le développement ou pour lancer une commande sans l’installer dans votre profil. La commande `guix pack` vous permet de créer des *lots applicatifs* qui peuvent facilement être distribués à des utilisateurs qui n’utilisent pas Guix.

### 8.1 Invoquer `guix shell`

Le but de `guix shell` est de faciliter la création d’environnements logiciels à usage unique, sans changer votre profil. Vous l’utiliserez typiquement pour créer des environnement de développement ; c’est aussi une commande pratique pour lancer des applications sans « polluer » votre profil.

**Remarque:** La commande `guix shell` a été introduite récemment pour remplacer `guix environment` (voir Section 8.2 [Invoquer `guix environment`], page 90). Si vous connaissez déjà `guix environment`, vous remarquerez qu’elle est similaire mais aussi (on l’espère) plus pratique.

La syntaxe générale est :

```
guix shell [options] [paquet...]
```

L’exemple suivant crée un environnement contenant Python et NumPy, en construisant ou en téléchargeant les paquets manquants et lance la commande `python3` dans cet environnement :

```
guix shell python python-numpy -- python3
```

Vous pouvez créer des environnements de développement comme dans l’exemple ci-dessous, qui ouvre un shell interactif contenant toutes les dépendance et les variables d’environnement requises pour travailler sur Inkscape :

```
guix shell --development inkscape
```

Sortir du shell vous replacera dans l’environnement de départ, avant d’avoir invoqué la commande `guix shell`. Au prochain lancement du ramasse-miettes (voir Section 6.5 [Invoquer `guix gc`], page 60), les paquets installés dans l’environnement et qui ne sont plus utilisés en dehors seront possiblement supprimés.

En plus, `guix shell` essaiera de faire ce que vous espérez quand il est invoqué de manière interactive et sans arguments, comme ceci :

```
guix shell
```

S’il trouve un `manifest.scm` dans le répertoire de travail actuel ou dans l’un de ses parents, il utilisera ce manifeste comme s’il avait été passé avec `--manifest`. De la même manière, s’il trouve un `guix.scm` dans ces mêmes répertoire, il l’utilisera pour construire un profil de développement comme si `--development` et `--file` étaient présents. Dans tous les cas, le fichier ne sera chargé que s’il réside dans un répertoire listé dans `~/config/guix/shell-authorized-directories`. Cela permet de facilement définir, partager et entrer dans des environnements de développement.

Par défaut, la session shell ou la commande est lancée dans un environnement *augmenté*, où les nouveaux paquets sont ajoutés aux variables d’environnement de recherche comme

PATH. Vous pouvez à la place créer un environnement *isolé* ne contenant rien d'autre que les paquets que vous avez demandé. Passez l'option `--pure` pour nettoyer les définitions des variables d'environnement qui se trouvent dans l'environnement parent<sup>1</sup> ; en passant `--container` vous pouvez aller plus loin en démarrant un *conteneur* isolé du reste du système :

```
guix shell --container emacs gcc-toolchain
```

La commande ci-dessus démarre un shell interactif dans un conteneur avec rien d'autre que `emacs`, `gcc-toolchain` et leurs dépendances. Le conteneur n'a pas d'accès au réseau et ne partage pas les fichiers autres que ceux du répertoire de travail avec son environnement. C'est utile pour éviter d'accéder à des ressources systèmes comme `/usr/bin` sur les distributions externes.

Cette option `--container` peut aussi être utile si vous voulez lancer une application sensible en terme de sécurité, comme un navigateur web, dans un environnement isolé. Par exemple, la commande suivante lance Ungogled-Chromium dans un environnement isolé, cette fois en partageant l'accès réseau de l'hôte et en préservant sa variable d'environnement `DISPLAY`, mais sans partager même le répertoire actuel :

```
guix shell --container --network --no-cwd ungoogled-chromium \
  --preserve='^DISPLAY$' -- chromium
```

`guix shell` définit la variable `GUIX_ENVIRONMENT` dans le shell qu'il crée ; sa valeur est le nom de fichier du profil de cet environnement. Cela permet aux utilisatrice-eur-s, disons, de définir un prompt spécifique pour les environnement de développement dans leur `.bashrc` (voir Section “Bash Startup Files” dans *The GNU Bash Reference Manual*) :

```
if [ -n "$GUIX_ENVIRONMENT" ]
then
  export PS1="\u@\h \w [dev]\$ "
fi
```

... ou de naviguer dans le profil :

```
$ ls "$GUIX_ENVIRONMENT/bin"
```

Les options disponibles sont résumées ci-dessous.

`--check` Met en place l'environnement et vérifie si le shell peut écraser les variables d'environnement. C'est une bonne idée d'utiliser cette option la première fois que vous lancez `guix shell` pour une session interactive pour vous assurer que votre configuration est correcte.

Par exemple, si le shell modifie la variable d'environnement `PATH`, cela sera rapporté car vous auriez un environnement différent de celui que vous avez demandé.

Ces problèmes indiquent en général que les fichiers de démarrage du shell modifient ces variables d'environnement de manière inattendue. Par exemple, si vous utilisez Bash, assurez-vous que les variables d'environnement sont définies ou modifiées dans `~/.bash_profile` et non dans `~/.bashrc` — ce premier est sourcé seulement par les shells de connexion. Voir Section “Bash Startup Files”

<sup>1</sup> Assurez-vous d'utiliser l'option `--check` la première fois que vous utilisez `guix shell` de manière interactive pour vous assurer que le shell ne défait pas les effets de `--pure`.

dans *le manuel de référence de Bash* pour plus de détails sur les fichiers de démarrage de Bash.

**--development**

**-D** Fait en sorte que `guix shell` ajoute à l'environnement les dépendances du paquet suivant au lieu du paquet lui-même. Vous pouvez combiner cette option avec d'autres paquets. Par exemple, la commande suivante démarre un shell interactif contenant les dépendances à la construction de GNU Guile, plus Autoconf, Automake et Libtool :

```
guix shell -D guile autoconf automake libtool
```

**--expression=expr**

**-e expr** Crée un environnement pour le paquet ou la liste de paquets en lesquels s'évalue *expr*.

Par exemple, lancer :

```
guix shell -D -e '(@ (gnu packages maths) petsc-openmpi)'
```

démarre un shell avec l'environnement pour cette variante spécifique du paquet PETSc.

Lancer :

```
guix shell -e '(@ (gnu) %base-packages)'
```

démarre un shell où tous les paquets de base du système sont disponibles.

Les commande au-dessus n'utilisent que les sorties par défaut des paquets donnés. Pour choisir d'autres sorties, on peut spécifier des pairs :

```
guix shell -e '(list (@ (gnu packages bash) bash) "include")'
```

Voir [package-development-manifest], page 123, pour apprendre à écrire un fichier manifeste ou l'environnement de développement d'un paquet.

**--file=fichier**

**-f fichier**

Crée un environnement contenant le paquet ou la liste de paquets en lesquels *fichier* s'évalue.

Par exemple, *fichier* peut contenir une définition comme celle-ci (voir Section 9.2 [Définition des paquets], page 105) :

```
(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))))
```



Avec le fichier ci-dessus, vous pouvez entrer un environnement de développement pour GDB en lançant :

```
guix shell -D -f gdb-devel.scm
```

**--manifest=fichier**

**-m fichier**

Crée un environnement pour les paquets contenus dans l'objet manifeste renvoyé par le code Scheme dans *fichier*. Vous pouvez répéter cette option plusieurs fois, auquel cas les manifestes sont concaténés.

C'est similaire à l'option de même nom de **guix package** (voir [profile-manifest], page 48) et utilise les mêmes fichiers manifestes.

Voir Section 9.4 [Écrire un manifeste], page 120, pour des informations sur l'écriture d'un manifeste. Voir **--export-manifest** ci-dessous pour apprendre à obtenir un premier manifeste.

**--export-manifest**

Écrire un manifeste vers la sortie standard utilisable avec **--manifest** et qui correspond aux options de la ligne de commande données.

C'est une manière de « convertir » les arguments de la ligne de commande en un manifeste. Par exemple, imaginez que vous en ayez marre de taper de longues lignes et souhaitez avoir un manifeste équivalent à cette ligne de commande :

```
guix shell -D guile git emacs emacs-geiser emacs-geiser-guile
```

Ajoutez l'option **--export-manifest** à la ligne de commande ci-dessus :

```
guix shell --export-manifest \
  -D guile git emacs emacs-geiser emacs-geiser-guile
```

... et vous obtiendrez un manifeste comme ceci :

```
(concatenate-manifests
 (list (specifications->manifest
       (list "git"
             "emacs"
             "emacs-geiser"
             "emacs-geiser-guile")))
 (package->development-manifest
  (specification->package "guile"))))
```

Vous pouvez l'enregistrer dans un fichier, disons *manifest.scm*, et ensuite le passer à **guix shell** ou n'importe quelle commande **guix** :

```
guix shell -m manifest.scm
```

Voilà, vous avez convertit une longue ligne de commande en un manifeste ! Ce processus de conversion prend en compte les options de transformation des paquets (voir Section 10.1.2 [Options de transformation de paquets], page 183) et ne devrait donc pas perdre d'information.

**--profile=profil**

**-p profil** Crée un environnement contenant les paquets installés dans *profil*. Utilisez **guix package** (voir Section 6.2 [Invoquer guix package], page 44) pour créer et gérer les profils.

**--pure** Réinitialisation des variables d'environnement existantes lors de la construction du nouvel environnement, sauf celles spécifiées avec l'option **--preserve**. (voir ci-dessous). Cela a pour effet de créer un environnement dans lequel les chemins de recherche ne contiennent que des entrées de paquets.

**--preserve=regex**

**-E regex** Lorsque vous utilisez **--pure**, préserver les variables d'environnement qui correspondent à *regex* — en d'autres termes, cela les met en « liste blanche » de variables d'environnement qui doivent être préservées. Cette option peut être répétée plusieurs fois.

```
guix shell --pure --preserve=~SLURM openmpi ... \
  -- mpirun ...
```

Cet exemple exécute `mpirun` dans un contexte où les seules variables d'environnement définies sont `PATH`, les variables d'environnement dont le nom commence par 'SLURM', ainsi que les variables « importantes » habituelles (`HOME`, `USER`, etc.).

**--search-paths**

Affiche les définitions des variables d'environnement qui composent l'environnement.

**--system=système**

**-s système**

Essaye de construire pour *système* — p. ex. `i686-linux`.

**--container**

**-C** Lance *commande* dans un conteneur isolé. Le répertoire de travail actuel en dehors du conteneur est monté dans le conteneur. En plus, à moins de le changer avec **--user**, un répertoire personnel fictif est créé pour correspondre à celui de l'utilisateur·rice actuel·le et `/etc/passwd` est configuré en conséquence.

Le processus de création s'exécute en tant qu'utilisateur·rice actuel·le à l'extérieur du conteneur. À l'intérieur du conteneur, il possède les mêmes UID et GID que l'utilisateur·rice actuel·le, sauf si l'option **--user** est passée (voir ci-dessous).

**--network**

**-N** Pour les conteneurs, partage l'espace de nom du réseau avec le système hôte. Les conteneurs créés sans cette option n'ont accès qu'à l'interface de boucle locale.

**--link-profile**

**-P** Pour les conteneurs, liez le profil d'environnement à `~/guix-profile` à l'intérieur du conteneur et définissez `GUIX_ENVIRONNEMENT` à cet endroit. Cela équivaut à faire de `~/guix-profile` un lien symbolique vers le profil réel à l'intérieur du conteneur. La liaison échouera et interrompra l'environnement si le répertoire existe déjà, ce qui sera certainement le cas si `guix shell` a été invoqué dans le répertoire personnel de l'utilisateur·rice.

Certains paquets sont configurés pour chercher des fichiers de configuration et des données dans `~/.guix-profile` ;<sup>2</sup> ; `--link-profile` permet à ces programmes de se comporter comme attendu dans l'environnement.

`--user=utilisateur`

`-u utilisateur`

Pour les conteneurs, utilise le nom d'utilisateur *utilisateur* à la place de l'utilisateur actuel. L'entrée générée dans `/etc/passwd` dans le conteneur contiendra le nom *utilisateur* ; le répertoire personnel sera `/home/utilisateur` ; et aucune donnée GECOS ne sera copiée. En plus, l'UID et le GID dans le conteneur seront 1000. *user* n'a pas besoin d'exister sur le système.

En plus, tous les chemins partagés ou exposés (voir `--share` et `--expose` respectivement) dont la cible est dans le répertoire personnel de l'utilisateur-*rice* seront remontés relativement à `/home/USER` ; cela comprend le montage automatique du répertoire de travail actuel.

```
# exposera les chemins comme /home/toto/wd, /home/toto/test et /home/toto/ta
cd $HOME/wd
guix shell --container --user=toto \
  --expose=$HOME/test \
  --expose=/tmp/target=$HOME/target
```

Bien que cela limite la fuite de l'identité de l'utilisateur à travers le chemin du répertoire personnel et des champs de l'utilisateur, ce n'est qu'un composant utile pour une solution d'anonymisation ou de préservation de la vie privée — pas une solution en elle-même.

`--no-cwd` Pour les conteneurs, le comportement par défaut est de partager le répertoire de travail actuel avec le conteneur isolé et de passer immédiatement à ce répertoire à l'intérieur du conteneur. Si cela n'est pas souhaitable, `--no-cwd` fera en sorte que le répertoire de travail courant soit automatiquement partagé *not* et passera au répertoire personnel de l'utilisateur-*rice* dans le conteneur à la place. Voir aussi `--user`.

`--expose=source[=cible]`

`--share=source[=cible]`

Pour les conteneurs, `--expose` (resp. `--share`) expose le système de fichiers *source* du système hôte comme un système de fichiers *target* en lecture seule (resp. en lecture-écriture) dans le conteneur. Si *target* n'est pas spécifiée, *source* est utilisé comme point de montage dans le conteneur.

L'exemple ci-dessous crée un REPL Guile dans un conteneur dans lequel le répertoire personnel de l'utilisateur est accessible en lecture-seule via le répertoire `/exchange` :

```
guix shell --container --expose=$HOME=/exchange guile -- guile
```

`--symlink=spec`

`-S spec` Pour les conteneurs, crée le lien symbolique spécifié par la *spec*, documenté dans `[pack-symlink-option]`, page 100.

<sup>2</sup> Par exemple, le paquet `fontconfig` inspecte `~/.guix-profile/share/fonts` pour trouver des polices supplémentaires.

**--emulate-fhs**

**-F** Lorsqu'elle est utilisée avec `--container`, cette option émule une configuration qui respecte le standard de la hiérarchie des systèmes de fichiers (FHS) (<https://refspecs.linuxfoundation.org/fhs.shtml>) dans le conteneur, en fournissant `/bin`, `/lib` et d'autres répertoires et fichiers spécifiés par le FHS.

Comme Guix ne respecte pas les spécifications du FHS, cette option configure le conteneur pour mieux reproduire le comportement des autres distributions GNU/Linux. C'est utile pour reproduire d'autres environnements de développement, tester et utiliser des programmes qui s'attendent à ce que les spécifications du FHS soient suivies. Avec cette option, le conteneur inclura une version de glibc qui lira `/etc/ld.so.cache` dans le conteneur pour trouver le cache des bibliothèques partagées (au contraire de glibc dans l'utilisation normale de Guix) et configurera les répertoires attendu du FHS : `/bin`, `/etc`, `/lib`, `/usr` à partir du profil du conteneur.

**--rebuild-cache**

La plupart du temps, `guix shell` met en cache l'environnement pour que les utilisations suivantes soient instantanées. Les entrées du cache utilisées le moins récemment sont régulièrement supprimées. Le cache est aussi invalidé lorsque vous utilisez `--file` ou `--manifest` à chaque fois que le fichier correspondant est modifié.

L'option `--rebuild-cache` force l'environnement en cache à être rafraîchi. C'est utile si vous utilisez les options `--file` ou `--manifest` et que les fichiers `guix.scm` ou `manifest.scm` ont des dépendances externes, ou si leur comportement dépend, disons, de variables d'environnements.

**--root=fichier****-r fichier**

Fait de *fichier* un lien symbolique vers le profil de cet environnement, et l'enregistre comme une racine du ramasse-miettes.

C'est utile si vous souhaitez protéger votre environnement du ramasse-miettes, pour le rendre « persistant ».

Lorsque vous n'utilisez pas cette option, `guix shell` met en cache les profils pour les utilisations suivantes du même environnement soient immédiates — vous pouvez comparer cela à l'utilisation de `--root` sauf que `guix shell` prend soin de régulièrement supprimer les racines du ramasse-miettes utilisées le moins récemment.

Dans certains cas, `guix shell` ne met pas les profils en cache – p. ex. si des options de transformations comme `--with-latest` sont utilisées. Dans ce cas, l'environnement n'est protégé du ramasse-miettes que le temps de la session `guix shell`. Cela signifie que la prochaine fois que vous créez le même environnement, vous pourriez avoir à reconstruire ou télécharger des paquets.

Voir Section 6.5 [Invoquer `guix gc`], page 60, pour en apprendre plus sur les racines du ramasse-miettes.

En plus, `guix shell` prend en charge toutes les options de construction communes prises en charge par `guix build` (voir Section 10.1.1 [Options de construction communes]),

page 180) et toutes les options de transformation de paquets (voir Section 10.1.2 [Options de transformation de paquets], page 183).

## 8.2 Invoquer guix environment

Le but de `guix environment` est de vous aider à créer des environnements de développement.

**Avertissement d’obsolescence:** La commande `guix environment` est obsolète et remplacée par `guix shell`, qui effectue les mêmes tâches mais est plus pratique à utiliser. Voir Section 8.1 [Invoquer guix shell], page 83.

Étant obsolète, `guix environment` sera supprimée à un moment, mais le projet Guix s’engage à la garder jusqu’au premier mai 2023. Contactez-nous sur `guix-devel@gnu.org` si vous voulez en parler.

La syntaxe générale est :

```
guix environment options paquet...
```

L’exemple suivant crée un nouveau shell préparé pour le développement de GNU Guile :

```
guix environment guile
```

Si les dépendances requises ne sont pas déjà construites, `guix environment` les construit automatiquement. L’environnement du nouveau shell est une version améliorée de l’environnement dans lequel `guix environment` a été lancé. Il contient les chemins de recherche nécessaires à la construction du paquet donné en plus des variables d’environnement existantes. Pour créer un environnement « pur », dans lequel les variables d’environnement de départ ont été nettoyées, utilisez l’option `--pure`<sup>3</sup>.

Sortir d’un environnement Guix est comme sortir du shell, et cela vous replacera dans l’ancien environnement avant d’avoir invoqué la commande `guix environment`. Au prochain lancement du ramasse-miettes (voir Section 6.5 [Invoquer guix gc], page 60), les paquets installés pour l’environnement seront supprimés et ne seront plus disponibles en dehors de celui-ci.

`guix environment` définit la variable `GUIX_ENVIRONMENT` dans le shell qu’il crée ; sa valeur est le nom de fichier du profil de cet environnement. Cela permet aux utilisatrice-eur-s, disons, de définir un prompt spécifique pour les environnements de développement dans leur `.bashrc` (voir Section “Bash Startup Files” dans *The GNU Bash Reference Manual*) :

```
if [ -n "$GUIX_ENVIRONMENT" ]
then
  export PS1="\u@\h \w [dev]\$ "
fi
```

... ou de naviguer dans le profil :

```
$ ls "$GUIX_ENVIRONMENT/bin"
```

<sup>3</sup> Les utilisatrice-eur-s ajoutent parfois à tort des valeurs supplémentaires dans les variables comme `PATH` dans leur `~/bashrc`. En conséquence, lorsque `guix environment` le lance, Bash peut lire `~/bashrc`, ce qui produit des « impuretés » dans ces variables d’environnement. C’est une erreur de définir ces variables d’environnement dans `.bashrc` ; à la place, elles devraient être définies dans `.bash_profile`, qui est sourcé uniquement par les shells de connexion. Voir Section “Bash Startup Files” dans *The GNU Bash Reference Manual*, pour des détails sur les fichiers de démarrage de Bash.

De surcroît, plus d'un paquet peut être spécifié, auquel cas l'union des entrées des paquets données est utilisée. Par exemple, la commande ci-dessous crée un shell où toutes les dépendances de Guile et Emacs sont disponibles :

```
guix environment guile emacs
```

Parfois, une session shell interactive est inutile. On peut invoquer une commande arbitraire en plaçant le jeton `--` pour séparer la commande du reste des arguments :

```
guix environment guile -- make -j4
```

Dans d'autres situations, il est plus pratique de spécifier la liste des paquets requis dans l'environnement. Par exemple, la commande suivante lance `python` dans un environnement contenant Python 3 et NumPy :

```
guix environment --ad-hoc python-numpy python -- python3
```

En plus, on peut vouloir les dépendance d'un paquet et aussi des paquets supplémentaires qui ne sont pas des dépendances à l'exécution ou à la construction, mais qui sont utiles au développement tout de même. À cause de cela, le tag `--ad-hoc` est positionnel. Les paquets qui apparaissent avant `--ad-hoc` sont interprétés comme les paquets dont les dépendances seront ajoutées à l'environnement. Les paquets qui apparaissent après `--ad-hoc` sont interprétés comme les paquets à ajouter à l'environnement directement. Par exemple, la commande suivante crée un environnement de développement pour Guix avec les paquets Git et `strace` en plus :

```
guix environment --pure guix --ad-hoc git strace
```

Parfois il est souhaitable d'isoler l'environnement le plus possible, pour une pureté et une reproductibilité maximale. En particulier, lorsque vous utilisez Guix sur une distribution hôte qui n'est pas le système Guix, il est souhaitable d'éviter l'accès à `/usr/bin` et d'autres ressources du système depuis les environnements de développement. Par exemple, la commande suivante crée un REPL Guile dans un « conteneur » où seuls le dépôt et le répertoire de travail actuel sont montés :

```
guix environment --ad-hoc --container guile -- guile
```

**Remarque:** L'option `--container` requiert Linux-libre 3.19 ou supérieur.

Un autre cas d'usage typique pour les conteneurs est de lancer des applications sensibles à la sécurité, comme un navigateur web. Pour faire fonctionner Eolie, nous devons exposer et partager certains fichiers et répertoires ; nous incluons `nss-certs` et exposons `/etc/ssl/certs/` pour l'authentification HTTPS ; enfin, nous préservons la variable d'environnement `DISPLAY` puisque les applications graphiques conteneurisées ne s'afficheront pas sans elle.

```
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
```

Les options disponibles sont résumées ci-dessous.

`--check` Configure l'environnement et vérifie si le shell écraserait les variables d'environnement. Voir Section 8.1 [Invoquer guix shell], page 83, pour plus d'informations.

`--root=fichier`

`-r fichier`

Fait de *fichier* un lien symbolique vers le profil de cet environnement, et l'enregistre comme une racine du ramasse-miettes.

C'est utile si vous souhaitez protéger votre environnement du ramasse-miettes, pour le rendre « persistant ».

Lorsque cette option est omise, l'environnement n'est protégé du ramasse-miettes que le temps de la session `guix environment`. Cela signifie que la prochaine fois que vous créez le même environnement, vous pourriez avoir à reconstruire ou télécharger des paquets. Voir Section 6.5 [Invoquer `guix gc`], page 60, pour plus d'informations sur les racines du GC.

`--expression=expr`

`-e expr` Crée un environnement pour le paquet ou la liste de paquets en lesquels s'évalue *expr*.

Par exemple, lancer :

```
guix environment -e '(@ (gnu packages maths) petsc-openmpi)'
```

démarré un shell avec l'environnement pour cette variante spécifique du paquet PETSc.

Lancer :

```
guix environment --ad-hoc -e '(@ (gnu) %base-packages)'
```

démarré un shell où tous les paquets de base du système sont disponibles.

Les commandes au-dessus n'utilisent que les sorties par défaut des paquets donnés. Pour choisir d'autres sorties, on peut spécifier des paires :

```
guix environment --ad-hoc -e '(list (@ (gnu packages bash) bash) "include")'
```

`--load=fichier`

`-l fichier`

Crée un environnement pour le paquet ou la liste de paquets en lesquels *fichier* s'évalue.

Par exemple, *fichier* peut contenir une définition comme celle-ci (voir Section 9.2 [Définition des paquets], page 105) :

```
(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=fichier`

`-m fichier`

Crée un environnement pour les paquets contenus dans l'objet manifeste renvoyé par le code Scheme dans *fichier*. Vous pouvez répéter cette option plusieurs fois, auquel cas les manifestes sont concaténés.

C'est similaire à l'option de même nom de `guix package` (voir [profile-manifest], page 48) et utilise les mêmes fichiers manifestes.

Voir [shell-export-manifest], page 86, pour des informations sur la « conversion » des options de la ligne de commande en un manifeste.

`--ad-hoc` Inclut tous les paquets spécifiés dans l'environnement qui en résulte, comme si un paquet *ad hoc* était spécifié, avec ces paquets comme entrées. Cette option est utile pour créer un environnement rapidement sans avoir à écrire une expression de paquet contenant les entrées désirées.

Par exemple la commande :

```
guix environment --ad-hoc guile guile-sdl -- guile
```

lance `guile` dans un environnement où Guile et Guile-SDDL sont disponibles.

Remarquez que cet exemple demande implicitement la sortie par défaut de `guile` et `guile-sdl`, mais il est possible de demander une sortie spécifique — p. ex. `glib:bin` demande la sortie `bin` de `glib` (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59).

Cette option peut être composée avec le comportement par défaut de `guix environment`. Les paquets qui apparaissent avant `--ad-hoc` sont interprétés comme les paquets dont les dépendances seront ajoutées à l'environnement, le comportement par défaut. Les paquets qui apparaissent après `--ad-hoc` sont interprétés comme les paquets à ajouter à l'environnement directement.

`--profile=profil`

`-p profil` Crée un environnement contenant les paquets installés dans *profil*. Utilisez `guix package` (voir Section 6.2 [Invoquer guix package], page 44) pour créer et gérer les profils.

`--pure` Réinitialisation des variables d'environnement existantes lors de la construction du nouvel environnement, sauf celles spécifiées avec l'option `--preserve`. (voir ci-dessous). Cela a pour effet de créer un environnement dans lequel les chemins de recherche ne contiennent que des entrées de paquets.

`--preserve=regex`

`-E regex` Lorsque vous utilisez `--pure`, préserver les variables d'environnement qui correspondent à *regex* — en d'autres termes, cela les met en « liste blanche » de variables d'environnement qui doivent être préservées. Cette option peut être répétée plusieurs fois.

```
guix environment --pure --preserve=~SLURM --ad-hoc openmpi ... \
-- mpirun ...
```

Cet exemple exécute `mpirun` dans un contexte où les seules variables d'environnement définies sont `PATH`, les variables d'environnement dont le nom commence par 'SLURM', ainsi que les variables « importantes » habituelles (`HOME`, `USER`, etc.).



**--search-paths**  
Affiche les définitions des variables d'environnement qui composent l'environnement.

**--system=systeme**

**-s systeme**

Essaye de construire pour *systeme* — p. ex. *i686-linux*.

**--container**

**-C** Lance *commande* dans un conteneur isolé. Le répertoire de travail actuel en dehors du conteneur est monté dans le conteneur. En plus, à moins de le changer avec **--user**, un répertoire personnel fictif est créé pour correspondre à celui de l'utilisateur-*rice* actuel·le et */etc/passwd* est configuré en conséquence.

Le processus de création s'exécute en tant qu'utilisateur-*rice* actuel·le à l'extérieur du conteneur. À l'intérieur du conteneur, il possède les mêmes UID et GID que l'utilisateur-*rice* actuel·le, sauf si l'option **--user** est passée (voir ci-dessous).

**--network**

**-N** Pour les conteneurs, partage l'espace de nom du réseau avec le système hôte. Les conteneurs créés sans cette option n'ont accès qu'à l'interface de boucle locale.

**--link-profile**

**-P** Pour les conteneurs, liez le profil d'environnement à *~/.guix-profile* à l'intérieur du conteneur et définissez **GUIX\_ENVIRONNEMENT** à cet endroit. Cela équivaut à faire de *~/.guix-profile* un lien symbolique vers le profil réel à l'intérieur du conteneur. La liaison échouera et interrompra l'environnement si le répertoire existe déjà, ce qui sera certainement le cas si *guix environment* a été invoqué dans le répertoire personnel de l'utilisateur-*rice*.

Certains paquets sont configurés pour chercher des fichiers de configuration et des données dans *~/.guix-profile* ;<sup>4</sup> ; **--link-profile** permet à ces programmes de se comporter comme attendu dans l'environnement.

**--user=utilisateur**

**-u utilisateur**

Pour les conteneurs, utilise le nom d'utilisateur *utilisateur* à la place de l'utilisateur actuel. L'entrée générée dans */etc/passwd* dans le conteneur contiendra le nom *utilisateur* ; le répertoire personnel sera */home/utilisateur* ; et aucune donnée GECOS ne sera copiée. En plus, l'UID et le GID dans le conteneur seront 1000. *user* n'a pas besoin d'exister sur le système.

En plus, tous les chemins partagés ou exposés (voir **--share** et **--expose** respectivement) dont la cible est dans le répertoire personnel de l'utilisateur-*rice* seront remontés relativement à */home/USER* ; cela comprend le montage automatique du répertoire de travail actuel.

# exposera les chemins comme */home/toto/wd*, */home/toto/test* et */home/toto/ta*

<sup>4</sup> Par exemple, le paquet *fontconfig* inspecte *~/.guix-profile/share/fonts* pour trouver des polices supplémentaires.

```
cd $HOME/wd
guix environment --container --user=toto \
  --expose=$HOME/test \
  --expose=/tmp/target=$HOME/target
```

Bien que cela limite la fuite de l'identité de l'utilisateur à travers le chemin du répertoire personnel et des champs de l'utilisateur, ce n'est qu'un composant utile pour une solution d'anonymisation ou de préservation de la vie privée — pas une solution en elle-même.

**--no-cwd** Pour les conteneurs, le comportement par défaut est de partager le répertoire de travail actuel avec le conteneur isolé et de passer immédiatement à ce répertoire à l'intérieur du conteneur. Si cela n'est pas souhaitable, **--no-cwd** fera en sorte que le répertoire de travail courant soit automatiquement partagé *not* et passera au répertoire personnel de l'utilisateur-*rice* dans le conteneur à la place. Voir aussi **--user**.

**--expose=source[=cible]**

**--share=source[=cible]**

Pour les conteneurs, **--expose** (resp. **--share**) expose le système de fichiers *source* du système hôte comme un système de fichiers *target* en lecture seule (resp. en lecture-écriture) dans le conteneur. Si *target* n'est pas spécifiée, *source* est utilisé comme point de montage dans le conteneur.

L'exemple ci-dessous crée un REPL Guile dans un conteneur dans lequel le répertoire personnel de l'utilisateur est accessible en lecture-seule via le répertoire `/exchange` :

```
guix environment --container --expose=$HOME=/exchange --ad-hoc guile -- guil
```

**--emulate-fhs**

**-F** Lorsqu'elle est utilisée avec **--container**, cette option émule une configuration qui respecte le standard de la hiérarchie des systèmes de fichiers (FHS) (<https://refspecs.linuxfoundation.org/fhs.shtml>) dans le conteneur, en fournissant `/bin`, `/lib` et d'autres répertoires et fichiers spécifiés par le FHS. Comme Guix ne respecte pas les spécifications du FHS, cette option configure le conteneur pour mieux reproduire le comportement des autres distributions GNU/Linux. C'est utile pour reproduire d'autres environnements de développement, tester et utiliser des programmes qui s'attendent à ce que les spécifications du FHS soient suivies. Avec cette option, le conteneur inclura une version de `glibc` qui lira `/etc/ld.so.cache` dans le conteneur pour trouver le cache des bibliothèques partagées (au contraire de `glibc` dans l'utilisation normale de Guix) et configurera les répertoires attendu du FHS : `/bin`, `/etc`, `/lib`, `/usr` à partir du profil du conteneur.

En plus, `guix environment` prend en charge toutes les options de construction communes prises en charge par `guix build` (voir Section 10.1.1 [Options de construction communes], page 180) et toutes les options de transformation de paquets (voir Section 10.1.2 [Options de transformation de paquets], page 183).

### 8.3 Invoquer guix pack

Parfois vous voulez passer un logiciel à des gens qui n'ont pas (encore !) la chance d'utiliser Guix. Vous leur diriez bien de lancer `guix package -i quelque chose` mais ce n'est pas possible dans ce cas. C'est là que `guix pack` entre en jeu.

**Remarque:** Si vous cherchez comment échanger des binaires entre des machines où Guix est déjà installé, voir Section 10.13 [Invoquer guix copy], page 229, Section 10.11 [Invoquer guix publish], page 222, et Section 6.10 [Invoquer guix archive], page 71.

La commande `guix pack` crée un *pack* ou *lot de logiciels* : elle crée une archive tar ou un autre type d'archive contenant les binaires pour le logiciel qui vous intéresse ainsi que ses dépendances. L'archive qui en résulte peut être utilisée sur toutes les machines qui n'ont pas Guix et les gens peuvent lancer exactement les mêmes binaires que ceux que vous avez avec Guix. Le pack lui-même est créé d'une manière reproductible au bit près, pour que n'importe qui puisse vérifier qu'il contient bien les résultats que vous prétendez proposer.

Par exemple, pour créer un lot contenant Guile, Emacs, Geiser et toutes leurs dépendances, vous pouvez lancer :

```
$ guix pack guile emacs emacs-geiser
...
/gnu/store/...-pack.tar.gz
```

Le résultat ici est une archive tar contenant un répertoire `/gnu/store` avec tous les paquets nécessaires. L'archive qui en résulte contient un *profil* avec les trois paquets qui vous intéressent ; le profil est le même que celui qui aurait été créé avec `guix package -i`. C'est ce mécanisme qui est utilisé pour créer les archives tar binaires indépendantes de Guix (voir Section 2.1 [Installation binaire], page 5).

Les utilisateurs de ce pack devraient lancer `/gnu/store/...-profile/bin/guile` pour lancer Guile, ce qui n'est pas très pratique. Pour éviter cela, vous pouvez créer, disons, un lien symbolique `/opt/gnu/bin` vers le profil :

```
guix pack -S /opt/gnu/bin=bin guile emacs emacs-geiser
```

De cette façon, les utilisateurs peuvent joyeusement taper `/opt/gnu/bin/guile` et profiter.

Et si le destinataire de votre pack n'a pas les privilèges root sur sa machine, et ne peut donc pas le décompresser dans le système de fichiers root ? Dans ce cas, vous pourriez utiliser l'option `--relocatable` (voir plus bas). Cette option produit des *binaires repositionnables*, ce qui signifie qu'ils peuvent être placés n'importe où dans l'arborescence du système de fichiers : dans l'exemple au-dessus, les utilisateur-riche-s peuvent décompresser votre archive dans leur répertoire personnel et lancer directement `./opt/gnu/bin/guile`.

Autrement, vous pouvez produire un pack au format d'image Docker avec la commande suivante :

```
guix pack -f docker -S /bin=bin guile guile-readline
```

Le résultat est une archive tar qui peut être passée à la commande `docker load`, puis à `docker run` :

```
docker load < file
docker run -ti guile-guile-readline /bin/guile
```

où *fichier* est l'image renvoyée par *guix pack* et *guile-guile-readline* est son « tag d'image ». Voir la documentation de Docker (<https://docs.docker.com/engine/reference/commandline/load/>) pour plus d'informations.

Autrement, vous pouvez produire une image SquashFS avec la commande suivante :

```
guix pack -f squashfs bash guile emacs emacs-geiser
```

Le résultat est une image de système de fichiers SquashFS qui peut soit être montée directement soit être utilisée comme image de conteneur de système de fichiers avec l'Singularity container execution environment (<https://singularity.lbl.gov>), avec des commandes comme *singularity shell* ou *singularity exec*.

Diverses options en ligne de commande vous permettent de personnaliser votre pack :

`--format=format`

`-f format` Produire un pack dans le *format* donné.

Les formats disponibles sont :

**tarball** C'est le format par défaut. Il produit une archive tar contenant tous les binaires et les liens symboliques spécifiés.

**docker** Cela produit une archive tar qui suit la spécification des images Docker (<https://github.com/docker/docker/blob/master/image/spec/v1.2.md>). Le « nom de dépôt » qui apparaît dans la sortie de la commande `docker images` est calculé à partir des noms des paquets passés sur la ligne de commande ou dans le fichier manifeste.

**squashfs** Cela produit une image SquashFS contenant tous les binaires et liens symboliques spécifiés, ainsi que des points de montages vides pour les systèmes de fichiers virtuels comme procfs.

**Remarque:** Singularity *requiert* que vous fournissiez `/bin/sh` dans l'image. Pour cette raison, `guix pack -f squashfs` implique toujours `-S /bin=bin`. Ainsi, votre invocation `guix pack` doit toujours commencer par quelque chose comme :

```
guix pack -f squashfs bash ...
```

Si vous oubliez le paquet `bash` (ou similaire), `singularity run` et `singularity exec` vont échouer avec un message « no such file or directory » peu utile.

**deb** Cela produit une archive Debian (un paquet avec l'extension « `.deb` ») contenant tous les binaires et les liens symboliques spécifiés, qui peut être installée sur n'importe quelle distribution GNU(/Linux) basée sur dpkg. Vous trouverez des options avancées avec l'option `--help-deb-format`. Elles permettent d'insérer des fichiers control pour un contrôle plus fin, comme pour activer des trigger ou fournir un script de configuration pour mainteneur pour lancer du code de configuration arbitraire à l'installation.

```
guix pack -f deb -C xz -S /usr/bin/hello=bin/hello hello
```

**Remarque:** Comme les archives produites par `guix pack` contiennent une collection d'objets du dépôt et comme chaque paquet `dpkg` ne doit pas avoir de conflits de fichiers, en pratique cela signifie que vous ne pourrez sans doute pas installer plus d'une archive de ce genre sur un système donné.

**Attention:** `dpkg` prendra possession de tout fichier contenu dans le pack qu'il ne connaît *pas*. Il est peu avisé d'installer des fichiers `.deb` produits par Guix sur un système où `/gnu/store` est partagé avec un autre logiciel, comme une installation de Guix ou d'autres packs non `deb`.

`--relocatable`

`-R` Produire des *binaires repositionnables* — c.-à-d. des binaires que vous pouvez placer n'importe où dans l'arborescence du système de fichiers et les lancer à partir de là.

Lorsque vous passez cette option une fois, les binaires qui en résultent demandent le support des *espaces de nom utilisateurs* dans le noyau Linux ; lorsque vous la passez *deux fois*<sup>5</sup>, les binaires repositionnables utilisent `PRoot` si les espaces de noms ne sont pas utilisables, et ça fonctionne à peu près partout — voir plus bas pour comprendre les implications.

Par exemple, si vous créez un pack contenant `Bash` avec :

```
guix pack -RR -S /mybin=bin bash
```

... vous pouvez copier ce pack sur une machine qui n'a pas Guix et depuis votre répertoire personnel en tant qu'utilisateur non-privilégié, lancer :

```
tar xf pack.tar.gz
./mybin/sh
```

Dans ce shell, si vous tapez `ls /gnu/store`, vous remarquerez que `/gnu/store` apparaît et contient toutes les dépendances de `bash`, même si la machine n'a pas du tout de `/gnu/store` ! C'est sans doute la manière la plus simple de déployer du logiciel construit avec Guix sur une machine sans Guix.

**Remarque:** Par défaut ,les binaires repositionnables s'appuient sur les *espaces de noms utilisateurs* du noyau Linux, qui permet à des utilisateurs non-privilégiés d'effectuer des montages et de changer de racine. Les anciennes versions de Linux ne le supportait pas et certaines distributions GNU/Linux le désactive.

Pour produire des binaires repositionnables qui fonctionnent même sans espace de nom utilisatrice-eur, passez `--relocatable` ou `-R deux fois`. Dans ce cas, les binaires testeront la prise en charge des espaces de noms utilisatrice-eur-s et utiliseront `PRoot` s'ils ne sont pas pris en charge. Les moteurs d'exécution suivants sont pris en charge :

---

<sup>5</sup> Il y a une astuce pour s'en souvenir : on peut envisager `-RR`, qui ajoute le support `PRoot`, comme étant l'abréviation de « Réellement Repositionnable ». Pas mal, hein ?

**default** Essayez les espaces de noms utilisateur-*rice-s* et revenez à PRoot si les espaces de noms utilisateur-*rice-s* ne sont pas pris en charge (voir ci-dessous).

**performance**

Essayez les espaces de noms utilisateur-*rice-s* et revenez à Fakechroot si les espaces de noms utilisateur-*rice-s* ne sont pas pris en charge (voir ci-dessous).

**users** Lance le programme à travers les espaces de nom utilisateur-*rice* et échoue s'ils ne sont pas supportés.

**proot** Passez à travers PRoot. Le programme PRoot (<https://proot-me.github.io/>) fournit la prise en charge nécessaire pour la virtualisation du système de fichier. Il y parvient en utilisant l'appel système `ptrace` sur le programme courant. Cette approche a l'avantage de fonctionner sans demander de support spécial de la part du noyau, mais occasionne un coût supplémentaire en temps pour chaque appel système effectué.

**fakechroot**

Passez à travers Fakechroot. Fakechroot (<https://github.com/dex4er/fakechroot/>) virtualise les accès au système de fichier en interceptant les appels vers les fonctions de la bibliothèque C telles que `open`, `stat`, `exec`, et ainsi de suite. Contrairement à PRoot, il n'engendre que très peu de coûts généraux. Cependant, il ne fonctionne pas encore tout-à-fait : par exemple, certains accès au système de fichier effectués à partir de la bibliothèque C ne sont pas interceptés, et les accès au système de fichier effectués *via* les appels système directs ne sont pas non plus interceptés, conduisant à un comportement erratique.

Lors de l'exécution d'un programme complet, vous pouvez demander explicitement l'un des moteurs d'exécution énumérés ci-dessus en définissant en conséquence la variable d'environnement `GUIX_EXECUTION_ENGINE`.

**--entry-point=commande**

Utiliser *commande* comme *point d'entrée* du paquet résultant, si le format du paquet le supporte—actuellement `docker` et `squashfs` (Singularity) la supportent. *commande* doit être relatif au profil contenu dans le pack.

Le point d'entrée spécifie la commande que des outils comme `docker run` or `singularity run` lancent automatiquement par défaut. Par exemple, vous pouvez faire :

```
guix pack -f docker --entry-point=bin/guile guile
```

Le pack résultant peut être facilement chargé et `docker run` sans arguments supplémentaires engendrera `bin/guile` :

```
docker load -i pack.tar.gz
docker run image-id
```

`--expression=expr`

`-e expr` Considérer le paquet évalué par *expr*.

Cela a le but identique que l'option de même nom de `guix build` (voir Section 10.1.3 [Options de construction supplémentaires], page 188).

`--manifest=fichier`

`-m fichier`

Utiliser les paquets contenus dans l'objet manifeste renvoyé par le code Scheme dans *fichier*. Vous pouvez répéter cette option plusieurs fois, auquel cas les manifestes sont concaténés.

Elle a un but similaire à l'option de même nom dans `guix package` (voir [profile-manifest], page 48) et utilise les mêmes fichiers manifeste. Ils vous permettent de définir une collection de paquets une fois et de l'utiliser aussi bien pour créer des profils que pour créer des archives pour des machines qui n'ont pas Guix d'installé. Remarquez que vous pouvez spécifier *soit* un fichier manifeste, *soit* une liste de paquet, mais pas les deux.

Voir Section 9.4 [Écrire un manifeste], page 120, pour des informations sur l'écriture d'un manifeste. Voir [shell-export-manifest], page 86, pour des informations sur la « conversion » des options de la ligne de commande en un manifeste.

`--system=système`

`-s système`

Tenter de construire pour le *système* — p. ex. `i686-linux` — plutôt que pour le type de système de l'hôte de construction.

`--target=triplet`

Effectuer une compilation croisée pour *triplet* qui doit être un triplet GNU valide, comme `"aarch64-linux-gnu"` (voir Section "Specifying target triplets" dans *Autoconf*).

`--compression=outil`

`-C outil` Compresser l'archive résultante avec *outil* — l'un des outils parmi `gzip`, `zstd`, `bzip2`, `xz`, `lzip`, ou `none` pour aucune compression.

`--symlink=spec`

`-S spec` Ajouter les liens symboliques spécifiés par *spec* dans le pack. Cette option peut apparaître plusieurs fois.

*spec* a la forme `source=cible`, où *source* est le lien symbolique qui sera créé et *cible* est la cible du lien.

Par exemple, `-S /opt/gnu/bin=bin` crée un lien symbolique `/opt/gnu/bin` qui pointe vers le sous-répertoire `bin` du profil.

**--save-provenance**

Sauvegarder les informations de provenance des paquets passés sur la ligne de commande. Les informations de provenance contiennent l'URL et le commit des canaux utilisés (voir Chapitre 7 [Canaux], page 75).

Les informations de provenance sont sauvegardées dans le fichier `/gnu/store/...-profile/manifest` du pack, avec les métadonnées de paquets habituelles — le nom et la version de chaque paquet, leurs entrées propagées, etc. Ce sont des informations utiles pour le destinataire du pack, qui sait alors comment le pack a (normalement) été obtenu.

Cette option n'est pas activée par défaut car, comme l'horodatage, les informations de provenance ne contribuent en rien au processus de construction. En d'autres termes, il y a une infinité d'URL et d'ID de commit qui permettent d'obtenir le même pack. Enregistrer de telles métadonnées « silencieuses » dans la sortie casse donc éventuellement la propriété de reproductibilité au bit près.

**--root=fichier****-r fichier**

Fait de *fichier* un lien symbolique vers le résultat, et l'enregistre en tant que racine du ramasse-miettes.

**--localstatedir****--profile-name=nom**

Inclus le « répertoire d'état local », `/var/guix`, dans le lot qui en résulte, et notamment le profil `/var/guix/profiles/per-user/root/nom` — par défaut *nom* est `guix-profile`, ce qui correspond à `~root/.guix-profile`.

`/var/guix` contient la base de données du dépôt (voir Section 9.9 [Le dépôt], page 157) ainsi que les racines du ramasse-miettes (voir Section 6.5 [Invoquer `guix gc`], page 60). Le fournir dans le pack signifie que le dépôt est « complet » et gérable par Guix ; ne pas le fournir dans le pack signifie que le dépôt est « mort » : aucun élément ne peut être ajouté ni enlevé après l'extraction du pack.

Un cas d'utilisation est l'archive binaire indépendante de Guix (voir Section 2.1 [Installation binaire], page 5).

**--derivation**

**-d** Affiche le nom de la dérivation que le pack construit.

**--bootstrap**

Utiliser les programmes d'amorçage pour construire le pack. Cette option n'est utile que pour les personnes qui développent Guix.

En plus, `guix pack` supporte toutes les options de construction communes (voir Section 10.1.1 [Options de construction communes], page 180) et toutes les options de transformation de paquets (voir Section 10.1.2 [Options de transformation de paquets], page 183).

## 8.4 La chaîne d'outils GCC

Guix offre des paquets de compilateurs individuels comme `gcc` mais si vous avez besoin d'une chaîne de compilation complète pour compiler et lier du code source, utilisez le paquet `gcc-toolchain`. Ce paquet fournit une chaîne d'outils complète GCC pour le développement



C/C++, dont GCC lui-même, la bibliothèque C de GNU (les en-têtes et les binaires, plus les symboles de débogage dans la sortie `debug`), Binutils et un wrapper pour l'éditeur de liens.

Le rôle de l'enveloppe est d'inspecter les paramètres `-L` et `-l` passés à l'éditeur de liens, d'ajouter des arguments `-rpath` correspondants et d'invoquer l'actuel éditeur de liens avec ce nouvel ensemble d'arguments. Vous pouvez dire au wrapper de refuser de lier les programmes à des bibliothèques en dehors du dépôt en paramétrant la variable d'environnement `GUIX_LD_WRAPPER_ALLOW_IMPURITIES` sur `no`.

Le paquet `gfortran-toolchain` fournit une chaîne d'outils GCC complète pour le développement en Fortran. Pour d'autres langages, veuillez utiliser '`guix search gcc toolchain`' (voir [Invoquer guix package], page 51).

## 8.5 Invoquer `guix git authenticate`

La commande `guix git authenticate` authentifie un checkout Git en suivant la même règle que pour les canaux (voir [channel-authentication], page 77). C'est-à-dire qu'à partir d'un commit donné, il s'assure que tous les commit suivants sont signés par une clé OpenPGP dont l'empreinte digitale apparaît dans le fichier `.guix-authorizations` de son ou ses commit(s) parent(s).

Vous allez trouver cette commande utile si vous maintenez un canal. Mais en fait, ce mécanisme d'authentification est utile dans un contexte plus large, Vous pourriez donc vouloir l'utiliser pour des dépôts Git qui n'ont rien à voir avec Guix.

La syntaxe générale est :

```
guix git authenticate commit signer [options...]
```

Par défaut, cette commande authentifie le checkout Git dans le répertoire courant ; il ne produit rien et sort avec le code de sortie zéro en cas de succès et non zéro en cas d'échec. *commit* ci-dessus désigne le premier commit où l'authentification a lieu, et *signer* est l'empreinte OpenPGP de la clé publique utilisée pour signer *commit*. Ensemble, ils forment une «introduction de canal» (voir [channel-authentication], page 77). Les options ci-dessous vous permettent d'affiner le processus.

`--repository=répertoire`

`-r répertoire`

Ouvre le dépôt Git dans *directory* au lieu du répertoire courant.

`--keyring=référence`

`-k référence`

Chargez le porte-clés OpenPGP à partir de *référence*, la référence d'une branche telle que `origin/keyring` ou `my-keyring`. La branche doit contenir des clés publiques OpenPGP dans des fichiers `.key`, soit sous forme binaire, soit "blindée ASCII". Par défaut, le porte-clés est chargé à partir de la branche nommée `keyring`.

`--stats` Affiche les statistiques sur les signatures de commits à l'issue de la procédure.

`--cache-key=key`

Les commits préalablement authentifiés sont mis en cache dans un fichier sous `~/.cache/guix/authentication`. Cette option force le cache à être stocké dans le fichier *key* de ce répertoire.

**--historical-authorizations=*fichier***

Par défaut, tout commit dont le(s) commit(s) parent(s) ne contient(nt) pas le fichier `.guix-authorizations` est considéré comme non authentique. En revanche, cette option considère les autorisations dans *file* pour tout commit dont le fichier `.guix-authorizations` est manquant. Le format de *file* est le même que celui de `.guix-authorizations` (au format voir [channel-authorizations], page 80).

## 9 Interface de programmation

GNU Guix fournit diverses interface de programmation Scheme (API) qui pour définir, construire et faire des requêtes sur des paquets. La première interface permet aux utilisateurs d'écrire des définitions de paquets de haut-niveau. Ces définitions se réfèrent à des concepts de création de paquets familiers, comme le nom et la version du paquet, son système de construction et ses dépendances. Ces définitions peuvent ensuite être transformées en actions concrètes lors de la construction.

Les actions de construction sont effectuées par le démon Guix, pour le compte des utilisateur·rice·s. Dans un environnement standard, le démon possède les droits en écriture sur le dépôt — le répertoire `/gnu/store` — mais pas les utilisateur·rice·s. La configuration recommandée permet aussi au démon d'effectuer la construction dans des chroots, à l'adresse des utilisateur·rice·s de construction spécifiques, pour minimiser les interférences avec le reste du système.

Il y a des API de plus bas niveau pour interagir avec le démon et le dépôt. Pour demander au démon d'effectuer une action de construction, les utilisateurs lui donnent en fait une *dérivation*. Une dérivation est une représentation à bas-niveau des actions de construction à entreprendre et l'environnement dans lequel elles devraient avoir lieu — les dérivations sont aux définitions de paquets ce que l'assembleur est aux programmes C. Le terme de « dérivation » vient du fait que les résultats de la construction en *dérivent*.

Ce chapitre décrit toutes ces API tour à tour, à partir des définitions de paquets à haut-niveau.

### 9.1 Modules de paquets

D'un point de vue programmatique, les définitions de paquets de la distribution GNU sont fournies par des modules Guile dans l'espace de noms (`gnu packages . . .`)<sup>1</sup> (voir Section “Modules” dans *GNU Guile Reference Manual*). Par exemple, le module (`gnu packages emacs`) exporte une variable nommée `emacs`, qui est liée à un objet `<package>` (voir Section 9.2 [Définition des paquets], page 105).

L'espace de nom (`gnu packages . . .`) est automatiquement scanné par les outils en ligne de commande. Par exemple, lorsque vous lancez `guix install emacs`, tous les modules (`gnu packages . . .`) sont scannés jusqu'à en trouver un qui exporte un objet de paquet dont le nom est `emacs`. Cette capacité à chercher des paquets est implémentée dans le module (`gnu packages`).

Les utilisateur·rice·s peuvent stocker les définitions de paquets dans des modules ayant des noms différents—par exemple, (`my-packages emacs`)<sup>2</sup>. Il y a deux façons de rendre ces définitions de paquets visibles pour les interfaces utilisateur·rice :

<sup>1</sup> Remarquez que les paquets sous l'espace de nom (`gnu packages . . .`) ne sont pas nécessairement des « paquets GNU ». Le nom de ce module suit la convention de nommage usuelle de Guile : `gnu` signifie que ces modules sont distribués dans le système GNU, et `packages` identifie les modules qui définissent les paquets.

<sup>2</sup> Remarquez que le nom du fichier et le nom du module doivent correspondre. Par exemple, le module (`my-packages emacs`) doit être stocké dans un fichier `my-packages/emacs.scm` relatif au chemin de chargement spécifié avec `--load-path` ou `GUIX_PACKAGE_PATH`. Voir Section “Modules and the File System” dans *GNU Guile Reference Manual*, pour plus de détails.

1. En ajoutant le répertoire contenant vos modules de paquets au chemin de recherche avec le tag `-L` de `guix package` et des autres commandes (voir Section 10.1.1 [Options de construction communes], page 180) ou en indiquant la variable d'environnement `GUIX_PACKAGE_PATH` décrite plus bas.
2. En définissant un *canal* et en configurant `guix pull` pour qu'il l'utilise. Un canal est en substance un dépôt Git contenant des modules de paquets. Voir Chapitre 7 [Canaux], page 75, pour plus d'informations sur comment définir et utiliser des canaux.

`GUIX_PACKAGE_PATH` fonctionne comme les autres variables de chemins de recherche :

**GUIX\_PACKAGE\_PATH** [Variable d'environnement]  
 C'est une liste séparée par des deux-points de répertoires dans lesquels trouver des modules de paquets supplémentaires. Les répertoires listés dans cette variable sont prioritaires par rapport aux paquets de la distribution.

La distribution est entièrement *bootstrappée* et *auto-contenue* : chaque paquet est construit uniquement à partir d'autres paquets de la distribution. La racine de ce graphe de dépendance est un petit ensemble de *binaires de bootstrap* fournis par le module (`gnu packages bootstrap`). Pour plus d'informations sur le bootstrap, voir Chapitre 20 [Bootstrapping], page 659.

## 9.2 Définition des paquets

L'interface de haut-niveau pour les définitions de paquets est implémentée dans les modules (`guix packages`) et (`guix build-system`). Par exemple, la définition du paquet, ou la *recette*, du paquet GNU Hello ressemble à cela :

```
(define-module (gnu packages hello)
  #:use-module (guix packages)
  #:use-module (guix download)
  #:use-module (guix build-system gnu)
  #:use-module (guix licenses)
  #:use-module (gnu packages gawk))

(define-public hello
  (package
    (name "hello")
    (version "2.10")
    (source (origin
              (method url-fetch)
              (uri (string-append "mirror://gnu/hello/hello-" version
                                   ".tar.gz"))
              (sha256
                (base32
                  "0ssi1wpaf7plawqqjwigppsg5fyh99vdlb9kz17c9lmg89ndq1i")))))
    (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+))
```

Sans être expert·e en Scheme, on peut comprendre la signification des différents champs présents. Cette expression lie la variable `hello` à un objet `<package>`, qui est en substance un enregistrement (voir Section “SRFI-9” dans *GNU Guile Reference Manual*). On peut inspecter cet objet de paquet avec les procédures qui se trouvent dans le module (`guix packages`) ; par exemple, (`package-name hello`) renvoie — ô surprise ! — `"hello"`.

Avec un peu de chance, vous pourrez importer tout ou partie de la définition du paquet qui vous intéresse depuis un autre répertoire avec la commande `guix import` (voir Section 10.5 [Invoquer `guix import`], page 197).

Dans l'exemple précédent, `hello` est défini dans un module à part, (`gnu packages hello`). Techniquement, cela n'est pas strictement nécessaire, mais c'est pratique : tous les paquets définis dans des modules sous (`gnu packages ...`) sont automatiquement connus des outils en ligne de commande (voir Section 9.1 [Modules de paquets], page 104).

Il y a quelques points à remarquer dans la définition de paquet précédente :

- Le champ `source` du paquet est un objet `<origin>` (voir Section 9.2.2 [référence de origin], page 113, pour la référence complète). Ici, on utilise la méthode `url-fetch` de (`guix download`), ce qui signifie que la source est un fichier à télécharger par FTP ou HTTP.

Le préfixe `mirror://gnu` demande à `url-fetch` d'utiliser l'un des miroirs GNU définis dans (`guix download`).

Le champ `sha256` spécifie le hash SHA256 attendu pour le fichier téléchargé. Il est requis et permet à Guix de vérifier l'intégrité du fichier. La forme (`base32 ...`) introduit a représentation en base32 du hash. Vous pouvez obtenir cette information avec `guix download` (voir Section 10.3 [Invoquer `guix download`], page 195) et `guix hash` (voir Section 10.4 [Invoquer `guix hash`], page 196).

Lorsque cela est requis, la forme `origin` peut aussi avec un champ `patches` qui liste les correctifs à appliquer et un champ `snippet` qui donne une expression Scheme pour modifier le code source.

- Le champ `build-system` spécifie la procédure pour construire le paquet (voir Section 9.5 [Systèmes de construction], page 124). Ici, `gnu-build-system` représente le système de construction GNU familier, où les paquets peuvent être configurés, construits et installés avec la séquence `./configure && make && make check && make install` habituelle.

Lorsque vous commencez à empaqueter des logiciels non triviaux, vous pouvez avoir besoin d'outils pour manipuler ces phases de construction, manipuler des fichiers, etc. Voir Section 9.7 [Utilitaires de construction], page 146, pour en savoir plus.

- Le champ `arguments` spécifie des options pour le système de construction (voir Section 9.5 [Systèmes de construction], page 124). Ici il est interprété par `gnu-build-system` comme une demande de lancer `configure` avec le tag `--enable-silent-rules`. Que sont ces apostrophes (`'`) ? C'est de la syntaxe Scheme pour introduire une liste ; `'` est synonyme de la fonction `quote`. Parfois vous verrez aussi ``` (un accent grave synonyme de `quasiquote`) et `,` (une virgule, synonyme de `unquote`). Voir Section

“Expression Syntax” dans *GNU Guile Reference Manual*, pour des détails. Ice la valeur du champ `arguments` est une liste d’arguments passés au système de construction plus tard, comme avec `apply` (voir Section “Fly Evaluation” dans *GNU Guile Reference Manual*).

La séquence dièse-deux-points (`#:`) définit un *mot-clef* Scheme (voir Section “Keywords” dans *GNU Guile Reference Manual*), et `#:configure-flags` est un mot-clef utilisé pour passer un argument au système de construction (voir Section “Coding With Keywords” dans *GNU Guile Reference Manual*).

- Le champ `inputs` spécifie les entrées du processus de construction — c.-à-d. les dépendances à la construction ou à l’exécution du paquet. Ici, nous ajoutons en entrée une référence à la variable `gawk` ; `gawk` est lui-même lié à un objet `<package>`.

Remarquez que GCC, Coreutils, Bash et les autres outils essentiels n’ont pas besoin d’être spécifiés en tant qu’entrées ici. À la place, le `gnu-build-system` est en mesure de s’assurer qu’ils sont présents (voir Section 9.5 [Systèmes de construction], page 124).

Cependant, toutes les autres dépendances doivent être spécifiées dans le champ `inputs`. Toute dépendance qui ne serait pas spécifiée ici sera simplement indisponible pour le processus de construction, ce qui peut mener à un échec de la construction.

Voir Section 9.2.1 [référence de package], page 108, pour une description complète des champs possibles.

#### **Pour aller plus loin:**

Vous vous sentez intimidé·e par le langage Scheme ou vous êtes curieux·se de le découvrir ? Le livre de recettes a une petite section pour démarrer qui résume une partie de ce qui a été montré au-dessus et explique les bases. Voir Section “Cours accéléré du langage Scheme” dans *le livre de recettes de GNU Guix*, pour plus d’information.

Lorsqu’une définition de paquet est en place, le paquet peut enfin être construit avec l’outil en ligne de commande `guix build` (voir Section 10.1 [Invoquer `guix build`], page 180), pour résoudre les échecs de construction que vous pourriez rencontrer (voir Section 10.1.4 [Débogage des échecs de construction], page 193). Vous pouvez aisément revenir à la définition du paquet avec la commande `guix edit` (voir Section 10.2 [Invoquer `guix edit`], page 194). Voir Section 22.4 [Consignes d’empaquetage], page 673, pour plus d’informations sur la manière de tester des définitions de paquets et Section 10.8 [Invoquer `guix lint`], page 213, pour des informations sur la manière de vérifier que la définition respecte les conventions de style.

Enfin, voir Chapitre 7 [Canaux], page 75, pour des informations sur la manière d’étendre la distribution en ajoutant vos propres définitions de paquets dans un « canal ».

Finalement, la mise à jour de la définition du paquet à une nouvelle version amont peut en partie s’automatiser avec la commande `guix refresh` (voir Section 10.6 [Invoquer `guix refresh`], page 205).

Sous le capot, une dérivation qui correspond à un objet `<package>` est d’abord calculé par la procédure `package-derivation`. Cette dérivation est stockée dans un fichier `.drv` dans `/gnu/store`. Les actions de construction qu’il prescrit peuvent ensuite être réalisées par la procédure `build-derivation` (voir Section 9.9 [Le dépôt], page 157).

**package-derivation store package** [*system*] [Procédure Scheme]

Renvoie l'objet `<derivation>` du *paquet* pour le *système* (voir Section 9.10 [Dérivations], page 159).

*paquet* doit être un objet `<package>` valide et *système* une chaîne indiquant le type de système cible — p. ex. "x86\_64-linux" pour un système GNU x86\_64 basé sur Linux. *dépôt* doit être une connexion au démon, qui opère sur les dépôt (voir Section 9.9 [Le dépôt], page 157).

De manière identique, il est possible de calculer une dérivation qui effectue une compilation croisée d'un paquet pour un autre système :

**package-cross-derivation store paquet cible** [*système*] [Procédure Scheme]

*renvoie l'objet* `<derivation>`

du *paquet* construit depuis *système* pour *cible*.

*cible* doit être un triplet GNU valide indiquant le matériel cible et le système d'exploitation, comme "aarch64-linux-gnu" (voir Section "Specifying Target Triplets" dans *Autoconf*).

Une fois qu'on a une définition de paquet, on peut facilement définir des *variantes* du paquet. Voir Section 9.3 [Définition de variantes de paquets], page 116, pour plus d'informations.

### 9.2.1 Référence de package

Cette section résume toutes les options disponibles dans les déclarations `package` (voir Section 9.2 [Définition des paquets], page 105).

**package** [Type de données]

C'est le type de donnée représentant une recette de paquets.

**name** Le nom du paquet, comme une chaîne de caractères.

**version** La version du paquet, comme une chaîne de caractères. Voir Section 22.4.3 [Numéros de version], page 674, pour un guide.

**source** Un objet qui indique comment le code source du paquet devrait être récupéré. La plupart du temps, c'est un objet `origin` qui indique un fichier récupéré depuis internet (voir Section 9.2.2 [référence de origin], page 113). Il peut aussi s'agir de tout autre objet « simili-fichier » comme un `local-file` qui indique un fichier du système de fichier local (voir Section 9.12 [G-Expressions], page 166).

**build-system**

Le système de construction qui devrait être utilisé pour construire le paquet (voir Section 9.5 [Systèmes de construction], page 124).

**arguments** (par défaut : '() )

Les arguments à passer au système de construction (voir Section 9.5 [Systèmes de construction], page 124). C'est une liste qui contient typiquement une séquence de paires de clefs-valeurs, comme dans cet exemple :

```
(package
```

```
(name "exemple")
;; plusieurs champs omis
(arguments
  (list #:tests? #f ;passer les tests
        #:make-flags #~'("VERBOSE=1") ;passer les arguments à « mak
        #:configure-flags #~'("--enable-frobbing"))))
```

L'ensemble exact des mots-clés pris en charge dépend du système de construction (voir Section 9.5 [Systèmes de construction], page 124), mais vous trouverez qu'ils prennent presque tous en compte `#:configure-flags`, `#:make-flags`, `#:tests?` et `#:phases`. Le mot-clé `#:phases` en particulier vous permet de modifier l'ensemble des phases de construction pour votre paquet (voir Section 9.6 [Phases de construction], page 143).

```
inputs (par défaut : '())
native-inputs (par défaut : '())
propagated-inputs (par défaut : '())
```

Ces champs listent les dépendances du paquet. Chacune de ces listes est soit un paquet, une origine ou un autre « objet simili-fichier » (voir Section 9.12 [G-Expressions], page 166) ; pour spécifier la sortie de cet objet simili-fichier à utiliser, passez une liste à deux éléments où le second élément est la sortie (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59, pour plus d'informations sur les sorties des paquets). Par exemple, la liste suivante spécifie trois entrées :

```
(list libffi libunistring
      `(,glib "bin")) ;la sortie « bin » de GLib
```

Dans l'exemple au-dessus, on utilise la sortie "out" de `libffi` et de `libunistring`.

**Notes de compatibilité:** Jusqu'à la version 1.3.0, les listes d'entrées étaient des listes de tuples, où chaque tuple a une étiquette pour une entrée (une chaîne de caractères) comme premier élément, un paquet, une origine ou une dérivation comme deuxième élément et éventuellement le nom d'une sortie à utiliser qui est "out" par défaut. Par exemple, la liste ci-dessous est équivalente à celle au-dessus, mais utilise *l'ancien style d'entrées* :

```
;; Ancien style d'entrées (obsolète)
`(("libffi" ,libffi)
  ("libunistring" ,libunistring)
  ("glib:bin" ,glib "bin")) ;la sortie "bin" de Glib
```

Ce style est maintenant obsolète ; il est toujours pris en charge mais la prise en charge sera supprimée dans une future version. Vous ne devriez pas l'utiliser pour de nouvelles définitions de paquets. Voir Section 10.7 [Invoquer guix style], page 210, pour apprendre à migrer vers le nouveau style.

La distinction entre `native-inputs` et `inputs` est nécessaire lorsqu'on considère la compilation croisée. Lors d'une compilation croisée, les



dépendances listées dans `inputs` sont construites pour l'architecture *cible* ; inversement, les dépendances listées dans `native-inputs` sont construites pour l'architecture de la machine de *construction*.

`native-inputs` est typiquement utilisé pour lister les outils requis à la construction mais pas à l'exécution, comme Autoconf, Automake, pkg-config, Gettext ou Bison. `guix lint` peut rapporter des erreurs de ce type (voir Section 10.8 [Invoquer guix lint], page 213).

Enfin, `propagated-inputs` est similaire à `inputs`, mais les paquets spécifiés seront automatiquement installés sur les profils (voir Section 6.1 [Fonctionnalités], page 43) à côté du paquet auquel ils appartiennent (voir [package-cmd-propagated-inputs], page 46, pour savoir comment `guix package` traite les entrées propagées).

Par exemple, cela est nécessaire lors de l'empaquetage d'une bibliothèque C/C++ qui a besoin des en-têtes d'une autre bibliothèque pour se compiler, ou lorsqu'un fichier pkg-config fait référence à un autre *via* son champ `Requires`.

Un autre exemple où le `propagated-inputs` est utile, c'est pour les langues qui ne disposent pas de la possibilité d'enregistrer le chemin de recherche à l'exécution comme le `RUNPATH` des fichiers ELF ; cela inclut Guile, Python, Perl, et plus encore. Lors de l'empaquetage de bibliothèques écrites dans ces langages, assurez-vous qu'elles peuvent trouver le code de bibliothèque dont elles dépendent au moment de l'exécution en listant les dépendances d'exécution dans `propagated-inputs` plutôt que `inputs`.

`outputs` (par défaut : `'("out")`)

La liste des noms de sorties du paquet. Voir Section 6.4 [Des paquets avec plusieurs résultats], page 59, pour des exemples typiques d'utilisation de sorties supplémentaires.

`native-search-paths` (par défaut : `'()`)

`search-paths` (par défaut : `'()`)

Une liste d'objets `search-path-specification` décrivant les variables d'environnement de recherche de chemins que ce paquet prend en compte. Voir Section 9.8 [Chemins de recherche], page 154, pour plus de détails sur les spécifications de chemins de recherche.

Comme pour les entrées, la distinction entre `native-search-paths` et `search-paths` n'est importante que pour la compilation croisée. Lors d'une compilation croisée, `native-search-paths` ne s'applique qu'aux entrées natives alors que `search-paths` ne s'applique qu'aux entrées hôtes.

Les paquets comme les compilateurs croisés font attention aux entrées de la cible — par exemple notre compilateur GCC croisé (modifié) a `CROSS_C_INCLUDE_PATH` dans `search-paths`, ce qui lui permet de trouver les fichiers `.h` du système cible et *pas* ceux des entrées natives. Pour la plupart des paquets cependant, seul `native-search-paths` a du sens.

**replacement** (par défaut : **#f**)  
 Ce champ doit être soit **#f** soit un objet de paquet qui sera utilisé comme *remplaçant* de ce paquet. Voir Chapitre 19 [Mises à jour de sécurité], page 657, pour plus de détails.

**synopsis** Une description sur une ligne du paquet.

**description**  
 Une description plus détaillée du paquet, en tant que chaîne au format Texinfo.

**license** La licence du paquet ; une valeur tirée de (**guix licenses**) ou une liste de ces valeurs.

**home-page**  
 L'URL de la page d'accueil du paquet, en tant que chaîne de caractères.

**supported-systems** (par défaut : **%supported-systems**)  
 La liste des systèmes supportés par le paquet, comme des chaînes de caractères de la forme **architecture-noyau**, par exemple **"x86\_64-linux"**.

**location** (par défaut : emplacement de la source de la forme **package**)  
 L'emplacement de la source du paquet. C'est utile de le remplacer lorsqu'on hérite d'un autre paquet, auquel cas ce champ n'est pas automatiquement corrigé.

**this-package** [Syntaxe Scheme]  
 Lorsque vous l'utilisez dans la *portée lexicale* du champ d'une définition de paquet, cet identifiant est résolu comme étant le paquet définit.  
 L'exemple ci-dessous montre l'ajout d'un paquet comme entrée native de lui-même pour la compilation croisée :

```
(package
  (name "guile")
  ;; ...

  ;; Lors de la compilation croisée, Guile par exemple dépend
  ;; d'une version native de lui-même. On l'ajoute ici.
  (native-inputs (if (%current-target-system)
                    (list this-package)
                    '()))))
```

C'est une erreur que de se référer à **this-package** en dehors de la définition d'un paquet.

Les procédures auxiliaires suivantes vous aideront à utiliser les entrées des paquets.

**lookup-package-input** *package name* [Procédure Scheme]  
**lookup-package-native-input** *package name* [Procédure Scheme]  
**lookup-package-propagated-input** *package name* [Procédure Scheme]  
**lookup-package-direct-input** *package name* [Procédure Scheme]  
 Cherche *name* dans les entrées de *package* (ou les entrées natives, propagées ou directes). Renvoie l'entrée si elle la trouve, et **#f** sinon.

*name* est le nom d'un paquet de dépendance. Voici comment vous pouvez l'utiliser :

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

```
(lookup-package-direct-input coreutils "gmp")
⇒ #<package gmp@6.2.1 ...>
```

Dans cet exemple, on obtient le paquet `gmp` qui fait partie des entrées directes de `coreutils`.

Parfois vous voudrez obtenir la liste des entrées requises pour *développer* un paquet — toutes les entrées visibles quand le paquet est compilé. C'est ce que la procédure `package-development-inputs` renvoie.

```
package-development-inputs package [system] [#:target] [Procédure Scheme]
#f]
```

Renvoie la liste des entrées requises par *package* pour le développement sur *system*. Lorsque *target* est vrai, les entrées renvoyées sont celles requises pour la compilation croisée de *package* de *system* vers *target*, où *target* est un triplet comme `"aarch64-linux-gnu"`.

Remarquez que le résultat inclus à la fois les entrées explicites et implicites — les entrées ajoutées automatiquement par le système de construction (voir Section 9.5 [Systèmes de construction], page 124). Prenons le paquet `hello` pour illustrer :

```
(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 ...>) ...)
```

Dans cet exemple, `package-direct-inputs` renvoie la liste vide, parce que `hello` n'a pas de dépendance explicite. À l'inverse, `package-development-inputs` inclut les entrées implicites ajoutées par `gnu-build-system` qui sont requises pour construire `hello` : `tar`, `gzip`, `GCC`, `libc`, `Bash` et d'autres. Pour les visualiser, `guix graph hello` montrerait les entrées explicites, alors que `guix graph -t bag hello` inclurait les entrées implicites (voir Section 10.10 [Invoquer `guix graph`], page 218).

Comme les paquets sont des objets Scheme réguliers qui capturent un graphe de dépendance complet et les procédures de construction associées, il est souvent utile d'écrire des procédures qui prennent un paquet et renvoient une version modifiée de celui-ci en fonction de certains paramètres. En voici quelques exemples.

```
package-with-c-toolchain package toolchain [Procédure Scheme]
```

Renvoie une variante de *package* qui utilise *toolchain* au lieu de la chaîne d'outils GNU C/C++ par défaut. *toolchain* doit être une liste d'entrées (tuples `label/package`) fournissant une fonctionnalité équivalente, comme le paquet `gcc-toolchain`.

L'exemple ci-dessous renvoie une variante du paquet `hello` construit avec GCC 10.x et le reste de la chaîne d'outils GNU (Binutils et la bibliothèque C GNU) au lieu de la chaîne d'outils par défaut :

```
(let ((toolchain (specification->package "gcc-toolchain@10")))
      (package-with-c-toolchain hello `(("toolchain" ,toolchain))))
```

La chaîne d'outils de construction fait partie des *entrées implicites* des paquets— elle n'est généralement pas listée comme faisant partie des différents champs "entrées" et est à la place récupérée par le système de construction. Par conséquent, cette procédure fonctionne en modifiant le système de compilation de *paquet* de sorte qu'il utilise *toolchain* au lieu des valeurs par défaut. Section 9.5 [Systèmes de construction], page 124, pour en savoir plus sur les systèmes de construction.

### 9.2.2 Référence de origin

Cette section documente *origins*. Une déclaration `origin` spécifie les données qui doivent être "produites" – téléchargées, généralement – et dont le contenu est connu à l'avance. Les origines sont principalement utilisées pour représenter le code source des paquets (voir Section 9.2 [Définition des paquets], page 105). Pour cette raison, le formulaire `origin` permet de déclarer des correctifs à appliquer au code source original ainsi que des bribes de code pour le modifier.

`origin` [Type de données]

C'est le type de donnée qui représente l'origine d'un code source.

**uri** Un objet contenant l'URI de la source. Le type d'objet dépend de la `method` (voir plus bas). Par exemple, avec la méthode `url-fetch` de (`guix download`), les valeurs valide d'`uri` sont : une URL représentée par une chaîne de caractères, ou une liste de chaînes de caractères.

**method** Une procédure monadique qui gère l'URI donné. La procédure doit accepter au moins trois arguments : la valeur du champ `uri` et l'algorithme de hachage et la valeur de hachage spécifiée par le champ `hash`. Elle doit renvoyer un élément du dépôt ou une dérivation dans la monade du dépôt (voir Section 9.11 [La monade du dépôt], page 162) ; la plupart des méthodes renvoient une dérivation à sortie fixe (voir Section 9.10 [Dérivations], page 159).

Les méthodes couramment utilisées comprennent `url-fetch`, qui récupère les données à partir d'une URL, et `git-fetch`, qui récupère les données à partir d'un dépôt Git (voir ci-dessous).

**sha256** Un bytevector contenant le hachage SHA-256 de la source. Cela équivaut à fournir un `content-hash` Objet SHA256 dans le champ `hash` décrit ci-dessous.

**hash** L'objet `content-hash` de la source—voir ci-dessous pour savoir comment utiliser `content-hash`.

Vous pouvez obtenir cette information avec `guix download` (voir Section 10.3 [Invoquer guix download], page 195) ou `guix hash` (voir Section 10.4 [Invoquer guix hash], page 196).

`file-name` (par défaut : `#f`)

Le nom de fichier à utiliser pour sauvegarder le fichier. Lorsqu'elle est à `#f`, une valeur par défaut raisonnable est utilisée dans la plupart des cas. Dans le cas où la source est récupérée depuis une URL, le nom de fichier est celui de l'URL. Pour les sources récupérées depuis un outil de contrôle de version, il est recommandé de fournir un nom de fichier explicitement parce que le nom par défaut n'est pas très descriptif.

`patches` (par défaut : `'()`)

Une liste de noms de fichiers, d'origines ou d'objets simili-fichiers (voir Section 9.12 [G-Expressions], page 166) qui pointent vers des correctifs à appliquer sur la source.

Cette liste de correctifs doit être inconditionnelle. En particulier, elle ne peut pas dépendre des valeurs de `%current-system` ou `%current-target-system`.

`snippet` (par défaut : `#f`)

Une G-expression (voir Section 9.12 [G-Expressions], page 166) ou une S-expression qui sera lancée dans le répertoire des sources. C'est une manière pratique de modifier la source, parfois plus qu'un correctif.

`patch-flags` (par défaut : `'("-p1")`)

Une liste de drapeaux à passer à la commande `patch`.

`patch-inputs` (par défaut : `#f`)

Paquets d'entrées ou dérivations pour le processus de correction. Lorsqu'elle est à `#f`, l'ensemble d'entrées habituellement nécessaire pour appliquer des correctifs est fournit, comme GNU Patch.

`modules` (par défaut : `'()`)

Une liste de modules Guile qui devraient être chargés pendant le processus de correction et pendant que le lancement du code du champ `snippet`.

`patch-guile` (par défaut : `#f`)

Le paquet Guile à utiliser dans le processus de correction. Lorsqu'elle est à `#f`, une valeur par défaut raisonnable est utilisée.

`content-hash` *value* [*algorithm*] [Data Type]

Construire un objet de hash de contenu pour l'*algorithme* donné, et avec *valeur* comme valeur de hachage. Lorsque *algorithme* est omis, on suppose qu'il s'agit de `sha256`.

*value* peut être une chaîne littérale, auquel cas elle est décodée en base32, ou il peut s'agir d'un bytevecteur.

Les options suivantes sont équivalentes :

```
(content-hash "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj")
(content-hash "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj"
 sha256)
(content-hash (base32
 "05zxkyz9bv3j9h0xyid1rhvh3klhsmrpkf3bcs6frvlgyr2gwilj"))
(content-hash (base64 "kkb+RPaP7uyMZmu4eXPVkm4BN8yhRd8BTHLslb6f/Rc=")
 sha256)
```

Techniquement, le `content-hash` est actuellement mis en œuvre sous forme de macro. Il effectue des contrôles de bon sens au moment de l'expansion de la macro, lorsque cela est possible, en s'assurant par exemple que *valeur* a la bonne taille pour *algorithme*.

Comme nous l'avons vu plus haut, la manière dont les données auxquelles une origine se réfère exactement sont récupérées est déterminée par son champ `method`. Le module (`guix download`) fournit la méthode la plus courante, `url-fetch`, décrite ci-dessous.

`url-fetch url hash-algo hash [name] [#:executable ? #f]` [Procédure Scheme]  
*Renvoie une dérivation à sortie fixe qui récupère*

les données de *url* (une chaîne de caractères, ou une liste de chaînes de caractères désignant des URLs alternatives), qui est censée avoir un hash *hash* de type *hash-algo* (un symbole). Par défaut, le nom du fichier est le nom de base de l'URL ; en option, *nom* peut spécifier un nom de fichier différent. Lorsque *executable?* est vrai, rendez le fichier téléchargé exécutable.

Lorsque l'une des URL commence par `mirror://`, sa partie hôte est alors interprétée comme le nom d'un schéma de miroir, tiré de `%mirror-file`.

Alternativement, lorsque l'URL commence par `file://`, renvoie le nom du fichier correspondant dans le dépôt.

De même, le module (`guix git-download`) définit la méthode d'origine `git-download`, qui récupère les données d'un dépôt de contrôle de version Git, et le type de données `git-reference` pour décrire le dépôt et la révision à récupérer.

`git-fetch ref hash-algo hash` [Procédure Scheme]  
*Renvoie une dérivation à sortie fixe qui va chercher *ref*, un objet <git-reference>. La sortie est censée avoir un hash récursif *hash* de type *hash-algo* (un symbole). Utilisez *name* comme nom de fichier, ou un nom générique si *#f*.*

`git-reference` [Type de données]  
 Ce type de données représente une référence Git pour le `git-fetch` à récupérer.

`url` L'URL du dépôt Git à cloner.

`commit` Cette chaîne indique soit le commit à récupérer (une chaîne hexadécimale), soit le tag à récupérer. Vous pouvez aussi utiliser une chaîne de commit « courte » ou un identifiant dans le style de `git describe` comme `v1.0.1-10-g58d7909c97`.

`recursive?` (par défaut : `#f`)

Ce booléen indique s'il faut aller chercher récursivement les sous-modules de Git.

L'exemple ci-dessous indique la balise `v2.10` du dépôt GNU Hello :

```
(git-reference
 (url "https://git.savannah.gnu.org/git/hello.git")
 (commit "v2.10"))
```

C'est équivalent à la référence ci-dessous, qui nomme explicitement le commit :

```
(git-reference
 (url "https://git.savannah.gnu.org/git/hello.git")
 (commit "dc7dc56a00e48fe6f231a58f6537139fe2908fb9"))
```

Pour les dépôts Mercurial, le module (`guix hg-download`) définit la méthode d'origine `hg-download` et le type de données `hg-reference` pour la prise en charge du système de contrôle de version Mercurial.

`hg-fetch ref hash-algo hash [name]` [Procédure Scheme]

Renvoie une dérivation à sortie fixe qui va chercher *ref*, un objet `<hg-reference>`. La sortie est censée avoir un hash récursif *hash* de type *hash-algo* (un symbole). Utilisez *name* comme nom de fichier, ou un nom générique si `#false`.

### 9.3 Définition de variantes de paquets

L'une des choses intéressantes avec Guix est qu'à partir d'une définition, on peut facilement *dériver* des variantes du paquet — pour une version en amont différente, avec des dépendances différentes, des options de compilation différentes, etc. Certains de ces paquets personnalisés peuvent être définis directement par la ligne de commande (voir Section 10.1.2 [Options de transformation de paquets], page 183). Cette section décrit comment définir des variantes de paquets avec du code. Cela est utile dans les « manifestes » (voir Section 9.4 [Écrire un manifeste], page 120) et dans votre propre collection de paquets (voir Section 7.6 [Écrire de nouveaux de canaux], page 78), entre autres choses !

Comme nous en avons parlé plus tôt, les paquets sont des objets de première classe dans le langage Scheme. Le module (`guix packages`) fournit la construction `package` pour définir de nouveaux objets de paquets (voir Section 9.2.1 [référence de package], page 108). La manière la plus simple de définir la variante d'un paquet est d'utiliser le mot-clé `inherit` avec `package`. Cela vous permet d'hériter la définition d'un paquet tout en redéfinissant les champs que vous voulez.

Par exemple, étant donné la variable `hello`, qui contient la définition d'une version récente de GNU Hello, voici comment définir une variante pour la version 2.2 (publiée en 2006, c'est vintage !) :

```
(use-modules (gnu packages base)) ;pour '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
                "0lappv4slgb5spyqbh6yl5r013zv72yqg2pc130mginf3wdqd8k9"))))))
```

L'exemple ci-dessus correspond à ce que l'option de transformation des paquets `--with-source` fait. En substance, `hello-2.2` préserve tous les champs de `hello`, sauf `version` et `source`, qu'il remplace. Remarquez que la variable `hello` originale est toujours là, dans le module (`gnu packages base`), et n'est pas modifiée. Lorsque vous définissez un paquet personnalisé comme ceci, vous *ajoutez* en fait une nouvelle définition de paquet ; la version originale est toujours disponible.

Vous pouvez aussi bien définir des variantes avec un ensemble de dépendances différent du paquet d'origine. Par exemple, le paquet `gdb` par défaut dépend de `guile`, mais comme c'est une dépendance facultative, vous pouvez définir une variante qui supprime cette dépendance de cette manière :

```
(use-modules (gnu packages gdb)) ; pour « gdb »

(define gdb-sans-guile
  (package
    (inherit gdb)
    (inputs (modify-inputs (package-inputs gdb)
                          (delete "guile")))))
```

La forme `modify-inputs` ci-dessus supprime le paquet `"guile"` du champ `inputs` de `gdb`. La macro `modify-inputs` est une macro auxiliaire qui peut être utile si vous voulez supprimer, ajouter et remplacer des paquets en entrée.

`modify-inputs` *inputs clauses* [Syntaxe Scheme]

Modifie les entrées de paquet spécifiées, telles que renvoyées par `package-inputs & co.`, suivant les clauses données. Chaque clause doit avoir l'une des formes suivantes :

```
(delete nom...)
    Supprime des entrées les paquets avec les noms donnés (chaines de caractères).

(prepend paquet...)
    Ajoute les paquets au début de la liste des entrées.

(append paquet...)
    Ajoute les paquets à la fin de la liste des entrées.
```

L'exemple suivant supprime les entrées `GMP` et `ACL` de `Coreutils` et ajoute `libcap` au début de la liste des entrées :

```
(modify-inputs (package-inputs coreutils)
  (delete "gmp" "acl")
  (prepend libcap))
```

L'exemple suivant remplace le paquet `guile` des entrées de `guile-redis` avec `guile-2.2` :

```
(modify-inputs (package-inputs guile-redis)
  (replace "guile" guile-2.2))
```

Le dernier type de clause est `append`, pour ajouter des entrées à la fin de la liste.

Dans certains cas, vous pouvez trouver cela utile d'écrire des fonctions (« procédures », pour Scheme) qui renvoie un paquet en fonction de certains paramètres. Par exemple, considérez la bibliothèque `luasocket` pour le langage de programmation lua. Nous voulons créer des paquets `luasocket` pour les versions majeures de Lua. Une manière de faire est de définir une procédure qui prend un paquet Lua et renvoie un paquet `luasocket` qui en dépend :

```
(define (make-lua-socket name lua)
  ;; Renvoie un paquet luasocket construit avec LUA.
```



```
(package
  (name name)
  (version "3.0")
  ;; plusieurs champs ont été omis
  (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))
```

ici nous avons défini les paquets `lua5.1-socket` et `lua5.2-socket` en appelant `make-lua-socket` avec différents arguments. Voir Section “Procedures” dans *GNU Guile Reference Manual*, pour plus d’informations sur les procédures. Avoir des définitions publiques au premier niveau pour ces deux paquets signifie qu’ils sont disponibles depuis la ligne de commande (voir Section 9.1 [Modules de paquets], page 104).

Ce sont des variantes simples de paquets. Par simplicité, le module (`guix transformations`) fournit une interface de haut niveau qui correspond directement aux options de transformations de paquets plus sophistiquées (voir Section 10.1.2 [Options de transformation de paquets], page 183) :

`options->transformation opts` [Procédure Scheme]

Renvoie une procédure qui, lorsqu’on lui passe un objet à construire (un paquet, une dérivation, etc), applique les transformations spécifiées par *opts* et renvoie les objets qui en résultent. *opts* doit être une liste de paires de symboles et de chaîne comme dans :

```
((with-branch . "guile-gcrypt=master")
 (without-tests . "libgcrypt"))
```

Chaque nom de symbole nomme une transformation et la chaîne correspondante est un argument pour cette transformation.

Par exemple, un manifeste équivalent à cette commande :

```
guix build guix \
  --with-branch=guile-gcrypt=master \
  --with-debug-info=zlib
```

... ressemblerait à ceci :

```
(use-modules (guix transformations))

(define transform
  ;; La procédure de transformation des paquets.
  (options->transformation
    '((with-branch . "guile-gcrypt=master")
      (with-debug-info . "zlib"))))

(packages->manifest
```

```
(list (transform (specification->package "guix"))))
```

La procédure `options->transformation` est pratique, mais elle n'est peut-être pas aussi flexible que vous pourriez le souhaiter. Comment est-elle implémentée ? Le lecteur attentif aura sans doute remarqué que la plupart des options de transformation des paquets va plus loin que les changements superficiels montrés dans les premiers exemples de cette sections : ils font de la *réécriture d'entrées*, où le graphe de dépendance d'un paquet est réécrit en remplaçant des entrées spécifiques par d'autres.

La procédure `package-input-rewriting` de (`guix packages`) implémente la réécriture du graphe de dépendance, pour remplacer des paquets dans le graphe.

`package-input-rewriting replacements [nom-réécrit]` [Procédure Scheme]

[#:deep ? #t] Renvoie une procédure qui, lorsqu'elle est passée à un paquet, remplace ses dépendances directes et indirectes, y compris les entrées implicites lorsque *deep?* est vrai, selon *replacements*. *replacements* est une liste de paires de paquets ; le premier élément de chaque paire est le paquet à remplacer, et le second est le remplacement.

De manière facultative, *nom-réécrit* est une procédure à un argument qui prend le nom d'un paquet et renvoie son nouveau nom après l'avoir réécrit.

Regardez cet exemple :

```
(define libressl-instead-of-openssl
  ;; Cette procédure remplace OPENSSL par LIBRESSL,
  ;; récursivement.
  (package-input-rewriting `((,openssl . ,libressl))))

(define git-with-libressl
  (libressl-instead-of-openssl git))
```

Ici nous définissons d'abord une procédure de réécriture qui remplace *openssl* par *libressl*. Ensuite nous l'utilisons pour définir une *variante* du paquet *git* qui utilise *libressl* plutôt que *openssl*. cela est exactement ce que l'option en ligne de commande `--with-input` fait (voir Section 10.1.2 [Options de transformation de paquets], page 183).

La variante suivante de `package-input-rewriting` peut repérer les paquets à remplacer par nom à la place de leur identité.

`package-input-rewriting/spec replacements[#:deep? #t]` [Procédure Scheme]

Renvoie une procédure qui, étant donné un paquet, applique les *replacements* donnés à tout le graphe du paquet, y compris les entrées implicites, à moins que *deep?* ne soit faux. *replacements* est une liste de paires spécifications/procédures ; chaque spécification est une spécification de paquet telle que "gcc" ou "guile@2", et chaque procédure prend un paquet correspondant et renvoie un remplacement pour ce paquet.

L'exemple ci-dessus pourrait être réécrit de cette manière :

```
(define libressl-instead-of-openssl
  ;; Remplace tous les paquets nommés « openssl » par LibreSSL.
  (package-input-rewriting/spec `(("openssl" . ,(const libressl)))))
```

Le différence clef est que, cette fois-ci, les paquets correspondent à la spécification et non à l'identité. En d'autres termes, tout paquet dans le graphe qui est appelé `openssl` sera remplacé.

Une procédure plus générique pour réécrire un graphe de dépendance d'un paquet est `package-mapping` : elle supporte n'importe quel changement dans les nœuds du graphe.

`package-mapping` *proc* [*cut?*] [*#:deep?* *#f*] [Procédure Scheme]

Renvoie une procédure qui, pour un paquet donné, applique *proc* à tous les paquets dont il dépend et renvoie le paquet résultant. La procédure arrête la récursion lorsque *cut?* renvoie true pour un paquet donné. Lorsque *deep?* est vrai, *proc* est également appliqué aux entrées implicites.

## 9.4 Écrire un manifeste

Les commandes `guix` vous permettent de spécifier des listes de paquets sur la ligne de commande. C'est pratique, mais en même temps que la ligne de commande devient plus longue et moins triviales, il devient plus pratique d'avoir la liste des paquets dans ce que nous appelons un *manifeste*. Un manifeste est une sorte de « nomenclature » qui définit un ensemble de paquets. Habituellement, vous trouverez un bout de code qui construit le manifeste, vous l'enregistrez dans un fichier, disons `manifest.scm` puis passerez ce fichier à l'option `-m` (ou `--manifest`) que de nombreuses commandes `guix` prennent en charge. Par exemple, voici ce à quoi un manifeste pour un ensemble de paquets simple peut ressembler :

```
;; Manifeste pour trois paquets.
(specifications->manifest '("gcc-toolchain" "make" "git"))
```

Une fois que vous avez le manifeste, vous pouvez le passer, par exemple, à `guix package` pour installer uniquement ces trois paquets dans votre profil (voir [profile-manifest], page 48) :

```
guix package -m manifest.scm
```

... ou vous pouvez le passer à `guix shell` (voir [shell-manifest], page 86) pour créer un environnement temporaire :

```
guix shell -m manifest.scm
```

... ou vous pouvez le passer à `guix pack` de la même manière (voir [pack-manifest], page 100). Vous pouvez garder un manifeste sous contrôle de version, le partager avec d'autres personnes pour qu'elles puissent facilement l'installer, etc.

Mais comment écrire votre premier manifeste ? Pour commencer, vous voudrez peut-être écrire un manifeste qui reflète ce que vous avez déjà dans un profil. Plutôt que de commencer par une feuille blanche, `guix package` peut générer un manifeste pour vous (voir [export-manifest], page 53) :

```
# Écrit dans « manifest.scm » un manifeste correspondant au profil
# par défaut, ~/.guix-profile.
guix package --export-manifest > manifest.scm
```

Ou vous voudrez peut-être « traduire » des arguments de la ligne de commande en un manifeste. Dans ce cas, `guix shell` peut aider (voir [shell-export-manifest], page 86) :

```
# Écrit un manifeste pour les paquets spécifiés sur la ligne de commande. ■
```

```
guix shell --export-manifest gcc-toolchain make git > manifest.scm
```

Dans les deux cas, l'option `--export-manifest` essaye de générer un manifeste fidèle ; en particulier, il prend en compte les options de transformation des paquets (voir Section 10.1.2 [Options de transformation de paquets], page 183).

**Remarque:** Les manifestes sont *symboliques* : ils font référence aux paquets des canaux *actuellement utilisés* (voir Chapitre 7 [Canaux], page 75). Dans l'exemple ci-dessus, `gcc-toolchain` peut se référer à la version 11 aujourd'hui, mais pourra se référer à la version 13 dans deux ans.

Si vous voulez « épinglez » votre environnement logiciel à une version et une variante spécifique d'un paquet, vous devez donner des informations supplémentaires : la liste des révisions des canaux utilisés, renvoyée par `guix describe`. Voir Section 7.3 [Répliquer Guix], page 76, pour plus de détails.

Une fois que vous avez votre premier manifeste, vous voudrez peut-être le personnaliser. Comme votre manifeste est du code, vous avez maintenant accès à toutes les interfaces de programmation de Guix !

Supposons que vous souhaitiez avoir un manifeste pour déployer une variante personnalisée de GDB, le débogueur de GNU, qui ne dépend pas de Guile, avec un autre paquet. En complétant l'exemple vu dans la section précédente (voir Section 9.3 [Définition de variantes de paquets], page 116), vous pouvez écrire un manifeste de ce type :

```
(use-modules (guix packages)
             (gnu packages gdb)                ;pour « gdb »
             (gnu packages version-control)) ;pour « git »

;; Définie une variante de GDB qui ne dépend pas de Guile.
(define gdb-sans-guile
  (package
    (inherit gdb)
    (inputs (modify-inputs (package-inputs gdb)
                          (delete "guile")))))

;; Renvoie un manifeste contenant ce paquet plus Git.
(packages->manifest (list gdb-sans-guile git))
```

Remarquez que dans cet exemple, le manifeste se réfère directement aux variables `gdb` et `git`, qui sont liées à un objet `package` (voir Section 9.2.1 [référence de package], page 108), au lieu d'appeler `specifications->manifest` pour rechercher les paquets par leur nom comme nous l'avons fait précédemment. La forme `use-modules` au début nous permet d'accéder à l'interface centrale des paquets (voir Section 9.2 [Définition des paquets], page 105) et aux modules qui définissent `gdb` et `git` (voir Section 9.1 [Modules de paquets], page 104). Nous combinons tout cela harmonieusement — les possibilités sont infinies, débridez votre créativité !

Le type de données pour les manifestes et les procédures auxiliaires sont définis dans le module `(guix profiles)`, qui est automatiquement disponible pour le code passé à `-m`. Voici la documentation de référence.

**manifest** [Type de données]

Type de données représentant un manifeste.

Il a actuellement un seul champ :

**entries** Ce doit être une liste d'enregistrements **manifest-entry** — voir plus bas.

**manifest-entry** [Type de données]

Type de donnée représentant une entrée d'un manifeste. Une entrée de manifeste contient les métadonnées nécessaires : un nom et une chaîne de version, l'objet (généralement un paquet) pour cette entrée, la sortie souhaitée (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59) et un certain nombre d'autres informations facultatives décrites plus bas.

La plupart du temps, vous n'aurez pas à construire une entrée de manifeste directement ; au lieu de cela, vous passerez un paquet à **package->manifest-entry**, décrit plus bas. Dans certains cas rares cependant, vous pourriez être amené à créer des entrées de manifeste pour des objets qui ne sont pas des paquets, comme dans cet exemple :

```
;; Construit manuellement une entrée de manifeste pour un objet qui n'est pas un
(let ((hello (program-file "hello" #~(display "Hi!"))))
  (manifest-entry
   (name "toto")
   (version "42")
   (item
    (computed-file "hello-directory"
                   #~(let ((bin (string-append #$/bin)))
                       (mkdir #$/output) (mkdir bin)
                       (symlink #$/hello
                                (string-append bin #$/hello)))))))
```

Les champs disponibles sont les suivants :

**name**

**version** Nom et chaîne de version pour cette entrée.

**item** Un paquet ou n'importe quel objet simili-fichier (voir Section 9.12 [G-Expressions], page 166).

**output** (par défaut : "out")

Sortie de **item** à utiliser, dans le cas où **item** a plusieurs sorties (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59).

**dependencies** (par défaut : '())

Liste des entrées de manifeste desquelles dépend cette entrée. Lorsqu'un profil est construit, les dépendances sont ajoutées au profil.

Généralement, les entrées propagées d'un paquet (voir Section 9.2.1 [référence de package], page 108) ont une entrée de manifeste correspondante parmi les dépendances de l'entrée du paquet.

**search-paths** (par défaut : '())

La liste des spécifications de chemins de recherche que cette entrée prend en compte (voir Section 9.8 [Chemins de recherche], page 154).

`properties` (par défaut : '() )

Une liste de paires de symboles et de valeurs. Lorsqu'un profil est construit, ces propriétés sont sérialisées.

Cela peut être utilisé pour intégrer des métadonnées supplémentaires — p. ex. les transformations appliquées à un paquet (voir Section 10.1.2 [Options de transformation de paquets], page 183).

`parent` (par défaut : (delay #f) )

Une promesse pointant vers l'entrée de manifeste « parente ».

cela est utilisé comme une indication pour fournir du contexte si une erreur liée à une entrée de manifeste provenant du champ `dependencies` doit être rapportée.

`concatenate-manifests lst` [Procédure Scheme]

Concatène les manifestes listés dans *lst* et renvoie le manifeste qui en résulte.

`package->manifest-entry paquet [sortie] [#:properties]` [Procédure Scheme]

Renvoie une entrée de manifeste pour la *sortie* du paquet *paquet*, où *sortie* vaut "out" par défaut, et avec les propriétés *properties* données. Par défaut *properties* est la liste vide ou, si des transformations de paquets sont appliquées à *paquet*, une liste d'association représentant ces transformations, utilisable en tant qu'argument à `options->transformation` (voir Section 9.3 [Définition de variantes de paquets], page 116).

Le bout de code ci-dessous construit un manifeste avec une entrée pour la sortie par défaut et la sortie `send-email` du paquet `git` :

```
(use-modules (gnu packages version-control))

(manifest (list (package->manifest-entry git)
                (package->manifest-entry git "send-email")))
```

`packages->manifest paquets` [Procédure Scheme]

Renvoie une liste d'entrées de manifestes, un paquet élément listé dans *paquets*. Les éléments de *paquets* peuvent être soit des objets paquets soit des tuples paquet / chaîne dénotant une sortie particulière d'un paquet.

Avec cette procédure, le manifeste ci-dessus peut être réécrit de manière plus concise :

```
(use-modules (gnu packages version-control))

(packages->manifest (list git `(,git "send-email")))
```

`package->development-manifest paquet [système]` [Procédure Scheme]

`[:target]`

Renvoie un manifeste pour les *entrées de développement* de *paquet* pour *système*, éventuellement pour la compilation croisée vers la cible *target*. Les entrées de développement comprennent à la fois les entrées explicites et les entrées implicites de *paquet*.

Comme l'option `-D` de `guix shell` (voir [shell-development-option], page 85), le manifeste qui en résulte décrit l'environnement dans lequel on peut développer le *paquet*.

Par exemple, si vous voulez configurer un environnement de développement pour Inkscape, avec Git pour le contrôle de version, vous pouvez décrire cette « nomenclature » avec le manifeste suivant :

```
(use-modules (gnu packages inkscape)           ;pour « inkscape »
             (gnu packages version-control)) ;pour « git »

(concatenate-manifests
 (list (package->development-manifest inkscape)
       (packages->manifest (list git))))
```

Dans cet exemple, le manifeste de développement qui renvoie `package->development-manifest` inclut le compilateur (GCC), les nombreuses bibliothèques qui portent le projet (Boost, GLib, GTK, etc) et quelques outils de développement supplémentaires — ce sont les dépendances listées par `guix show inkscape`.

Enfin, le module `(gnu packages)` fournit des fonctions pour construire des manifestes. En particulier, il vous permet de rechercher des paquets par leur nom — voir ci-dessous.

`specifications->manifest specs` [Procédure Scheme]

Étant données les *specs*, une liste de spécifications comme `"emacs@25.2"` ou `"guile:debug"`, renvoie un manifeste. Les spécifications ont le même format que celui attendu par les outils en ligne de commande comme `guix install` et `guix package` (voir Section 6.2 [Invoquer `guix package`], page 44).

Par exemple, elle vous permet de réécrire le manifeste avec Git que nous avons vu plus haut comme ceci :

```
(specifications->manifest ("git" "git:send-email"))
```

Remarque que nous n'avons pas à nous soucier de `use-modules`, d'importer le bon ensemble de modules et de se référer aux bonnes variables. À la place, nous nous référons directement aux paquets de la même manière que sur la ligne de commande, ce qui peut souvent être plus pratique.

## 9.5 Systèmes de construction

Chaque définition de paquet définit un *système de construction* et des arguments pour ce système de construction (voir Section 9.2 [Définition des paquets], page 105). Ce champ `build-system` représente la procédure de construction du paquet, ainsi que des dépendances implicites pour cette procédure de construction.

Les systèmes de construction sont des objets `<build-system>`. L'interface pour les créer et les manipuler est fournie par le module `(guix build-system)` et les systèmes de construction eux-mêmes sont exportés par des modules spécifiques.

Sous le capot, les systèmes de construction compilent d'abord des objets paquets en sacs. Un sac est comme un paquet, mais avec moins de décoration — en d'autres mots, un sac est une représentation à bas-niveau d'un paquet, qui inclut toutes les entrées de ce paquet, dont certaines ont été implicitement ajoutées par le système de construction. Cette représentation intermédiaire est ensuite compilée en une dérivation (voir Section 9.10 [Dérivations], page 159). Le `package-with-c-toolchain` est un exemple d'une manière de modifier les entrées implicites que le système de construction d'un paquet récupère (voir Section 9.2.1 [référence de package], page 108).

Les systèmes de construction acceptent une liste d'*arguments* facultatifs. Dans les définitions de paquets, ils sont passés *via* le champ `arguments` (voir Section 9.2 [Définition des paquets], page 105). Ce sont typiquement des arguments par mot-clef (voir Section “Optional Arguments” dans *GNU Guile Reference Manual*). La valeur de ces arguments est habituellement évaluée dans la *strate de construction* — c.-à-d. par un processus Guile lancé par le démon (voir Section 9.10 [Dérivations], page 159).

Le système de construction principal est le `gnu-build-system` qui implémente les procédures de construction standard pour les paquets GNU et de nombreux autres. Il est fourni par le module `(guix build-system gnu)`.

`gnu-build-system` [Variable Scheme]

`gnu-build-system` représente le système de construction GNU et ses variantes (voir Section “Configuration” dans *GNU Coding Standards*).

En résumé, les paquets qui l'utilisent sont configurés, construits et installés avec la séquence `./configure && make && make check && make install` habituelle. En pratique, des étapes supplémentaires sont souvent requises. Toutes ces étapes sont séparées dans des *phases* différentes, Voir Section 9.6 [Phases de construction], page 143, pour plus d'informations sur les phases de construction et comment les personnaliser.

En plus, ce système de construction s'assure que l'environnement « standard » pour les paquets GNU est disponible. Cela inclus des outils comme GCC, libc, Coreutils, Bash, Make, Diffutils, grep et sed (voir le module `(guix build-system gnu)` pour une liste complète). Nous les appelons les *entrées implicites* d'un paquet parce que la définition du paquet ne les mentionne pas.

Ce système de construction prend en charge un certain nombre de mot-clés, qui peuvent être passés via le champ `arguments` d'un paquet. Voici certains des paramètres principaux :

`#:phases` Cet argument spécifie le code du côté de la construction qui s'évalue en une liste d'association des phases de construction. Voir Section 9.6 [Phases de construction], page 143, pour plus d'informations.

`#:configure-flags`  
C'est une liste de drapeaux (chaînes) passés au script `configure`. Voir Section 9.2 [Définition des paquets], page 105, pour un exemple.

`#:make-flags`  
Cette liste de chaînes contient les drapeaux passés en argument aux invocations de `make` dans les phases `build`, `check` et `install`.

`#:out-of-source?`  
Ce booléen, `#f` par défaut, indique s'il faut lancer les constructions dans un répertoire séparé de l'arborescence des sources.

Lorsque la valeur est vraie, la phase `configure` crée un répertoire de construction séparé, se déplace dedans et lance le script `configure` à partir de celui-ci. C'est utile pour les paquets qui en ont besoin, comme `glibc`.

`#:tests?` Ce booléen, `#t` par défaut, indique si la phase `check` doit lancer la suite de tests du paquet.



**#:test-target**

Cette chaîne, "check" par défaut, donne le nom de la cible du makefile à utiliser dans la phase `check`.

**#:parallel-build?****#:parallel-tests?**

Ces booléens spécifient s'il faut construire, respectivement lancer la suite de tests, en parallèle, avec le drapeau `-j` de `make`. Lorsque la valeur est vraie, `make` reçoit `-jn`, où `n` est le nombre spécifié dans l'option `--cores` de `guix-daemon` ou celle de la commande du client `guix` (voir Section 10.1.1 [Options de construction communes], page 180).

**#:validate-runpath?**

Ce booléen, `#t` par défaut, détermine s'il faut « valider » le `RUNPATH` des binaires ELF (les bibliothèques partagées `.so` et les exécutables) installés précédemment par la phase `install`. Voir [phase-validate-runpath], page 144, pour plus de détails.

**#:substitutable?**

Ce booléen, `#t` par défaut, indique si les sorties du paquet peuvent être substituées — c.-à-d. si les utilisateur·rices peuvent obtenir des substituts au lieu de les construire localement (voir Section 6.3 [Substituts], page 54).

**#:allowed-references****#:disallowed-references**

Lorsque la valeur est vraie, ces arguments doivent être une liste de dépendance qui ne doivent pas apparaître dans les dépendances des résultats de la construction. Si, à la fin de la construction, certaines de ces références sont retenues, le processus échoue.

C'est utile pour s'assurer qu'un paquet ne garde pas une référence par erreur à ses entrées de construction, sis cela augmente par exemple la taille de sa clôture (voir Section 10.9 [Invoquer `guix size`], page 215).

La plupart des autres systèmes de construction prennent en charge ces mot-clés.

D'autres objets `<build-system>` sont définis pour supporter d'autres conventions et outils utilisés par les paquets de logiciels libres. Ils héritent de la plupart de `gnu-build-system` et diffèrent surtout dans l'ensemble des entrées implicites ajoutées au processus de construction et dans la liste des phases exécutées. Certains de ces systèmes de construction sont listés ci-dessous.

**ant-build-system**

[Variable Scheme]

Cette variable est exportée par (`guix build-system ant`). Elle implémente la procédure de construction pour les paquets Java qui peuvent être construits avec l'outil de construction Ant (<https://ant.apache.org/>).

Elle ajoute à la fois `ant` et the *kit de développement Java* (JDK) fournit par le paquet `icedtea` à l'ensemble des entrées. Des paquets différents peuvent être spécifiés avec les paramètres `#:ant` et `#:jdk` respectivement.

Lorsque le paquet d'origine ne fournit pas de fichier de construction Ant acceptable, le paramètre `#:jar-name` peut être utilisé pour générer un fichier de construction Ant

`build.xml` minimal, avec des tâches pour construire l'archive jar spécifiée. Dans ce cas, le paramètre `#:source-dir` peut être utilisé pour spécifier le sous-répertoire des sources, par défaut « `src` ».

Le paramètre `#:main-class` peut être utilisé avec le fichier de construction minimal pour spécifier la classe principale du jar. Cela rend le fichier jar exécutable. Le paramètre `#:test-include` peut être utilisé pour spécifier la liste des tests junit à lancer. Il vaut par défaut (`list "**/*Test.java"`). Le paramètre `#:test-exclude` peut être utilisé pour désactiver certains tests. Sa valeur par défaut est (`list "**/Abstract*.java"`), parce que les classes abstraites ne peuvent pas être utilisées comme des tests.

Le paramètre `#:build-target` peut être utilisé pour spécifier la tâche Ant qui devrait être lancée pendant la phase `build`. Par défaut la tâche « `jar` » sera lancée.

**android-ndk-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system android-ndk`). Elle implémente une procédure de construction pour les paquets du NDK Android (*native development kit*) avec des processus de construction spécifiques à Guix.

Le système de compilation suppose que les paquets installent leurs fichiers d'interface publique (en-tête) dans le sous-répertoire `include` de la sortie `out` et leurs bibliothèques dans le sous-répertoire `lib` de la sortie `out`.

Il est aussi supposé que l'union de toutes les dépendances d'un paquet n'a pas de fichiers en conflit.

Pour l'instant, la compilation croisée n'est pas supportées — donc pour l'instant les bibliothèques et les fichiers d'en-têtes sont supposés être des outils de l'hôte.

**asdf-build-system/source** [Variable Scheme]

**asdf-build-system/sbcl** [Variable Scheme]

**asdf-build-system/ecl** [Variable Scheme]

Ces variables, exportées par (`guix build-system asdf`), implémentent les procédures de constructions pour les paquets en Common Lisp qui utilisent « ASDF » (<https://common-lisp.net/project/asdf/>). ASDF est un dispositif de définition de systèmes pour les programmes et les bibliothèques en Common Lisp.

Le système `asdf-build-system/source` installe les paquets au format source qui peuvent être chargés avec n'importe quelle implémentation de common lisp, via ASDF. Les autres, comme `asdf-build-system/sbcl`, installent des binaires au format qu'un implémentation particulière comprend. Ces systèmes de constructions peuvent aussi être utilisés pour produire des programmes exécutables ou des images lisp qui contiennent un ensemble de paquets pré-chargés.

Le système de construction utilise des conventions de nommage. Pour les paquets binaires, le nom du paquet devrait être préfixé par l'implémentation lisp, comme `sbcl-` pour `asdf-build-system/sbcl`.

En plus, le paquet source correspondant devrait être étiqueté avec la même convention que les paquets python (voir Section 22.4.7 [Modules Python], page 678), avec le préfixe `cl-`.

Pour créer des programmes exécutables et des images, les procédures côté construction `build-program` et `build-image` peuvent être utilisées. Elles devraient être appelées

dans une phase de construction après la phase `create-asdf-configuration` pour que le système qui vient d'être construit puisse être utilisé dans l'image créée. `build-program` requiert une liste d'expressions Common Lisp dans l'argument `#:entry-program`.

Par défaut, tous les fichiers `.asd` présents dans les sources sont lus pour trouver les définitions du système. Le paramètre `#:asd-file` peut être utilisé pour préciser la liste des fichiers `.asd` à lire. En outre, si le paquet définit un système pour ses tests dans un fichier séparé, il sera chargé avant l'exécution des tests s'il est spécifié par le paramètre `#:test-asd-file`. S'il n'est pas défini, les fichiers `<system>-tests.asd`, `<system>-test.asd`, `tests.asd` et `test.asd` seront essayés s'ils existent.

Si pour quelque raison que ce soit le paquet doit être nommé d'une manière différente de ce que la convention de nommage suggère, ou si plusieurs systèmes doivent être compilés, le paramètre `#:asd-systems` peut être utilisé pour spécifier la liste des noms de systèmes.

### `cargo-build-system` [Variable Scheme]

Cette variable est exportée par `(guix build-system cargo)`. Elle supporte les constructions de paquets avec Cargo, le système de construction du langage de programmation Rust (<https://www.rust-lang.org>).

Cela ajoute `rustc` et `cargo` à l'ensemble des entrées. Un autre paquet Rust peut être spécifié avec le paramètre `#:rust`.

Les dépendances régulières de cargo doivent être ajoutées à la définition du paquet comme avec les autres paquets ; celles qui ne sont requises qu'à la construction dans « `native-inputs` », les autres dans « `inputs` ». Si vous devez ajouter des crates sources alors vous devez les ajouter via le paramètre `#:cargo-inputs` sous la forme d'une liste de paires de noms et de spécifications, où la spécification peut être un paquet ou une définition de source. Notez que la spécification doit évaluer un chemin vers une archive gzippée qui inclut un fichier `Cargo.toml` à sa racine, sinon elle sera ignorée. De même, les dépendances de développement de cargo doivent être ajoutées à la définition du paquet via le paramètre `#:cargo-development-inputs`.

Dans sa phase `configure`, ce système de construction mettra à disposition de cargo toutes les entrées sources spécifiées dans les paramètres `#:cargo-inputs` et `#:cargo-development-inputs`. Il supprimera également un fichier `Cargo.lock` inclus, qui sera recréé par `cargo` pendant la phase `build`. La phase `package` lancera `cargo package` pour créer un crate source réutilisable plus tard. La phase `install` installe les binaires définis par le crate. À moins de définir `install-source? #f` elle installera aussi un crate source et les sources décompressées pour faciliter le travail avec les paquets rust.

### `chicken-build-system` [Variable Scheme]

Cette variable est exportée par `(guix build-system chicken)`. Elle construit des modules CHICKEN Scheme (<https://call-cc.org/>), aussi appelés « `eggs` » ou « `extensions` ». CHICKEN génère du code source C, qui est ensuite compilé par un compilateur C, dans ce cas GCC.

Ce système de construction ajoute `chicken` aux entrées du paquet, en plus des paquets de `gnu-build-system`.

Le système de construction ne peut pas (encore) déterminer le nom de l'egg automatiquement, donc comme avec le `go-build-system` et son `#:import-path`, vous devriez définir `#:egg-name` dans le champ `arguments` du paquet.

Par exemple, si vous créez un paquet pour l'egg `srfi-1` :

```
(arguments '(:egg-name "srfi-1"))
```

Les dépendances des egg doivent être définies dans `propagated-inputs`, pas `inputs` parce que CHICKEN n'inclut pas de références absolues dans les eggs compilés. Les dépendances des tests doivent aller dans `native-inputs`, comme d'habitude.

### `copy-build-system` [Variable Scheme]

Cette variable est exportée par (`guix build-system copy`). Il prend en charge la construction de paquets simples qui ne nécessitent pas beaucoup de compilation, la plupart du temps juste le déplacement de fichiers.

Cela ajoute une grande partie des paquets `gnu-build-system` à l'ensemble des entrées. De ce fait, le `copy-build-system` n'a pas besoin de tout le code passe-partout souvent nécessaire pour le `trivial-build-system`.

Pour simplifier davantage le processus d'installation des fichiers, un argument `#:install-plan` est exposé pour permettre au packager de spécifier quels fichiers vont où. Le plan d'installation est une liste de (`source cible [filtres]`). Les *filtres* sont facultatifs.

- Lorsque *source* correspond à un fichier ou à un répertoire sans barre oblique, installez-le sur *cible*.
  - Si *target* a une barre oblique, installez le nom de base *source* sous *target*.
  - Sinon, installez *source* comme *cible*.
- Lorsque *source* est un répertoire avec une barre oblique, ou lorsque des *filtres* sont utilisés, la barre oblique de *target* est implicite avec la même signification que ci-dessus.
  - Sans *filtres*, installez le *source* complet *contenu* à *cible*.
  - Avec *filtres* parmi `#:include`, `#:include-regex`, `#:exclude`, `#:exclude-regex`, seuls les fichiers sélectionnés sont installés en fonction des filtres. Chaque filtre est spécifié par une liste de chaînes de caractères.
    - Avec `#:include`, installez tous les fichiers dont le suffixe de chemin correspond au moins un des éléments dans la liste donnée.
    - Avec `#:include-regex`, installez tous les fichiers que le les sous-chemins correspondent à au moins une des expressions régulières de la liste donnée.
    - Les filtres `#:exclude` et `#:exclude-regex` sont le complément de leur homologue pour l'inclusion. Sans les drapeaux `#:include`, installez tous les fichiers sauf ceux qui correspondent aux filtres d'exclusion. Si les inclusions et les exclusions sont toutes deux spécifiées, les exclusions sont faites en complément des inclusions.

Dans tous les cas, les chemins relatifs à *source* sont préservés dans *cible*.

Exemples :

- `("toto/titi" "share/mon-app/")` : Installer titi sur share/mon-app/titi.

- (`"toto/titi" "share/mon-app/tata"`) : Installer `titi` sur `share/mon-app/tata`.
- (`"toto/" "share/mon-app"`) : Installez le contenu de `toto` dans `share/mon-app`, par exemple, installe `toto/sub/file` dans `share/mon-app/sub/file`.
- (`"toto/" "share/mon-app" #:include ("sub/file")`) : Installe seulement `toto/sub/file` vers `share/mon-app/sub/file`.
- (`"toto/sub" "share/mon-app" #:include ("file")`) : Installe `toto/sub/file` vers `share/mon-app/file`.

### `clojure-build-system` [Variable Scheme]

Cette variable est exportée par (`guix build-system clojure`). Elle implémente une procédure de construction des paquets simple qui utilise le bon vieux `compile` de Clojure. La compilation croisée n'est pas encore supportée.

Elle ajoute `clojure`, `icedtea` et `zip` à l'ensemble des entrées. Des paquets différents peuvent être spécifiés avec les paramètres `#:clojure`, `#:jdk` et `#:zip`.

Une liste de répertoires sources, de répertoires de tests et de noms de jar peuvent être spécifiés avec les paramètres `#:source-dirs`, `#:test-dirs` et `#:jar-names`. Le répertoire de construction est la classe principale peuvent être spécifiés avec les paramètres `#:compile-dir` et `#:main-class`. Les autres paramètres sont documentés plus bas.

Ce système de construction est une extension de `ant-build-system`, mais avec les phases suivantes modifiées :

**build** Cette phase appelle `compile` en Clojure pour compiler les fichiers sources et lance `jar` pour créer les fichiers jar à partir des fichiers sources et des fichiers compilés en suivant la liste d'inclusion et d'exclusion spécifiées dans `#:aot-include` et `#:aot-exclude`. La liste d'exclusion a la priorité sur la liste d'inclusion. Ces listes consistent en des symboles représentant des bibliothèques Clojure ou le mot clef spécial `#:all`, représentant toutes les bibliothèques Clojure trouvées dans les répertoires des sources. Le paramètre `#:omit-source?` décide si les sources devraient être incluses dans les fichiers jar.

**check** Cette phase lance les tests en suivant les liste d'inclusion et d'exclusion spécifiées dans `#:test-include` et `#:test-exclude`. Leur signification est analogue à celle de `#:aot-include` et `#:aot-exclude`, sauf que le mot-clef spécial `#:all` signifie maintenant toutes les bibliothèques Clojure trouvées dans les répertoires de tests. Le paramètre `#:tests?` décide si les tests devraient être lancés.

**install** Cette phase installe tous les fichiers jar précédemment construits.

En dehors de cela, le système de construction contient aussi la phase suivante :

#### `install-doc`

Cette phase installe tous les fichiers dans le répertoire de plus haut niveau dont le nom correspond à `%doc-regex`. On peut spécifier une regex différente avec le paramètre `#:doc-regex`. Tous les fichiers (récursivement) dans les répertoires de documentations spécifiés dans `#:doc-dirs` sont aussi installés.

**cmake-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system cmake`). Elle implémente la procédure de construction des paquets qui utilisent l'outil de construction CMake (<https://www.cmake.org>).

Elle ajoute automatiquement le paquet `cmake` à l'ensemble des entrées. Le paquet utilisé peut être spécifié par le paramètre `#:cmake`.

Le paramètre `#:configure-flags` est pris comme une liste de drapeaux à passer à la commande `cmake`. Le paramètre `#:build-type` spécifie en termes abstraits les drapeaux passés au compilateur ; sa valeur par défaut est `"RelWithDebInfo"` (ce qui veut dire « mode public avec les informations de débogage » en plus court), ce qui signifie en gros que le code sera compilé avec `-O2 -g` comme pour les paquets autoconf par défaut.

**dune-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system dune`). Elle prend en charge la construction des paquets qui utilisent Dune (<https://dune.build/>), un outil de construction pour le langage de programmation OCaml. Elle est implémentée comme une extension de `ocaml-build-system` décrit plus bas. En tant que tel, les paramètres `#:ocaml` et `#:findlib` peuvent être passés à ce système de construction.

Elle ajoute automatiquement le paquet `dune` à l'ensemble des entrées. Le paquet utilisé peut être spécifié par le paramètre `#:dune`.

Il n'y a pas de phase `configure` parce que les paquets `dune` n'ont habituellement pas besoin d'être configurés. Le paramètre `#:build-flags` est interprété comme une liste de drapeaux pour la commande `dune` pendant la construction.

Le paramètre `#:jbuild?` peut être passé pour utiliser la commande `jbuild` à la place de la commande `dune` plus récente pour la construction d'un paquet. Sa valeur par défaut est `#f`.

Le paramètre `#:package` peut être passé pour spécifier un nom de paquet, ce qui est utile lorsqu'un paquet contient plusieurs paquets et que vous voulez n'en construire qu'un. C'est équivalent à passer l'argument `-p` à `dune`.

**elm-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system elm`). Elle implémente une procédure de construction pour les paquets Elm (<https://elm-lang.org>) qui ressemble à `'elm install'`.

Le système de construction ajoute un paquet qui contient le compilateur Elm à l'ensemble des entrées. Le paquet de compilateur par défaut (actuellement `elm-sans-reactor`) peut être remplacé avec l'argument `#:elm`. En plus, les paquets Elm requis par le système de construction lui-même sont ajoutés en tant qu'entrées implicites s'ils ne sont pas déjà présents : pour supprimer ce comportement, utilisez l'argument `#:implicit-elm-package-inputs?` qui est surtout utile pour l'armorçage.

Les `"dependencies"` et `"test-dependencies"` du fichier `elm.json` d'un paquet Elm correspondent à `propagated-inputs` et `inputs`, respectivement.

Elm a besoin d'une structure particulière pour les noms des paquets : voir Section 22.4.11 [Paquets elm], page 680, pour plus de détails, en particulier sur les outils fournis par (`guix build-system elm`).

Il y a actuellement quelques limites notables à `elm-build-system` :

- Le système de construction se concentre sur les *paquets* au sens d'Elm : les *projets* Elm qui déclarent `{ "type": "package" }` dans leur fichier `elm.json`. Utiliser `elm-build-system` pour construire des *applications* Elm (qui déclarent `{ "type": "application" }`) est possible, mais nécessite des modifications ad-hoc des phases de construction. Par exemple, voir les définitions de l'application d'exemple `elm-todomvc` et du paquet `elm` lui-même (parce que l'interface de la commande `'elm reactor'` est une application Elm).
- Elm permet la coexistence simultanée de plusieurs versions d'un paquet dans `ELM_HOME`, mais cela ne fonctionne pas encore bien avec `elm-build-system`. Cette limite affecte principalement les applications Elm, parce qu'elles spécifient les versions exactes de leurs dépendances, alors que les paquets Elm spécifient des intervalles de versions prises en charge. Pour contourner le problème, les applications d'exemple mentionnées plus haut utilisent la procédure `patch-application-dependencies` fournie par `(guix build elm-build-system)` pour réécrire leur fichier `elm.json` pour qu'ils se réfèrent aux versions effectivement présentes dans l'environnement de construction. Autrement, les transformations des paquets Guix (voir Section 9.3 [Définition de variantes de paquets], page 116) peuvent aussi réécrire tout le graphe de dépendance d'une application.
- Nous ne pouvons pas encore lancer les tests des projets Elm parce que ni `elm-test-rs` (<https://github.com/mpizenberg/elm-test-rs>) ni l'exécuteur `elm-test` (<https://github.com/rtfeldman/node-test-runner>) basé sur Node.js ne sont empaquetés dans Guix.

### `go-build-system`

[Variable Scheme]

Cette variable est exportée par `(guix build-system go)`. Elle implémente la procédure pour les paquets Go utilisant les mécanismes de construction Go ([https://golang.org/cmd/go/#hdr-Compile\\_packages\\_and\\_dependencies](https://golang.org/cmd/go/#hdr-Compile_packages_and_dependencies)) standard.

L'utilisateur doit fournir une valeur à la clef `#:import-path` et, dans certains cas, `#:unpack-path`. Le chemin d'import (<https://golang.org/doc/code.html#ImportPaths>) correspond au chemin dans le système de fichiers attendu par le script de construction du paquet et les paquets qui s'y réfèrent et fournit une manière unique de se référer à un paquet Go. Il est typiquement basé sur une combinaison de l'URI du code source du paquet et d'une structure hiérarchique du système de fichier. Dans certains cas, vous devrez extraire le code source du paquet dans une structure de répertoires différente que celle indiquée par le chemin d'import et `#:unpack-path` devrait être utilisé dans ces cas-là.

Les paquets qui fournissent des bibliothèques Go devraient installer leur code source dans la sortie du paquet. La clef `#:install-source?`, qui vaut `#t` par défaut, contrôle l'installation du code source. Elle peut être mise à `#f` pour les paquets qui ne fournissent que des fichiers exécutables.

Vous pouvez effectuer une compilation croisée des paquets et si vous voulez une architecture ou un système d'exploitation spécifique alors vous pouvez utiliser les mot-clés `#:goarch` et `#:goos` pour forcer le paquet à être construit pour l'architecture et le

système d'exploitation donnés. Vous trouverez les combinaisons connues pour Go dans leur documentation ("<https://golang.org/doc/install/source#environment>").

**glib-or-gtk-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system glib-or-gtk`). Elle est conçue pour être utilisée par des paquets qui utilisent GLib ou GTK+.

Ce système de construction ajoute les deux phases suivantes à celles définies par `gnu-build-system` :

**glib-or-gtk-wrap**

La phase `glib-or-gtk-wrap` garantit que les programmes dans `bin/` sont capables de trouver les « schémas » GLib et GTK+ modules (<https://developer.gnome.org/gtk3/stable/gtk-running.html>).

Cela est réalisé en enveloppant les programmes dans des scripts de lancement qui définissent de manière appropriée les variables d'environnement `XDG_DATA_DIRS` et `GTK_PATH`.

Il est possible d'exclure des sorties spécifiques de ce processus d'enveloppement en listant leur nom dans le paramètre `#:glib-or-gtk-wrap-excluded-outputs`. C'est utile lorsqu'une sortie est connue pour ne pas contenir de binaires GLib ou GTK+, et où l'enveloppe ajouterait une dépendance inutile vers GLib et GTK+.

**glib-or-gtk-compile-schemas**

La phase `glib-or-gtk-compile-schemas` s'assure que tous les schémas GSettings (<https://developer.gnome.org/gio/stable/glib-compile-schemas.html>) de GLib sont compilés. La compilation est effectuée par le programme `glib-compile-schemas`. Il est fourni par le paquet `glib:bin` qui est automatiquement importé par le système de construction. Le paquet `glib` qui fournit `glib-compile-schemas` peut être spécifié avec le paramètre `#:glib`.

Ces deux phases sont exécutées après la phase `install`.

**guile-build-system** [Variable Scheme]

Ce système de construction sert aux paquets Guile qui consistent exclusivement en code Scheme et qui sont si simple qu'ils n'ont même pas un `makefile`, sans parler d'un script `configure`. Il compile le code Scheme en utilisant `guild compile` (voir Section "Compilation" dans *GNU Guile Reference Manual*) et installe les fichiers `.scm` et `.go` aux bons emplacements. Il installe aussi la documentation.

Ce système de construction supporte la compilation croisée en utilisant l'option `--target` de `guild compile`.

Les paquets construits avec `guile-build-system` doivent fournir un paquet Guile dans leur champ `native-inputs`.

**julia-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system julia`). Elle implémente la procédure de compilation utilisée par les paquets `julia` (<https://julialang.org/>), qui est globalement similaire à l'exécution de `'julia -e 'using Pkg ; Pkg.add(package)''` dans un environnement où `JULIA_LOAD_PATH` contient les



chemins de toutes les entrées des paquets Julia. Les tests sont effectués en appelant `/test/runtests.jl`.

Le nom et l'uuid du paquet Julia est lu dans le fichier `Project.toml`. Ces valeurs peuvent être remplacées en passant l'argument `#:julia-package-name` (le nom doit être correctement capitalisé) ou `#:julia-package-uuid`.

Les paquets Julia gèrent généralement leurs dépendances binaires via `JLLWrappers.jl`, un paquet Julia qui crée un module (nommé d'après la bibliothèque enveloppée suivie de `_jll.jl`).

Pour ajouter le chemin vers les binaires `_jll.jl`, vous devez corriger les fichiers dans `src/wrappers/`, en remplaçant l'appel à la macro `JLLWrappers.@generate_wrapper_header`, en ajoutant un second argument contenant le chemin du binaire dans le dépôt.

Par exemple, dans le paquets Julia MbedTLS, on ajoute un phase de construction (voir Section 9.6 [Phases de construction], page 143) pour insérer le nom de fichier absolu du paquet enveloppé MbedTLS :

```
(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")))))
  ;; Il y a un fichier Julia pour chaque plateforme, on les remplace
  (find-files "src/wrappers/" "\\_jll$"))))
```

Certains paquets plus anciens qui n'utilisent pas encore `Project.toml` nécessiteront aussi la création de ce fichier. Cela s'effectue en interne si les arguments `#:julia-package-name` et `#:julia-package-uuid` sont fournis.

**maven-build-system** [Variable Scheme]

Cette variable est exportée par `(guix build-system maven)`. Elle implémente une procédure de construction pour les paquets Maven (<https://maven.apache.org>). Maven est un outil de gestion des dépendances et du cycle de vie pour Java. Un-e utilisateur-riche de Maven spécifie les dépendances et les plugins dans un fichier `pom.xml` que Maven lit. Lorsque Maven n'a pas l'une des dépendances ou l'un des plugins dans son dépôt, il les télécharge et les utilise pour construire le paquet.

Le système de construction de maven garantit que maven n'essaiera pas de télécharger une dépendance en s'exécutant en mode hors ligne. Maven échouera si une dépendance est manquante. Avant de lancer Maven, le fichier `pom.xml` (et sous-projets) sont modifiés pour spécifier la version des dépendances et des plugins qui correspondent aux versions disponibles dans l'environnement de construction de guix. Les dépendances et plugins doivent être installés dans le faux dépôt maven à `lib/m2`, et sont liés par un lien symbolique à un dépôt approprié avant que maven ne soit lancé. Maven a pour instruction d'utiliser ce dépôt pour la construction et y installe les artefacts construits. Les fichiers modifiés sont copiés dans le répertoire `lib/m2` de la sortie du paquet.

Vous pouvez spécifier un fichier `pom.xml` avec l'argument `#:pom-file`, ou bien laisser le système de construction utiliser le fichier par défaut `pom.xml` dans les sources.

Dans le cas où vous avez besoin de spécifier manuellement une version de dépendances, vous pouvez utiliser l'argument `#:local-packages`. Il prend une liste d'association dont la clé est le `groupId` du paquet et sa valeur est une liste d'association où la clé est l'`artifactId` du paquet et sa valeur est la version que vous souhaitez remplacer dans le fichier `pom.xml`.

Certains paquets utilisent des dépendances ou des plugins qui ne sont pas utiles au moment de l'exécution ou de la compilation dans Guix. Vous pouvez modifier le fichier `pom.xml` pour les supprimer en utilisant l'argument `#:exclude`. Sa valeur est une liste d'association où la clé est le `groupId` du plugin ou de la dépendance que vous voulez supprimer, et la valeur est une liste d'`artifactId` que vous voulez supprimer.

Vous pouvez remplacer les paquets `jdk` et `maven` par défaut avec l'argument correspondant, `#:jdk` et `#:maven`.

L'argument `#:maven-plugins` est une liste de plugins maven utilisés pendant la construction, avec le même format que le champ `inputs` de la déclaration du paquet. Sa valeur par défaut est (`default-maven-plugins`) qui est également exportée.

#### `minetest-mod-build-system` [Variable Scheme]

Cette variable est exportée par (`guix build-system minetest`). Elle implémente une procédure de construction pour les mods de Minetest (<https://runwww.minetest.net>) qui consiste à copier du code Lua, des images et d'autres ressources aux emplacements où Minetest recherche des mods. Le système de construction minimise aussi les images PNG et vérifie que Minetest peut charger le mod sans erreur.

#### `minify-build-system` [Variable Scheme]

Cette variable est exportée par (`guix build-system minify`). Elle implémente une procédure de minification pour des paquets JavaScript simples.

Elle ajoute `uglify-js` à l'ensemble des entrées et l'utilise pour compresser tous les fichiers JavaScript du répertoire `src`. Un minifieur différent peut être spécifié avec le paramètre `#:uglify-js` mais il est attendu que ce paquet écrive le code minifié sur la sortie standard.

Lorsque les fichiers JavaScript d'entrée ne sont pas situés dans le répertoire `src`, le paramètre `#:javascript-files` peut être utilisé pour spécifier une liste de noms de fichiers à donner au minifieur.

#### `ocaml-build-system` [Variable Scheme]

Cette variable est exportée par (`guix build-system ocaml`). Elle implémente une procédure de construction pour les paquets OCaml (<https://ocaml.org>) qui consiste à choisir le bon ensemble de commande à lancer pour chaque paquet. Les paquets OCaml peuvent demander des commandes diverses pour être construit. Ce système de construction en essaye certaines.

Lorsqu'un fichier `setup.ml` est présent dans le répertoire de plus haut niveau, elle lancera `ocaml setup.ml -configure`, `ocaml setup.ml -build` et `ocaml setup.ml -install`. Le système de construction supposera que ces fichiers ont été générés par OASIS (<http://oasis.forge.ocamlcore.org/>) et prendra soin d'initialiser

le préfixe et d'activer les tests s'ils ne sont pas désactivés. Vous pouvez passer des drapeaux de configuration et de construction avec `#:configure-flags` et `#:build-flags`. La clef `#:test-flags` peut être passée pour changer l'ensemble des drapeaux utilisés pour activer les tests. La clef `#:use-make?` peut être utilisée pour outrepasser ce système dans les phases de construction et d'installation.

Lorsque le paquet a un fichier `configure`, il est supposé qu'il s'agit d'un script `configure` écrit à la main qui demande un format différent de celui de `gnu-build-system`. Vous pouvez ajouter plus de drapeaux avec la clef `#:configure-flags`.

Lorsque le paquet a un fichier `Makefile` (ou `#:use-make?` vaut `#t`), il sera utilisé et plus de drapeaux peuvent être passés à la construction et l'installation avec la clef `#:make-flags`.

Enfin, certains paquets n'ont pas ces fichiers mais utilisent un emplacement plus ou moins standard pour leur système de construction. Dans ce cas, le système de construction lancera `ocaml pkg/pkg.ml` ou `pkg/build.ml` et prendra soin de fournir le chemin du module `findlib` requis. Des drapeaux supplémentaires peuvent être passés via la clef `#:bulid-flags`. L'installation se fait avec `opam-installer`. Dans ce cas, le paquet `opam` doit être ajouté au champ `native-inputs` de la définition du paquet.

Remarquez que la plupart des paquets OCaml supposent qu'ils seront installés dans le même répertoire qu'OCaml, ce qui n'est pas ce que nous voulons faire dans Guix. En particulier, ils installeront leurs fichiers `.so` dans leur propre répertoire de module, ce qui est normalement correct puisqu'il s'agit du répertoire du compilateur OCaml. Dans Guix en revanche, ces bibliothèques ne peuvent pas y être trouvées et on utilise `CAML_LD_LIBRARY_PATH` à la place. Cette variable pointe vers `lib/ocaml/site-lib/stublibs` et c'est là où les bibliothèques `.so` devraient être installées.

`python-build-system` [Variable Scheme]

Cette variable est exportée par (`guix build-system python`). Elle implémente la procédure de construction plus ou moins standard utilisée pour les paquets Python, qui consiste à lancer `python setup.py build` puis `python setup.py install --prefix=/gnu/store/...`

Pour les paquets qui installent des programmes autonomes Python dans `bin/`, elle prend soin d'envelopper ces binaires pour que leur variable d'environnement `GUIX_PYTHONPATH` pointe vers toutes les bibliothèques Python dont ils dépendent.

Le paquet Python utilisé pour effectuer la construction peut être spécifié avec le paramètre `#:python`. C'est une manière utile de forcer un paquet à être construit avec une version particulière de l'interpréteur python, ce qui peut être nécessaire si le paquet n'est compatible qu'avec une version de l'interpréteur.

Par défaut, `guix` appelle `setup.py` sous le contrôle de `setuptools`, tout comme `pip` le fait. Certains paquets ne sont pas compatibles avec `setuptools` (et `pip`), vous pouvez donc le désactiver en réglant le paramètre `#:use-setuptools?` sur `#f`.

Si la sortie `"python"` est disponible, le paquet y est installé au lieu de la sortie `"out"` par défaut. C'est utile pour les paquets qui incluent un paquet Python seulement comme partie du logiciel, et donc qu'il faut combiner les phases de `python-build-system` avec celles d'un autre système de construction. Les liaisons Python sont un cas courant.

**pyproject-build-system** [Variable Scheme]

C'est une variable exportée par `guix build-system pyproject`. Elle est basée sur *python-build-system* et ajoute la prise en charge de `pyproject.toml` et PEP 517 (<https://peps.python.org/pep-0517/>). Elle prend aussi en charge plusieurs moteurs de construction et cadriciels de test.

L'API est un peu différente de celle de *python-build-system* :

- `#:use-setuptools?` et `#:test-target` sont supprimés.
- `#:build-backend` est ajouté. Sa valeur par défaut est `#false` et le système essaiera de deviner le moteur approprié en fonction de `pyproject.toml`.
- `#:test-backend` est ajouté. Sa valeur par défaut est `#false` et le système essaiera de deviner un moteur de test approprié en fonction de ce qui est disponible dans les entrées du paquet.
- `#:test-flags` est ajouté. La valeur par défaut est `'()`. Ces drapeaux sont passés en argument à la commande de test. Remarquez que les drapeaux pour la sortie verbeuse sont toujours activés pour les moteurs pris en charge.

Il est considéré comme « expérimental » dans le sens où les détails d'implémentation ne sont pas encore figés, mais nous vous encourageons à l'essayer pour les nouveaux projets Python (même ceux qui utilisent `setup.py`). L'API pourrait changer mais les changements provoquant des problèmes dans le canal Guix seront gérés.

Ce système de construction finira par être obsolète et fusionné dans *python-build-system*, sans doute au courant de l'année 2024.

**perl-build-system** [Variable Scheme]

Cette variable est exportée par `(guix build-system perl)`. Elle implémente la procédure de construction standard des paquets Perl, qui consiste soit à lancer `perl Build.PL --prefix=/gnu/store/...`, suivi de `Build` et `Build install` ; ou à lancer `perl Makefile.PL PREFIX=/gnu/store/...`, suivi de `make` et `make install`, en fonction de la présence de `Build.PL` ou `Makefile.PL` dans la distribution du paquet. Le premier a la préférence si `Build.PL` et `Makefile.PL` existent tous deux dans la distribution du paquet. Cette préférence peut être inversée en spécifiant `#t` pour le paramètre `#:make-maker?`.

L'invocation initiale de `perl Makefile.PL` ou `perl Build.PL` passe les drapeaux spécifiés par le paramètre `#:make-maker-flags` ou `#:module-build-flags`, respectivement.

Le paquet Perl utilisé peut être spécifié avec `#:perl`.

**renpy-build-system** [Variable Scheme]

Cette variable est exportée par `(guix build-system renpy)`. Elle implémente la procédure de construction plus ou moins standard utilisée pour les jeux Ren'py, qui consiste à charger `#:game` une fois, et donc à créer le bytecode pour le jeu.

Elle crée aussi un script enveloppe dans `bin/` et une entrée de bureau dans `share/applications`, tous deux pouvant être utilisés pour lancer le jeu.

Le paquet Ren'py utilisé peut être spécifié avec `#:renpy`. Les jeux peuvent aussi être installés dans des sorties différentes de « out » avec `#:output`.

**qt-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system qt`). Elle est destinée à être employée avec des applications utilisant Qt ou KDE.

Ce système de construction ajoute les deux phases suivantes à celles définies par `gnu-build-system` :

**check-setup**

La phase `check-setup` prépare l'environnement pour l'exécution des contrôles tels qu'ils sont couramment utilisés par les programmes de test Qt. Pour l'instant, elle ne définit que quelques variables d'environnement : `QT_QPA_PLATFORM=offscreen`, `DBUS_FATAL_WARNINGS=0` et `CTEST_OUTPUT_ON_FAILURE=1`.

Cette phase est ajoutée avant la phase `check`. C'est une phase distincte pour faciliter l'ajustement si nécessaire.

**qt-wrap** La phase `qt-wrap` recherche les chemins des plugins Qt5, les chemins QML et certains XDG dans les entrées et sorties. Si un chemin est trouvé, tous les programmes des répertoires `bin/`, `sbin/`, `libexec/` et `lib/libexec/` de la sortie sont enveloppés dans des scripts définissant les variables d'environnement nécessaires.

Il est possible d'exclure des sorties de paquets spécifiques de ce processus d'emballage en listant leurs noms dans le paramètre `#:qt-wrap-excluded-outputs`. Ceci est utile lorsqu'une sortie est connue pour ne pas contenir de binaires Qt, et où le wrapping ajouterait gratuitement une dépendance de cette sortie à Qt, KDE, ou autre.

Cette phase est exécutée après la phase `install`.

**r-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system r`). Elle implémente la procédure de compilation utilisée par les paquets R (<https://r-project.org>), qui consiste en substance à exécuter `'R CMD INSTALL --library=/gnu/store/...'` dans un environnement où `R_LIBS_SITE` contient les chemins de toutes les entrées des paquets R. Les tests sont exécutés après l'installation en utilisant la fonction `R tools::testInstalledPackage`.

**rakudo-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system rakudo`). Elle implémente la procédure de construction utilisée par Rakudo (<https://rakudo.org/>) pour les paquets Perl6 (<https://perl6.org/>). Elle installe le paquet dans `/gnu/store/.../NOM-VERSION/share/perl6` et installe les binaires, les fichiers de bibliothèques et les ressources, et enveloppe les fichiers dans le répertoire `bin/`. Les tests peuvent être passés en indiquant `#f` au paramètre `tests?`.

Le paquet `rakudo` utilisé peut être spécifié avec `rakudo`. Le paquet `perl6-tap-harness` utilisé pour les tests peut être spécifié avec `#:prove6` ou supprimé en passant `#f` au paramètre `with-prove6?`. Le paquet `perl6-zef` utilisé pour les tests et l'installation peut être spécifié avec `#:ef` ou supprimé en passant `#f` au paramètre `with-zef?`.

**rebar-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system rebar`). Elle implémente une procédure de construction pour `rebar3` (<https://rebar3.org>), un système de construction pour les programmes écrits en Erlang.

Elle ajoute à la fois `rebar3` et `erlang` à l'ensemble des entrées. Des paquets différents peuvent être spécifiés avec les paramètres `#:rebar` et `#:erlang`, respectivement.

Ce système de construction est basé sur `gnu-build-system`, mais avec les phases suivantes modifiées :

**unpack** Cette phase, après avoir déballé les sources comme le fait le `gnu-build-system`, vérifie qu'un fichier `contents.tar.gz` existe au plus haut niveau des sources. Si ce fichier existe, il sera aussi déballé. Cela facilite la gestion des paquets hébergés sur <https://hex.pm/>, le dépôt des paquets Erlang et Elixir.

**bootstrap  
configure**

Il n'y a ni phase `bootstrap` ni `configure` parce que les paquets `erlang` n'ont pas besoin d'être configurés.

**build** Cette phase lance `rebar3 compile` avec les drapeaux listés dans `#:rebar-flags`.

**check** À moins que `#:tests? #f` ne soit passé, cette phase lance `rebar3 eunit`, ou une cible spécifiée avec `#:test-target`, avec les drapeaux listés dans `#:rebar-flags`,

**install** Cela installe les fichiers créés dans le profil `default`, ou un autre profil spécifié avec `#:install-profile`.

**texlive-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system texlive`). Elle est utilisée pour construire des paquets TeX en mode batch avec le moteur spécifié. Le système de construction initialise la variable `TEXINPUTS` pour trouver tous les fichiers source TeX dans ses entrées.

Par défaut, elle lance `luatex` sur tous les fichiers qui se terminent par `ins`. Un moteur et un format différent peuvent être spécifiés avec l'argument `#:tex-format`. Plusieurs cibles de constructions peuvent être indiquées avec l'argument `#:build-targets` qui attend une liste de noms de fichiers. Le système de construction ajoute uniquement `texlive-bin` et `texlive-latex-base` (de (`gnu packages tex`)) à la liste des entrées. Les deux peuvent être remplacés avec les arguments `#:texlive-bin` et `#:texlive-latex-base`, respectivement.

Le paramètre `#:tex-directory` dit au système de construction où installer les fichiers construits dans l'arbre `texmf`.

**ruby-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system ruby`). Elle implémente la procédure de construction RubyGems utilisée par les paquets Ruby qui consiste à lancer `gem build` suivi de `gem install`.

Le champ `source` d'un paquet qui utilise ce système de construction référence le plus souvent une archive `gem`, puisque c'est le format utilisé par les personnes qui développent en Ruby quand elles publient leur logiciel. Le système de construction décompresse l'archive `gem`, éventuellement en corrigeant les sources, lance la suite de tests, recomprime la gemme et l'installe. En plus, des répertoires et des archives peuvent être référencés pour permettre de construire des gemmes qui n'ont pas été publiées depuis Git ou une archive de sources traditionnelle.

Le paquet Ruby utilisé peut être spécifié avec le paramètre `#:ruby`. Une liste de drapeaux supplémentaires à passer à la commande `gem` peut être spécifiée avec le paramètre `#:gem-flags`.

**waf-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system waf`). Elle implémente une procédure de construction autour du script `waf`. Les phases usuelles — `configure`, `build` et `install` — sont implémentées en passant leur nom en argument au script `waf`.

Le script `waf` est exécuté par l'interpréteur Python. Le paquet Python utilisé pour lancer le script peut être spécifié avec le paramètre `#:python`.

**scons-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system scon`s). Elle implémente la procédure de construction utilisée par l'outil de construction SCons. Ce système de construction lance `scons` pour construire le paquet, `scons test` pour lancer les tests puis `scons install` pour installer le paquet.

Des drapeaux supplémentaires à passer à `scons` peuvent être spécifiés avec le paramètre `#:scons-flags`. Les cibles de construction et d'installation par défaut peuvent être remplacées respectivement par les paramètres `#:build-targets` et `#:install-targets`. La version de Python utilisée pour exécuter SCons peut être spécifiée en sélectionnant le paquet SCons approprié avec le paramètre `#:scons`.

**haskell-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system haskell`). Elle implémente la procédure de construction Cabal utilisée par les paquets Haskell, qui consiste à lancer `runhaskell Setup.hs configure --prefix=/gnu/store/...` et `runhaskell Setup.hs build`. Plutôt que d'installer les paquets en lançant `runhaskell Setup.hs install`, pour éviter d'essayer d'enregistrer les bibliothèques dans le répertoire du dépôt en lecture-seule du compilateur, le système de construction utilise `runhaskell Setup.hs copy`, suivi de `runhaskell Setup.hs register`. En plus, le système de construction génère la documentation du paquet en lançant `runhaskell Setup.hs haddock`, à moins que `#:haddock? #f` ne soit passé. Des paramètres facultatifs pour Haddock peuvent être passés à l'aide du paramètre `#:haddock-flags`. Si le fichier `Setup.hs` n'est pas trouvé, le système de construction cherchera `Setup.lhs` à la place.

Le compilateur Haskell utilisé peut être spécifié avec le paramètre `#:haskell` qui a pour valeur par défaut `ghc`.

**dub-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system dub`). Elle implémente la procédure de construction Dub utilisée par les paquets D qui consiste à lancer `dub build` et `dub run`. L'installation est effectuée en copiant les fichiers manuellement.

Le compilateur D utilisé peut être spécifié avec le paramètre `#:ldc` qui vaut par défaut `ldc`.

**emacs-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system emacs`). Elle implémente une procédure d'installation similaire au système de gestion de paquet d'Emacs lui-même (voir Section "Packages" dans *The GNU Emacs Manual*).

Cela crée d'abord le fichier `package-autoloads.el`, puis compile tous les fichiers Emacs Lisp en bytecode. Contrairement au système de gestion de paquets d'Emacs, les fichiers de documentation info sont déplacés dans le répertoire standard et le fichier `dir` est supprimé. Les fichiers du paquet Elisp sont directement installés sous `share/emacs/site-lisp`.

**font-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system font`). Elle implémente une procédure d'installation pour les paquets de polices où des fichiers de polices TrueType, OpenType, etc. sont fournis en amont et n'ont qu'à être copiés à leur emplacement final. Elle copie les fichiers de polices à l'emplacement standard dans le répertoire de sortie.

**meson-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system meson`). Elle implémente la procédure de construction des paquets qui utilisent Meson (<https://mesonbuild.com>) comme système de construction.

Elle ajoute à la fois Meson et Ninja (<https://ninja-build.org/>) à l'ensemble des entrées, et ils peuvent être modifiés avec les paramètres `#:meson` et `#:ninja` si requis.

Ce système de construction est une extension de `gnu-build-system`, mais avec les phases suivantes modifiées pour Meson :

**configure**

La phase lance `meson` avec les drapeaux spécifiés dans `#:configure-flags`. Le drapeau `--buildtype` est toujours défini à `debugoptimized` à moins qu'autre chose ne soit spécifié dans `#:build-type`.

**build** La phase lance `ninja` pour construire le paquet en parallèle par défaut, mais cela peut être changé avec `#:parallel-build?`.

**check** La phase lance `'meson test'` avec un ensemble d'options de base qui ne peuvent pas être modifiées. Cette base d'options peut être étendue avec l'argument `#:test-options`, par exemple pour choisir ou passer une suite de test spécifique.

**install** La phase lance `ninja install` et ne peut pas être changée.



En dehors de cela, le système de construction ajoute aussi la phase suivante :

#### `fix-runpath`

Cette phase s'assure que tous les binaire peuvent trouver les bibliothèques dont ils ont besoin. Elle cherche les bibliothèques requises dans les sous-répertoires du paquet en construction et les ajoute au `RUNPATH` là où c'est nécessaire. Elle supprime aussi les références aux bibliothèques laissées là par la phase de construction par `meson` comme les dépendances des tests, qui ne sont pas vraiment requises pour le programme.

#### `glib-or-gtk-wrap`

Cette phase est la phase fournie par `glib-or-gtk-build-system` et n'est pas activée par défaut. Elle peut l'être avec `#:glib-or-gtk?`.

#### `glib-or-gtk-compile-schemas`

Cette phase est la phase fournie par `glib-or-gtk-build-system` et n'est pas activée par défaut. Elle peut l'être avec `#:glib-or-gtk?`.

### `linux-module-build-system` [Variable Scheme]

`linux-module-build-system` permet de construire des modules du noyau Linux.

Ce système de construction est une extension de *gnu-build-system*, mais avec les phases suivantes modifiées :

#### `configure`

Cette phase configure l'environnement pour que le Makefile du noyau Linux puisse être utilisé pour construire le module du noyau externe.

#### `build`

Cette phase utilise le Makefile du noyau Linux pour construire le module du noyau externe.

#### `install`

Cette phase utilise le Makefile du noyau Linux pour installer le module du noyau externe.

Il est possible et utile de spécifier le noyau Linux à utiliser pour construire le module (sous la forme `arguments` d'un paquet utilisant le `linux-module-build-system`, utilisez la clé `#:linux` pour le spécifier).

### `node-build-system` [Variable Scheme]

Cette variable est exportée par (`guix build-system node`). Elle implémente la procédure de compilation utilisée par Node.js (<https://nodejs.org>), qui implémente une approximation de la commande `npm install`, suivie d'une commande `npm test`.

Le paquet Node.js utilisé pour interpréter les commandes `npm` peut être spécifié avec le paramètre `#:node` dont la valeur par défaut est `node`.

Enfin, pour les paquets qui n'ont pas besoin de choses sophistiquées, un système de construction « trivial » est disponible. Il est trivial dans le sens où il ne fournit en gros aucun support : il n'apporte pas de dépendance implicite, et n'a pas de notion de phase de construction.

**trivial-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system trivial`).

Ce système de construction requiert un argument `#:builder`. Cet argument doit être une expression Scheme qui construit la sortie du paquet — comme avec `build-expression->derivation` (voir Section 9.10 [Dérivations], page 159).

**channel-build-system** [Variable Scheme]

Cette variable est exportée par (`guix build-system channel`).

Ce système de construction est surtout conçu pour usage interne. Pour utiliser ce système de construction, un paquet doit avoir une spécification de canal dans son champ `source` (voir Chapitre 7 [Canaux], page 75); autrement, sa source doit être un nom de répertoire, auquel cas un argument `#:commit` supplémentaire doit être fourni pour spécifier le commit à construire (une chaîne hexadécimale).

Le paquet qui en résulte est une instance de Guix d'un canal donné, comme `guix-time-machine` le construirait.

## 9.6 Phases de construction

Presque tous les systèmes de construction de paquets mettent en œuvre une notion de *phase de construction* : une séquence d'actions que le système de construction exécute, lorsque vous construisez le paquet, conduisant aux sous-produits installés dans le dépôt. Une exception notable est le `trivial-build-system` (voir Section 9.5 [Systèmes de construction], page 124) minimaliste.

Comme nous l'avons dit dans la section précédente, ces systèmes de construction fournissent une liste de phases standards. Pour `gnu-build-system`, les phases de construction principales sont les suivantes :

**set-paths**

Définit les variables d'environnement de chemins de recherche pour tous les paquets d'entrée, dont `PATH` (voir Section 9.8 [Chemins de recherche], page 154).

**unpack**

Décompresse l'archive des sources et se déplace dans l'arborescence des sources fraîchement extraites. Si la source est en fait un répertoire, le copie dans l'arborescence de construction et entre dans ce répertoire.

**patch-source-shebangs**

Corrige les shebangs (`#!`) rencontrés dans les fichiers pour qu'ils se réfèrent aux bons noms de fichiers. Par exemple, elle change `#!/bin/sh` en `#!/gnu/store/...-bash-4.3/bin/sh`.

**configure**

Lance le script `configure` avec un certain nombre d'options par défaut, comme `--prefix=/gnu/store/...`, ainsi que les options spécifiées par l'argument `#:configure-flags`.

**build**

Lance `make` avec la liste des drapeaux spécifiés avec `#:make-flags`. Si l'argument `#:parallel-build?` est vrai (par défaut), construit avec `make -j`.

**check**

Lance `make check`, ou une autre cible spécifiée par `#:test-target`, à moins que `#:tests? #f` ne soit passé. Si l'argument `#:parallel-tests?` est vrai (par défaut), lance `make check -j`.

**install** Lance `make install` avec les drapeaux listés dans `#:make-flags`.

**patch-shebangs**  
Corrige les shebangs des fichiers exécutables installés.

**strip** Nettoie les symboles de débogage dans les fichiers ELF (à moins que `#:strip-binaries?` ne soit faux), les copie dans la sortie `debug` lorsqu'elle est disponible (voir Chapitre 17 [Installer les fichiers de débogage], page 652).

**validate-runpath**  
Valide le `RUNPATH` des binaires ELF, à moins que `#:validate-runpath?` ne soit faux (voir Section 9.5 [Systèmes de construction], page 124).  
Cette étape de validation s'assure que toutes les bibliothèques partagées par les binaires ELF, qui sont listées dans des entrées `DT_NEEDED` de leur segment `PT_DYNAMIC` apparaissent dans l'entrée `DT_RUNPATH` de ce même binaire. En d'autres termes, elle s'assure que lancer ou utiliser ces binaires ne renvoie pas l'erreur « fichier introuvable » à l'exécution. Voir Section « Options » dans *The GNU Linker*, pour plus d'informations sur le `RUNPATH`.

Les autres systèmes de construction ont des phases similaires, avec quelques variations. Par exemple, `cmake-build-system` a des phases de même nom, mais sa phase `configure` exécute `cmake` au lieu de `./configure`. D'autres, tels que `python-build-system`, ont une liste de phases standard totalement différente. Tout ce code s'exécute *côté construction* : il est évalué lorsque vous construisez réellement le paquet, dans un processus de construction dédié engendré par le démon de construction (voir Section 2.5 [Invoquer `guix-daemon`], page 17).

Les phases de construction sont représentées sous forme de listes d'associations ou « alists » (voir Section « Association Lists » dans *Manuel de référence de GNU Guile*) où chaque clé est un symbole pour le nom de la phase et la valeur associée est une procédure qui accepte un nombre arbitraire d'arguments. Par convention, ces procédures reçoivent des informations sur la construction sous la forme de *paramètres nommés*, qu'elles peuvent utiliser ou ignorer.

Par exemple, voici comment `(guix build gnu-build-system)` définit `%standard-phases`, la variable contenant sa liste de phases de construction<sup>3</sup> :

```
;; Les phases de construction de « gnu-build-system ».

(define* (unpack #:key source #:allow-other-keys)
  ;; Extrait l'archive des sources.
  (invoke "tar" "xvf" source))

(define* (configure #:key outputs #:allow-other-keys)
  ;; Exécute le script « configure ». Installe la sortie « out ».
  (let ((out (assoc-ref outputs "out")))
    (invoke "./configure"
             (string-append "--prefix=" out))))
```

---

<sup>3</sup> Nous présentons une vue simplifiée de ces phases de construction, mais jetez un oeil à `(guix build gnu-build-system)` pour voir tous les détails !

```

(define* (build #:allow-other-keys)
  ;; Compile.
  (invoke "make"))

(define* (check #:key (test-target "check") (tests? #true)
             #:allow-other-keys)
  ;; Lance la suite de tests.
  (if tests?
      (invoke "make" test-target)
      (display "test suite not run\n")))

(define* (install #:allow-other-keys)
  ;; Installe les fichiers sous le préfixe spécifié à « configure ».
  (invoke "make" "install"))

(define %standard-phases
  ;; La liste des phases standard (un certain nombre d'entre elles sont omises
  ;; par souci de concision). Chaque élément est une paire symbole/procédure.
  (list (cons 'unpack unpack)
        (cons 'configure configure)
        (cons 'build build)
        (cons 'check check)
        (cons 'install install)))

```

Cela montre comment `%standard-phases` est défini comme une liste de paires symbole/procédure (voir Section “Pairs” dans *GNU Guile Reference Manual*). La première paire associe le symbole `unpack` — un nom — à la procédure `unpack` ; la deuxième paire définit la phase `configure` de manière similaire, et ainsi de suite. Lors de la construction d’un paquet qui utilise `gnu-build-system` avec sa liste de phases par défaut, ces phases sont exécutées de manière séquentielle. Vous pouvez voir le nom de chaque phase commencée et terminée dans le journal de construction des paquets que vous construisez.

Examinons maintenant les procédures elles-mêmes. Chacune est définie par `define*` : `#:key` énumère les paramètres nommés que la procédure accepte, éventuellement avec une valeur par défaut, et `#:allow-other-keys` précise que les autres paramètres nommés sont ignorés (voir Section “Optional Arguments” dans *GNU Guile Reference Manual*).

La procédure `unpack` prend en compte le paramètre `source`, que le système de compilation utilise pour passer le nom de fichier de l’archive source (ou le répertoire cloné d’un système de contrôle de version), et elle ignore les autres paramètres. La phase `configure` ne s’intéresse qu’au paramètre `outputs`, une liste d’association des noms des sorties du paquet avec le nom de leur fichier du dépôt (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59). Elle extrait le nom de fichier de `out`, la sortie par défaut, et le transmet à `./configure` comme préfixe d’installation, ce qui signifie que `make install` copiera tous les fichiers vers ce répertoire (voir Section “Configuration” dans *GNU Coding Standards*). `build` et `install` ignorent tous leurs arguments. `check` prend en compte l’argument `test-target`, qui spécifie le nom de la cible du Makefile pour exécuter les tests ; il imprime un message et saute les tests lorsque `tests?` est faux.

La liste des phases utilisées pour un paquet particulier peut être modifiée avec le paramètre `#:phases` du système de construction. La modification de l'ensemble des phases de construction se résume à la création d'une nouvelle liste de phases basée sur la liste `%standard-phases` décrite ci-dessus. On peut faire cela à l'aide des procédures standards qui manipulent des listes d'associations telles que `alist-delete` (voir Section "SRFI-1 Association Lists" dans *GNU Guile Reference Manual*) ; cependant, il est plus pratique de le faire avec `modify-phases` (voir Section 9.7 [Utilitaires de construction], page 146).

Voici un exemple de définition de paquet qui supprime la phase `configure` de `%standard-phases` et insère une nouvelle phase avant la phase `build`, appelée `set-prefix-in-makefile` :

```
(define-public exemple
  (package
    (name "exemple")
    ;; les autres champs sont omis
    (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)
                                ;; Modifie le Makefile pour que sa variable
                                ;; 'PREFIX' pointe vers « out ».
                                (let ((out (assoc-ref outputs "out")))
                                  (substitute* "Makefile"
                                    (("PREFIX =.*")
                                     (string-append "PREFIX = "
                                                    out "\n"))))))))))))
```

La nouvelle phase insérée est écrite comme une procédure anonyme, introduite avec `lambda*` ; elle prend en compte le paramètre `outputs` que nous avons vu auparavant. Voir Section 9.7 [Utilitaires de construction], page 146, pour en savoir plus sur les procédures auxiliaires utilisées par cette phase, et pour d'autres exemples de `modify-phases`.

Gardez à l'esprit que le code des phases de construction est évalué à la construction effective des paquets. Cela explique pourquoi toute l'expression `modify-phases` ci-dessus est citée (elle vient après ' ou apostrophe) : elle est *mise de côté* pour une exécution ultérieure. Voir Section 9.12 [G-Expressions], page 166, pour une explication de la mise en place du code et des *strates de code* concernées.

## 9.7 Utilitaires de construction

Dès que vous commencerez à écrire des définitions de paquets non triviales (voir Section 9.2 [Définition des paquets], page 105) ou d'autres actions de compilation (voir Section 9.12 [G-Expressions], page 166), vous commencerez probablement à chercher des fonctions auxiliaires pour les actions « de type shell » — créer des répertoires, copier et supprimer des fichiers de manière récursive, manipuler les phases de compilation, etc. Le module (`guix build utils`) fournit de telles procédures utilitaires.

La plupart des systèmes de construction chargent (`guix build utils`) (voir Section 9.5 [Systèmes de construction], page 124). Ainsi, lorsque vous écrivez des phases de construction personnalisées pour vos définitions de paquets, vous pouvez généralement supposer que ces procédures sont disponibles.

Lorsque vous écrivez des G-expressions, vous pouvez importer (`guix build utils`) « côté construction » en utilisant `with-imported-modules` et ensuite le rendre disponible avec la forme `use-modules` (voir Section “Using Guile Modules” dans *GNU Guile Reference Manual*) :

```
(with-imported-modules '((guix build utils)) ;on l'importe
  (computed-file "empty-tree"
    #~(begin
      ;; On le rend disponible.
      (use-modules (guix build utils))

      ;; On utilise sa procédure « mkdir-p » sans souci.
      (mkdir-p (string-append #$output "/a/b/c")))))
```

Le reste de cette section est la référence pour la plupart des procédures auxiliaires fournies par (`guix build utils`).

### 9.7.1 Traitement des noms de fichiers du dépôt

Cette section documente les procédures de traitement des noms de fichier du dépôt.

`%store-directory` [Procédure Scheme]  
Renvoie le nom de répertoire du dépôt.

`store-file-name? fichier` [Procédure Scheme]  
Renvoie vrai si *fichier* est dans le dépôt.

`strip-store-file-name fichier` [Procédure Scheme]  
Enlevez le `/gnu/store` et le hash de *fichier*, un nom de fichier du dépôt. Le résultat est généralement une chaîne `"paquet-version"`.

`package-name->name+version nom` [Procédure Scheme]  
Si l'on passe *nom*, un nom de paquet comme `"toto-0.9.1b"`, on obtient deux valeurs : `"toto"` et `"0.9.1b"`. Lorsque la partie version est indisponible, *nom* et `#f` sont renvoyés. Le premier trait d'union suivi d'un chiffre est considéré comme introduisant la partie version.

### 9.7.2 Types de fichier

Les procédures ci-dessous portent sur les fichiers et les types de fichiers.

`directory-exists? dir` [Procédure Scheme]  
Renvoie `#t` si *dir* existe et est un répertoire.

`executable-file? fichier` [Procédure Scheme]  
Renvoie `#t` si *fichier* existe et est exécutable.

`symbolic-link? fichier` [Procédure Scheme]  
Renvoie `#t` si *fichier* est un lien symbolique (un « symlink »).

`elf-file? fichier` [Procédure Scheme]  
`ar-file? fichier` [Procédure Scheme]  
`gzip-file? fichier` [Procédure Scheme]  
 Renvoie `#t` si *fichier* est, respectivement, un fichier ELF, une archive `ar` (telle qu'une bibliothèque statique `.a`), ou un fichier `gzip`.

`reset-gzip-timestamp fichier [#:keep-mtime? #t]` [Procédure Scheme]  
 Si *fichier* est un fichier `gzip`, réinitialise son horodatage intégré (comme avec `gzip --no-name`) et renvoie vrai. Sinon, renvoie `#f`. Lorsque *keep-mtime?* est vrai, conserve la date de modification de *fichier*.

### 9.7.3 Manipulation de fichiers

Les procédures et macros suivantes permettent de créer, modifier et supprimer des fichiers. Elles offrent des fonctionnalités comparables à celles des utilitaires shell courants tels que `mkdir -p`, `cp -r`, `rm -r` et `sed`. Elles complètent l'interface de système de fichiers étendue, mais de bas niveau, de Guile (voir Section "POSIX" dans *GNU Guile Reference Manual*).

`with-directory-excursion répertoire corps...` [Syntaxe Scheme]  
 Exécute *corps* avec *répertoire* comme répertoire actuel du processus.  
 En gros, cette macro change le répertoire actuel en *répertoire* avant d'évaluer *corps*, en utilisant `chdir`. (voir Section "Processus" dans *Manuel de référence de GNU Guile*). Elle revient au répertoire initial à la sortie de la portée dynamique de *corps*, qu'il s'agisse d'un retour de procédure normal ou d'une sortie non locale telle qu'une exception.

`mkdir-p dir` [Procédure Scheme]  
 Crée le répertoire *dir* et tous ses ancêtres.

`install-file fichier répertoire` [Procédure Scheme]  
 Crée *répertoire* s'il n'existe pas et y copie *fichier* sous le même nom.

`make-file-writable fichier` [Procédure Scheme]  
 Rend *fichier* inscriptible pour son propriétaire.

`copy-recursively source destination [#:log (current-output-port)] [#:follow-symlinks ? #f] [#:copy-file copy-file] [#:keep-mtime? #f] [#:keep-permissions? #t]` [Procédure Scheme]  
 Copie le répertoire *source* dans *destination*. Suit les liens symboliques si *follow-symlinks?* est vrai ; sinon, les conserve. Appelle *copy-file* pour copier les fichiers normaux. Lorsque *keep-mtime?* est vrai, conserve les heures de modification des fichiers de *source* pour ceux de *destination*. Si *keep-permissions?* est vrai, préserve les permissions des fichiers. Affiche une sortie verbuse sur le port *log*.

`delete-file-recursively dir [#:follow-mounts? #f]` [Procédure Scheme]  
 Efface récursivement *dir*, comme `rm -rf`, sans suivre les liens symboliques. Ne suit pas non plus les points de montage, à moins que *follow-mounts?* ne soit vrai. Rapporte mais ignore les erreurs.

**substitute\* fichier ((*regex* *match-var*...) *body*...) ...** [Syntaxe Scheme]

Remplace *regex* dans *fichier* par la chaîne renvoyée par *body*. *body* est évalué avec chaque *match-var* lié à la sous-expression de *regex* positionnelle correspondante. Par exemple :

```
(substitute* file
  ("hello")
  "good morning\n")
(("toto([a-z]+)titi(.*)$" all letters end)
 (string-append "tata" letters end))
```

Ici, chaque fois qu'une ligne de *fichier* contient **hello**, elle est remplacée par **good morning**. Chaque fois qu'une ligne de *fichier* correspond à la deuxième *regex*, **all** est lié à la correspondance complète, **letters** est lié à la première sous-expression, et **end** est lié à la dernière.

Lorsque l'une des *match-var* est `_`, aucune variable n'est liée à la sous-chaîne de correspondance associée.

Autrement, *fichier* peut être une liste de noms de fichier, auquel cas ils sont tous sujets aux substitutions.

Faites attention à l'utilisation du `$` pour faire correspondre la fin d'une ligne ; à lui seul, il ne correspondra pas à la fin d'une nouvelle ligne.

### 9.7.4 Recherche de fichiers

Cette section documente les procédures pour chercher et filtrer des fichiers.

**file-name-predicate *regex*** [Procédure Scheme]

Renvoie un prédicat qui devient vrai lorsqu'on lui passe un nom de fichier dont le nom de base correspond à *regex*.

**find-files *dir* [*pred*] [*#:stat lstat*] [*#:directories ? #f*] [*#:fail-on-error ? #f*]** [Procédure Scheme]

Renvoie la liste des fichiers triés lexicographiquement sous *dir* pour lesquels *pred* renvoie « vrai ». Deux arguments sont passés à *pred* : le nom absolu du fichier et son tampon *stat* ; le prédicat par défaut renvoie toujours « vrai ». *pred* peut également être une expression régulière, auquel cas elle est équivalente à (*file-name-predicate pred*). *stat* est utilisé pour obtenir des informations sur les fichiers ; l'utilisation de *lstat* signifie que les liens symboliques ne sont pas suivis. Si *directories?* est vrai, alors les répertoires seront également inclus. Si *fail-on-error?* est vrai, il lance une exception en cas d'erreur.

Voici quelques exemples où nous supposons que le répertoire actuel est la racine de l'arborescence des sources de Guix :

```
;; Liste tous les fichiers normaux dans le répertoire actuel.
(find-files ".")
⇒ ("./.dir-locals.el" "./.gitignore" ...)
```

```
;; Liste tous les fichiers .scm sous gnu/services.
(find-files "gnu/services" "\\\.scm$")
```



```
⇒ ("gnu/services/admin.scm" "gnu/services/audio.scm" ...)
```

```
;; Liste les fichiers ar dans le répertoire actuel.
(find-files "." (lambda (file stat) (ar-file? file)))
⇒ ("./libformat.a" "./libstore.a" ...)
```

**which *programme*** [Procédure Scheme]  
 Renvoie le nom complet du fichier pour *programme* tel qu'il se trouve dans \$PATH, ou #f si *programme* n'a pas pu être trouvé.

**search-input-file *entrées nom*** [Procédure Scheme]  
**search-input-directory *entrées nom*** [Procédure Scheme]

Renvoie le nom de fichier complet de *nom* trouvé dans *entrées* ; **search-input-file** recherche les fichiers normaux et **search-input-directory** recherche les dossiers. Si *nom* n'est pas trouvé, une exception est levée.

Ici, *entrées* doit être une liste d'association comme les variables **inputs** et **native-inputs** disponibles dans les phases de construction (voir Section 9.6 [Phases de construction], page 143).

Voici un exemple (simplifié) d'utilisation de **search-input-file** dans une phase de construction du paquet **wireguard-tools** :

```
(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 Invocation de programme

Vous trouverez des procédures pratiques pour invoquer des processus dans ce module, en particulier des enveloppes pratiques autour de **system\*** de Guile (voir Section “Processes” dans *le manuel de référence de Guile*).

**invoke *programme args...*** [Procédure Scheme]  
 Invoque le *programme* avec les *args* donnés. Lève une exception **&invoke-error** si le code de retour n'est pas zéro ; sinon renvoie #t.

L'avantage par rapport à **system\*** c'est que vous n'avez pas besoin de vérifier la valeur de sortie. Cela réduit le code dans les petits scripts comme par exemple dans les phases de construction des paquets.

**invoke-error? *c*** [Procédure Scheme]  
 Renvoie vrai si *c* est une condition **&invoke-error**.

**invoke-error-program *c*** [Procédure Scheme]  
**invoke-error-arguments *c*** [Procédure Scheme]  
**invoke-error-exit-status *c*** [Procédure Scheme]  
**invoke-error-term-signal *c*** [Procédure Scheme]

`invoke-error-stop-signal c` [Procédure Scheme]  
 Accède aux champs de *c*, une condition `&invoke-error`.

`report-invoke-error c [port]` [Procédure Scheme]  
 Rapporte *c*, une condition `&invoke-error`, sur *port* (par défaut le port d'erreur actuel), dans un format humainement lisible.

L'utilisation habituelle ressemblerait à ceci :

```
(use-modules (srfi srfi-34) ;pour « guard »
             (guix build utils))

(guard (c ((invoke-error? c)
          (report-invoke-error c)))
      (invoke "date" "--option-imaginaire"))

├─ command "date" "--option-imaginaire" failed with status 1
```

`invoke/quiet programme args...` [Procédure Scheme]  
 Invoque *programme* avec *args* et capture la sortie standard et l'erreur standard de *programme*. Si *programme* termine sans erreur, n'affiche rien et renvoie la valeur non spécifiée ; sinon, lève une condition d'erreur `&message` qui inclut le code de statut et la sortie de *programme*.

Voici un exemple :

```
(use-modules (srfi srfi-34) ;pour « guard »
             (srfi srfi-35) ;pour « message-condition? »
             (guix build utils))

(guard (c ((message-condition? c)
          (display (condition-message c))))
      (invoke/quiet "date") ; tout va bien
      (invoke/quiet "date" "--imaginary-option"))

├─ 'date --imaginary-option' exited with status 1; output follows:

    date : option non reconnue '--imaginary-option'
    Saisissez « date --help » pour plus d'informations.
```

## 9.7.6 Phases de construction

Le `(guix build utils)` contient également des outils permettant de manipuler les phases de construction telles qu'elles sont utilisées par les systèmes de construction (voir Section 9.5 [Systèmes de construction], page 124). Les phases de construction sont représentées sous forme de listes d'associations ou « *alists* » (voir Section “Association Lists” dans *Manuel de référence de GNU Guile*) où chaque clé est un symbole désignant la phase et où la valeur associée est une procédure (voir Section 9.6 [Phases de construction], page 143).

Le noyau Guile et le module `(srfi srfi-1)` fournissent tous deux des outils pour manipuler les alignements. Le module `(guix build utils)` complète ces outils avec des outils écrits en tenant compte des phases de construction.

`modify-phases phases clause...` [Syntaxe Scheme]

Modifie *phases* séquentiellement selon chaque *clause*, qui peut avoir l'une des formes suivantes :

```
(delete old-phase-name)
(replace old-phase-name new-phase)
(add-before old-phase-name new-phase-name new-phase)
(add-after old-phase-name new-phase-name new-phase)
```

Où chaque *phase-name* ci-dessus est une expression s'évaluant en un symbole, et *new-phase* une expression évaluant à une procédure.

L'exemple ci-dessous est tiré de la définition du paquet `grep`. Il ajoute une phase à exécuter après la phase `install`, appelée `fix-egrep-and-fgrep`. Cette phase est une procédure (`lambda*` pour les procédures anonymes) qui prend un paramètre nommé `#:outputs` et ignore les paramètres nommés supplémentaires (voir Section "Optional Arguments" dans *GNU Guile Reference Manual*, pour en savoir plus sur `lambda*` et les arguments optionnels et nommés). La phase utilise `substitute*` pour modifier les scripts `egrep` et `fgrep` installés afin qu'ils se réfèrent à `grep` par son nom de fichier absolu :

```
(modify-phases %standard-phases
  (add-after 'install 'fix-egrep-and-fgrep
    ;; Corrige « egrep » et « fgrep » pour exécuter « grep » via son
    ;; nom de fichier absolu au lieu de le chercher dans $PATH.
    (lambda* (#:key outputs #:allow-other-keys)
      (let* ((out (assoc-ref outputs "out"))
             (bin (string-append out "/bin")))
        (substitute* (list (string-append bin "/egrep")
                          (string-append bin "/fgrep"))
                    (("^exec grep")
                     (string-append "exec " bin "/grep"))))))))
```

Dans l'exemple ci-dessous, les phases sont modifiées de deux façons : la phase standard `configure` est supprimée, vraisemblablement parce que le paquet n'a pas de script `configure` ou quelque chose de similaire, et la phase par défaut `install` est remplacée par une phase qui copie manuellement les fichiers exécutables à installer :

```
(modify-phases %standard-phases
  (delete 'configure) ;pas de script « configure »
  (replace 'install
    (lambda* (#:key outputs #:allow-other-keys)
      ;; Le Makefile du paquet ne fournit pas de règle « install »
      ;; alors on le fait nous-mêmes.
      (let ((bin (string-append (assoc-ref outputs "out")
                               "/bin")))
        (install-file "footswitch" bin)
        (install-file "scythe" bin))))))
```

### 9.7.7 Enveloppes

Il est courant qu'une commande ait besoin que certaines variables d'environnement soient initialisées pour fonctionner correctement, souvent des chemins de recherche (voir Section 9.8

[Chemins de recherche], page 154). Sans cela, la commande peut échouer à trouver les fichiers et les autres commandes dont elle a besoin, ou elle pourrait trouver la « mauvaise » dépendance — en dépendant de l'environnement sur lequel elle tourne. Voici quelques exemples :

- un script shell qui suppose que toutes les commandes qu'il utilise sont dans `PATH` ;
- un programme Guile qui suppose que tous ses modules sont dans `GUILE_LOAD_PATH` et `GUILE_LOAD_COMPILED_PATH` ;
- une application Qt qui s'attend à trouver certains greffons dans `QT_PLUGIN_PATH`.

Pour l'auteur ou l'autrice d'un paquet, le but est de s'assurer que les commandes fonctionnent toujours pareil plutôt que de dépendre de paramètres externes. Une manière d'y arriver est d'*envelopper* les commandes dans un petit script qui définit ces variables d'environnement, ce qui assure que ces dépendances à l'exécution seront trouvées. L'enveloppe serait utilisée pour initialiser `PATH`, `GUILE_LOAD_PATH` ou `QT_PLUGIN_PATH` dans les exemples ci-dessus.

Pour faciliter cette tâche, le module (`guix build utils`) fournit quelques fonctions auxiliaires pour envelopper des commandes.

`wrap-program programme` `[#:sh sh]` `[#:rest variables]` [Procédure Scheme]  
 Crée une enveloppe pour *programme*. *variables* doit ressembler à ceci :

```
'(variable délimiteur position liste-de-répertoires)
```

où *délimiteur* est facultatif. : sera utilisé si *délimiteur* n'est pas fourni.

Par exemple, cet appel :

```
(wrap-program "toto"
  ("PATH" ":" = ("/gnu/.../titi/bin"))
  ("CERT_PATH" suffix ("/gnu/.../tata/certs"
                      "/qux/certs")))
```

copiera `toto` dans `.toto-real` et créera le fichier `toto` avec le contenu suivant :

```
#!emplacement/de/bin/bash
export PATH="/gnu/.../titi/bin"
export CERT_PATH="$CERT_PATH${CERT_PATH:+}/gnu/.../tata/certs:/qux/certs"
exec -a $0 emplacement/de/.foo-real "$@"
```

Si *programme* a déjà été enveloppé par `wrap-program`, l'enveloppe est agrandie avec les définitions de *variables*. Sinon, `sh` sera utilisé comme interpréteur.

`wrap-script programme` `[#:guile guile]` `[#:rest variables]` [Procédure Scheme]

Enveloppe le script *programme* pour que *variables* soient définies avant. Le format de *variables* est le même que pour la procédure `wrap-program`. Cette procédure est différente de `wrap-program` car elle ne crée pas de script shell séparé. Au lieu de cela, *programme* est modifié directement en ajoutant un script Guile au début, qui est interprété comme un commentaire par le langage de script.

Les commentaires à l'encodage spécifique pris en charge par Python sont recréés sur la seconde ligne.

Remarquez que cette procédure ne peut être utilisée qu'une seule fois par fichier car les scripts Guile ne sont pas pris en charge.

## 9.8 Chemins de recherche

De nombreux programmes et bibliothèques cherchent leurs données d'entrée dans un *chemin de recherche*, une liste de répertoire : les shells comme Bash cherchent des exécutables dans le chemin de recherche des commandes, un compilateur C cherche des fichiers `.h` dans son chemin de recherche d'en-têtes, l'interpréteur Python cherche des fichiers `.py` dans son chemin de recherche, le correcteur orthographique a un chemin de recherche pour les dictionnaires, et cætera.

Les chemins de recherche peuvent habituellement être définis ou remplacés par des variables d'environnement (voir Section "Environment Variables" dans *le manuel de référence de la bibliothèque C de GNU*). Par exemple, les chemins de recherches mentionnés ci-dessus peuvent être modifiés en définissant les variables d'environnement `PATH`, `C_INCLUDE_PATH`, `PYTHONPATH` (ou `GUIX_PYTHONPATH`) et `DICPATH` — vous savez, toutes ces variables comme `PATH` que vous devez paramétrer correctement sous peine de voir les choses « non trouvée ».

Vous aurez peut-être remarqué en utilisant la ligne de commande, Guix « sait » quelles variables d'environnement de chemin de recherche doivent être définies et comment. Quand vous installez des paquets dans votre profil par défaut, le fichier `~/.guix-profile/etc/profile` est créé, et vous pouvez le « sourcer » à partir du shell pour initialiser ces variables. De même, si vous demandez à `guix shell` de créer un environnement contenant Python et NumPy, une bibliothèque Python, et que vous passez l'option `--search-paths`, il vous parlera de `PATH` et `GUIX_PYTHONPATH` (voir Section 8.1 [Invoquer guix shell], page 83) :

```
$ 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"■
```

Si vous omettez `--search-paths`, il définit ces variables d'environnement directement, si bien que Python peut immédiatement trouver NumPy :

```
$ guix shell python python-numpy -- python3
Python 3.9.6 (default, Jan 1 1970, 00:00:01)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.version.version
'1.20.3'
```

Pour que cela fonctionne, la définition du paquet `python` *déclare* le chemin de recherche qui l'intéresse et sa variable d'environnement associée, `GUIX_PYTHONPATH`. Cela ressemble à ceci :

```
(package
  (name "python")
  (version "3.9.9")
  ;; quelques champs omis...
  (native-search-paths
    (list (search-path-specification
           (variable "GUIX_PYTHONPATH")
           (files (list "lib/python/3.9/site-packages"))))))))
```

Ce qui ce champ `native-search-paths` signifie que lorsque le paquet `python` est utilisé, la variable d'environnement `GUIX_PYTHONPATH` doit être définie et inclure tous les sous-répertoires `lib/python/3.9/site-packages` rencontrés dans son environnement. (`native-` signifie que, dans un environnement de compilation croisée, seules les entrées natives seront ajoutées au chemin de recherche ; voir Section 9.2.1 [référence de package], page 108.) Dans l'exemple NumPy ci-dessus, le profil où `python` apparaît contient exactement un de ces sous-répertoires, et `GUIX_PYTHONPATH` est initialisé à cette valeur. Lorsqu'il y a plusieurs `lib/python/3.9/site-packages` — c'est le cas dans l'environnement de construction des paquets — ils sont ajoutés à `GUIX_PYTHONPATH`, séparés par des deux-points (:).

**Remarque:** Remarquez que `GUIX_PYTHONPATH` est spécifié dans la définition du paquet `python`, et *pas* dans celle de `python-numpy`. C'est parce que cette variable d'environnement « appartient » à Python, pas à NumPy : Python lit effectivement la valeur de cette variable et la prend en compte.

Corollaire : si vous créez un profil qui ne contient pas `python`, `GUIX_PYTHONPATH` ne sera *pas* défini, même s'il contient des paquets qui fournissent des fichiers `.py` :

```
$ guix shell python-numpy --search-paths --pure
export PATH="/gnu/store/...-profile/bin"
```

Cela a plus de sens si on considère ce profil individuellement : aucun logiciel de ce profil ne lirait `GUIX_PYTHONPATH`.

Bien sûr, il y a de nombreuses variations sur ce thème : certains paquets prennent en compte plus d'un chemin de recherche, certains utilisent d'autres séparateurs que le deux-points, certains accumulent plusieurs répertoires dans leur chemin de recherche, etc. Un exemple plus complexe est le chemin de recherche de `libxml2` : la valeur de la variable d'environnement `XML_CATALOG_FILES` utilise des espaces pour séparer ses composants, elle doit contenir une liste de fichiers `catalog.xml` (pas des répertoires), qui doivent se trouver dans des sous-répertoires `xml` — rien de moins. La spécification du chemin de recherche ressemble à ceci :

```
(package
  (name "libxml2")
  ;; certains champs sont omis
  (native-search-paths
    (list (search-path-specification
           (variable "XML_CATALOG_FILES")
           (separator " ")
           (files '("xml"))
           (file-pattern "^catalog\\.xml$")
           (file-type 'regular))))))
```

Ne vous inquiétez pas, les spécifications de chemins de recherche ne sont généralement pas aussi complexes.

Le module (`guix search-paths`) définit le type de donnée des spécifications de chemins de recherche et un certain nombre de procédures auxiliaires. Vous trouverez ci-dessous la référence des spécifications de chemins de recherche.

`search-path-specification` [Type de données]

Le type de données représentant les spécifications de chemins de recherche.

- variable** le nom de la variable d'environnement pour ce chemin de recherche (une chaîne).
- files** La liste des sous-répertoires (des chaînes) qui devraient être ajoutés au chemin de recherche.
- separator** (par défaut : ":")  
La chaîne utilisée pour séparer les composants du chemin de recherche.  
Un cas spécial, quand **separator** vaut **#f**, signifie qu'il n'y a « qu'un composant au chemin de recherche » — en d'autres termes, le chemin de recherche ne peut contenir plus d'un élément. C'est utile dans certains cas, comme la variable **SSL\_CERT\_DIR** (prise en compte par OpenSSL, cURL, et quelques autres paquets) ou la variable **ASPELL\_DICT\_DIR** (prise en compte par le correcteur orthographique GNU Aspell), toutes deux devant pointer vers un unique répertoire.
- file-type** (par défaut : 'directory')  
Le type de fichier qui doit être trouvé — 'directory' ou 'regular', même si cela peut aussi être n'importe quel symbole renvoyé par **stat:type** (voir Section "File System" dans *le manuel de référence de Guile*).  
Dans l'exemple libxml2 plus haut, on cherche des fichiers normaux ; dans l'exemple Python, on cherche des répertoires.
- file-pattern** (par défaut : #f)  
Cela doit être soit #f, soit une expression régulière qui spécifie les fichiers à trouver à l'intérieur des sous-répertoires spécifiés par le champ **files**.  
De nouveau, l'exemple libxml2 montre une situation où cela est nécessaire.

Certains chemins de recherche ne sont pas liés à une seul paquet, mais à plusieurs. Pour réduire la duplication, certains sont prédéfinis dans (**guix search-paths**).

**\$SSL\_CERT\_DIR** [Variable Scheme]  
**\$SSL\_CERT\_FILE** [Variable Scheme]

Ces deux chemins de recherche indiquent où trouver les certificats X.509 (voir Section 12.11 [Certificats X.509], page 572).

Ces chemins de recherche prédéfinis peuvent être utilisés comme dans l'exemple suivant :

```
(package
  (name "curl")
  ;; certains champs sont omis ...
  (native-search-paths (list $SSL_CERT_DIR $SSL_CERT_FILE)))
```

Comment transformer des spécifications de chemins de recherche d'un côté en et un tas de répertoire de l'autre en un ensemble de définitions de variables d'environnement ? C'est le travail de **evaluate-search-paths**.

**evaluate-search-paths** *search-paths* *directories* [Procédure Scheme]  
[*getenv*]

Évalue *search-paths*, une liste de spécifications de chemins de recherche, par rapport à *directories*, une liste de noms de répertoires, et renvoie une liste de paires de

spécification / valeur. Utilise *getenv* pour déterminer les paramètres actuel et rapporter seulement les paramètres qui ne sont pas déjà en place.

Le module (`guix profiles`) fournit une procédure auxiliaire de haut-niveau, `load-profile`, qui met en place les variables d'environnement d'un profil.

## 9.9 Le dépôt

Conceptuellement, le *dépôt* est l'endroit où les dérivations qui ont bien été construites sont stockées — par défaut, `/gnu/store`. Les sous-répertoires dans le dépôt s'appellent des *éléments du dépôt* ou parfois des *chemins du dépôt*. Le dépôt a une base de données associée qui contient des informations comme les chemins du dépôt auxquels se réfèrent chaque chemin du dépôt et la liste des éléments du dépôt *valides* — les résultats d'une construction réussie. Cette base de données se trouve dans `localstatedir/guix/db` où `localstatedir` est le répertoire d'états spécifié *via* `--localstatedir` à la configuration, typiquement `/var`.

C'est *toujours* le démon qui accède au dépôt pour le compte de ses clients (voir Section 2.5 [Invoquer `guix-daemon`], page 17). Pour manipuler le dépôt, les clients se connectent au démon par un socket Unix-domain, envoient une requête dessus et lisent le résultat — ce sont des appels de procédures distantes, ou RPC.

**Remarque:** Les utilisateurs ne doivent *jamais* modifier les fichiers dans `/gnu/store` directement. Cela entraînerait des incohérences et casserait l'hypothèse d'immutabilité du modèle fonctionnel de Guix (voir Chapitre 1 [Introduction], page 1).

Voir Section 6.5 [Invoquer `guix gc`], page 60, pour des informations sur la manière de vérifier l'intégrité du dépôt et d'essayer de réparer des modifications accidentelles.

Le module (`guix store`) fournit des procédures pour se connecter au démon et pour effectuer des RPCs. Elles sont décrites plus bas. Par défaut, `open-connection`, et donc toutes les commandes `guix` se connectent au démon local ou à l'URI spécifiée par la variable d'environnement `GUIX_DAEMON_SOCKET`.

`GUIX_DAEMON_SOCKET` [Variable d'environnement]

Lorsqu'elle est initialisée, la valeur de cette variable devrait être un nom de fichier ou une URI qui désigne l'extrémité du démon. Lorsque c'est un nom de fichier, il dénote un socket Unix-domain où se connecter. En plus des noms de fichiers, les schémas d'URI supportés sont :

<code>file</code>	
<code>unix</code>	Pour les sockets Unix-domain. <code>file:///var/guix/daemon-socket/socket</code> est équivalent à <code>/var/guix/daemon-socket/socket</code> .
<code>guix</code>	Ces URI dénotent des connexions par TCP/IP, sans chiffrement ni authentification de l'hôte distant. L'URI doit spécifier le nom d'hôte et éventuellement un numéro de port (par défaut 44146) :

`guix://master.guix.example.org:1234`

Ce paramétrage est adapté aux réseaux locaux, comme dans le cas de grappes de serveurs, où seuls des noms de confiance peuvent se connecter au démon de construction sur `master.guix.example.org`.



L'option `--listen` de `guix-daemon` peut être utilisé pour lui dire d'écouter les connexions TCP (voir Section 2.5 [Invoquer `guix-daemon`], page 17).

`ssh` Ces URI vous permettent de vous connecter à un démon distant via SSH. Cette fonctionnalité nécessite Guile-SSH (voir Section 2.2 [Prérequis], page 8) et un binaire `guile` fonctionnel dans `PATH` sur la machine de destination. Elle prend en charge l'authentification par clé publique et GSSAPI. Une URL typique pourrait ressembler à ceci :

```
ssh://charlie@guix.example.org:22
```

Comme pour `guix copy`, les fichiers de configuration du client OpenSSH sont pris en compte (voir Section 10.13 [Invoquer `guix copy`], page 229).

Des schémas d'URI supplémentaires pourraient être supportés dans le futur.

**Remarque:** La capacité de se connecter à un démon de construction distant est considéré comme expérimental à la version 1.4.0. Contactez-nous pour partager vos problèmes ou des suggestions que vous pourriez avoir (voir Chapitre 22 [Contribuer], page 668).

`open-connection [uri] [#:reserve-space? #t]` [Procédure Scheme]

Se connecte au démon à travers le socket Unix-domain à `uri` (une chaîne de caractères). Lorsque `reserve-space?` est vrai, cela demande de réserver un peu de place supplémentaire sur le système de fichiers pour que le ramasse-miette puisse opérer au cas où le disque serait plein. Renvoie un objet serveur.

`file` prend par défaut la valeur `%default-socket-path`, qui est l'emplacement normal compte tenu des options qui ont été passées à `configure`.

`close-connection server` [Procédure Scheme]

Ferme la connexion au serveur.

`current-build-output-port` [Variable Scheme]

Cette variable est liée à un paramètre SRFI-39, qui se réfère au port où les journaux de construction et d'erreur envoyés par le démon devraient être écrits.

Les procédures qui font des RPC prennent toutes un objet serveur comme premier argument.

`valid-path? server path` [Procédure Scheme]

Renvoie `#t` lorsque `path` désigne un élément valide du dépôt et `#f` sinon (un élément invalide peut exister sur le disque mais être toujours invalide, par exemple parce qu'il est le résultat d'une construction avortée ou échouée).

Une condition `&store-protocol-error` est levée si `path` n'est pas préfixée par le répertoire du dépôt (`/gnu/store`).

`add-text-to-store server name text [references]` [Procédure Scheme]

Ajoute `text` dans le fichier `name` dans le dépôt et renvoie son chemin. `references` est la liste des chemins du dépôt référencés par le chemin du dépôt qui en résulte.

**build-derivations store derivations** [Procédure Scheme]  
 [*mode*] Construit *derivations*, une liste d'objets `<derivation>`, de noms de fichiers `.drv`, ou de paires dérivation/sortie, en utilisant le *mode*—(`build-mode normal`) spécifié par défaut.

Remarque que le module (`guix monads`) fournit une monade ainsi que des version monadiques des procédures précédentes, avec le but de rendre plus facile de travailler avec le code qui accède au dépôt (voir Section 9.11 [La monade du dépôt], page 162).

*Cette section est actuellement incomplète.*

## 9.10 Dérivations

Les actions de construction à bas-niveau et l'environnement dans lequel elles sont effectuées sont représentés par des *dérivations*. Une dérivation contient cet ensemble d'informations :

- Les sorties de la dérivation — les dérivations produisent au moins un fichier ou répertoire dans le dépôt, mais peuvent en produire plus.
- Les entrées des dérivations - c'est-à-dire leurs dépendances de temps de construction - qui peuvent être d'autres dérivations ou des fichiers simples dans le dépôt (patches, scripts de construction, etc.).
- Le type de système ciblé par la dérivation — p.ex. `x86_64-linux`.
- Le nom de fichier d'un script de construction dans le dépôt avec les arguments à lui passer.
- Une liste de variables d'environnement à définir.

Les dérivations permettent aux client du démon de communiquer des actions de construction dans le dépôt. Elles existent sous deux formes : en tant que représentation en mémoire, à la fois côté client et démon, et en tant que fichiers dans le dépôt dont le nom finit par `.drv` — on dit que ce sont des *chemins de dérivations*. Les chemins de dérivations peuvent être passés à la procédure `build-derivations` pour effectuer les actions de construction qu'ils prescrivent (voir Section 9.9 [Le dépôt], page 157).

Des opérations comme le téléchargement de fichiers et la récupération de sources gérés par un logiciel de contrôle de version pour lesquels le hash du contenu est connu à l'avance sont modélisés par des *dérivations à sortie fixe*. Contrairement aux dérivation habituelles, les sorties d'une dérivation à sortie fixe sont indépendantes de ses entrées — p.ex. un code source téléchargé produit le même résultat quelque soit la méthode de téléchargement utilisée.

Les sorties des dérivations — c.-à-d. les résultats de la construction — ont un ensemble de *références*, comme le rapporte le RPC `references` ou la commande `guix gc --references` (voir Section 6.5 [Invoquer guix gc], page 60). Les références sont l'ensemble des dépendances à l'exécution des résultats de la construction. Les références sont un sous-ensemble des entrées de la dérivation ; ce sous-ensemble est automatiquement calculé par le démon de construction en scannant tous les fichiers dans les sorties.

Le module (`guix derivations`) fournit une représentation des dérivations comme des objets Scheme, avec des procédures pour créer et manipuler des dérivations. La primitive de plus bas-niveau pour créer une dérivation est la procédure `derivation` :

```

derivation store name builder args [#:outputs '("out")] [Procédure Scheme]
  [#: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 '()]

```

Construit une dérivation avec les arguments donnés et renvoie l'objet `<derivation>` obtenu.

Lorsque *hash* et *hash-algo* sont donnés, une *dérivation à sortie fixe* est créée — c.-à-d. une dérivation dont le résultat est connu à l'avance, comme dans le cas du téléchargement d'un fichier. Si, en plus, *recursive?* est vrai, alors la sortie fixe peut être un fichier exécutable ou un répertoire et *hash* doit être le hash d'une archive contenant la sortie.

Lorsque *references-graphs* est vrai, il doit s'agir d'une liste de paires de noms de fichiers et de chemins du dépôt. Dans ce cas, le graphe des références de chaque chemin du dépôt est exporté dans l'environnement de construction dans le fichier correspondant, dans un simple format texte.

Lorsque *allowed-references* est vrai, il doit s'agir d'une liste d'éléments du dépôt ou de sorties auxquelles la sortie de la dérivation peut faire référence. De même, *disallowed-references*, si vrai, doit être une liste de choses que la sortie ne doit *pas* référencer.

Lorsque *leaked-env-vars* est vrai, il doit s'agir d'une liste de chaînes de caractères qui désignent les variables d'environnements qui peuvent « fuiter » de l'environnement du démon dans l'environnement de construction. Ce n'est possible que pour les dérivation à sortie fixe — c.-à-d. lorsque *hash* est vrai. L'utilisation principale est de permettre à des variables comme `http_proxy` d'être passées aux dérivation qui téléchargent des fichiers.

Lorsque *local-build?* est vrai, déclare que la dérivation n'est pas un bon candidat pour le téléchargement et devrait plutôt être construit localement (voir Section 2.4.2 [Réglages du téléchargement du démon], page 12). C'est le cas des petites dérivation où le coût du transfert de données est plus important que les bénéfices.

Lorsque *substitutable?* est faux, déclare que les substituts de la sortie de la dérivation ne devraient pas être utilisés (voir Section 6.3 [Substituts], page 54). Cela est utile par exemple pour construire des paquets qui utilisent des détails du jeu d'instruction du CPU hôte.

*properties* doit être une liste d'association décrivant les « propriétés » de la dérivation. Elle est gardée telle-quelle, sans être interprétée, dans la dérivation.

Voici un exemple avec un script shell comme constructeur, en supposant que *store* est une connexion ouverte au démon et *bash* pointe vers un exécutable Bash dans le dépôt :

```

(use-modules (guix utils)
             (guix store)
             (guix derivations))

(let ((builder ; ajoute le script Bash au dépôt
      (add-text-to-store store "my-builder.sh"
                        "echo hello world > $out\n" '())))

```

```
(derivation store "toto"
  bash `("-e" ,builder)
  #:inputs `((,bash) (,builder))
  #:env-vars '(("HOME" . "/homeless"))))
⇒ #<derivation /gnu/store/...-toto.drv => /gnu/store/...-toto>
```

Comme on pourrait s'en douter, cette primitive est difficile à utiliser directement. Une meilleure approche est d'écrire les scripts de construction en Scheme, bien sur ! Le mieux à faire pour cela est d'écrire le code de construction comme une « G-expression » et de la passer à `gexp->derivation`. Pour plus d'informations, voir Section 9.12 [G-Expressions], page 166.

Il fut un temps où `gexp->derivation` n'existait pas et où construire une dérivation donc le code de construction était écrit en Scheme se faisait avec `build-expression->derivation`, documenté plus bas. Cette procédure est maintenant obsolète, remplacée par `gexp->derivation` qui est meilleure.

```
build-expression->derivation store name exp [#:system [Procédure Scheme]
  (%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]
```

Renvoie une dérivation qui exécute l'expression Scheme *exp* comme un constructeur pour la dérivation *name*. *inputs* doit être une liste de tuples (**name drv-path sub-drv**) ; lorsque *sub-drv* est omis, "out" est utilisé. *modules* est une liste de noms de modules Guile du chemin de recherche actuel qui seront copiés dans le dépôt, compilés et rendus disponibles dans le chemin de chargement pendant l'exécution de *exp* — p. ex. ((`guix build utils`) (`guix build gnu-build-system`)).

*exp* est évaluée dans un environnement où `%outputs` est lié à une liste de paires de sortie/chemin, et où `%build-inputs` est lié à une liste de paires de chaînes de caractères et de chemin de sortie construite à partir de *inputs*. Éventuellement, *env-vars* est une liste de paires de chaînes de caractères spécifiant le nom et la valeur de variables d'environnement visibles pour le constructeur. Le constructeur termine en passant le résultat de *exp* à `exit` ; ainsi, lorsque *exp* renvoie `#f`, la construction est considérée en échec.

*exp* est construite avec `guile-for-build` (une dérivation). Lorsque `guile-for-build` est omis où est `#f`, la valeur du fluide `%guile-for-build` est utilisée à la place.

Voir la procédure `derivation` pour la signification de `references-graph`, `allowed-references`, `disallowed-references`, `local-build?` et `substitutable?`.

Voici un exemple de dérivation à sortie unique qui crée un répertoire avec un fichier :

```
(let ((builder '(let ((out (assoc-ref %outputs "out")))
  (mkdir out) ; create /gnu/store/...-goo
  (call-with-output-file (string-append out "/test")
    (lambda (p)
      (display '(hello guix) p))))))
  (build-expression->derivation store "goo" builder))
```

```
⇒ #<derivation /gnu/store/...-goo.drv => ...>
```

## 9.11 La monade du dépôt

Les procédures qui travaillent sur le dépôt décrites dans les sections précédentes prennent toutes une connexion ouverte au démon de construction comme premier argument. Bien que le modèle sous-jacent soit fonctionnel, elles ont soit des effets de bord, soit dépendent de l'état actuel du dépôt.

Le premier point est embêtant : on doit se balader avec la connexion au démon dans toutes ces fonctions, ce qui rend impossible le fait de composer des fonctions qui ne prennent pas ce paramètre avec des fonctions qui le prennent. Le deuxième point est problématique : comme les opérations sur le dépôt ont des effets de bord ou dépendent d'états externes, elles doivent être enchaînées correctement.

C'est là que le module (`guix monads`) arrive à la rescousse. Ce module fournit un cadre pour travailler avec des *monads*, en particulier une monade très utile pour notre usage, la *monade du dépôt*. Les monades sont des constructions qui permettent deux choses : associer un « contexte » avec une valeur (dans notre cas, le contexte est le dépôt) et construire une séquence de calculs (ici les calculs comprennent des accès au dépôt). Les valeurs dans une monade — les valeurs qui contiennent ce contexte supplémentaire — sont appelées des *valeurs monadiques* ; les procédures qui renvoient ce genre de valeur sont appelées des *procédures monadiques*.

Considérez cette procédure « normale » :

```
(define (sh-symlink store)
  ;; Renvoie une dérivation qui crée un lien symbolique vers l'exécutable « bash ».
  (let* ((drv (package-derivation store bash))
        (out (derivation->output-path drv))
        (sh (string-append out "/bin/bash")))
    (build-expression->derivation store "sh"
      `(symlink ,sh %output))))
```

En utilisant (`guix monads`) et (`guix gexp`), on peut la réécrire en une fonction monadique :

```
(define (sh-symlink)
  ;; Pareil, mais renvoie une valeur monadique.
  (mlet %store-monad ((drv (package->derivation bash)))
    (gexp->derivation "sh"
      #~(symlink (string-append #$drv "/bin/bash")
                 #$output))))
```

Il y a plusieurs choses à remarquer avec cette deuxième version : le paramètre `store` est maintenant implicitement « enfilé » dans les appels aux procédures monadiques `package->derivation` et `gexp->derivation`, et la valeur monadique renvoyée par `package->derivation` est *liée* avec `mlet` plutôt qu'avec un simple `let`.

Il se trouve que l'appel à `package->derivation` peut même être omis puisqu'il aura lieu implicitement, comme nous le verrons plus tard (voir Section 9.12 [G-Expressions], page 166) :

```
(define (sh-symlink)
```

```
(gexp->derivation "sh"
  #~(symlink (string-append #$bash "/bin/bash")
            #$output)))
```

L'appel à la procédure monadique `sh-symlink` n'a aucun effet. Comme on pourrait le dire, « on sort d'une monade comme de la monarchie : en l'exécutant »<sup>4</sup>. Donc, pour sortir de la monade et obtenir l'effet escompté, on doit utiliser `run-with-store` :

```
(run-with-store (open-connection) (sh-symlink))
⇒ /gnu/store/...-sh-symlink
```

Remarquez que le module `(guix monad-repl)` étend la console Guile avec de nouvelles « commandes » pour rendre plus facile la manipulation de procédures monadiques : `run-in-store` et `enter-store-monad` (voir Section 9.14 [Utiliser Guix de manière interactive], page 177). La première est utilisée pour « lancer » une seule valeur monadique à travers le dépôt :

```
scheme@(guile-user)> ,run-in-store (package->derivation hello)
$1 = #<derivation /gnu/store/...-hello-2.9.drv => ...>
```

La deuxième entre dans une console récursive, où toutes les valeurs de retour sont automatiquement lancées à travers le dépôt :

```
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 "toto" "Hello!")
$3 = "/gnu/store/...-toto"
store-monad@(guile-user) [1]> ,q
scheme@(guile-user)>
```

Remarquez qu'on ne peut pas renvoyer de valeur non monadique dans la console `store-monad`.

D'autres méta-commandes sont disponibles sur la REPL, comme `,build` pour construire un objet simili-fichier (voir Section 9.14 [Utiliser Guix de manière interactive], page 177).

Les formes syntaxiques principales pour utiliser des monades en général sont disponibles dans le module `(guix monads)` et sont décrites ci-dessous.

`with-monad monad body ...` [Syntaxe Scheme]

Évalue n'importe quelle forme `>>=` ou `return` dans `body` comme une *monad*.

`return val` [Syntaxe Scheme]

Renvoie une valeur monadique qui encapsule *val*.

`>>= mval mproc ...` [Syntaxe Scheme]

Lie une valeur monadique *mval*, en passant son « contenu » aux procédures monadiques *mproc*...<sup>5</sup>. Il peut y avoir une ou plusieurs *mproc*, comme dans cet exemple :

```
(run-with-state
```

<sup>4</sup> NdT : il y a là un jeu de mot en anglais qui se base sur un double sens de « run », qui peut se traduire par « exécuter » dans ce contexte.

<sup>5</sup> Cette opération est souvent appelée « bind », mais ce nom dénote une procédure qui n'a rien à voir en Guile. Ainsi, nous empruntons ce symbole quelque peu cryptique au langage Haskell

```
(with-monad %state-monad
  (>>= (return 1)
        (lambda (x) (return (+ 1 x)))
        (lambda (x) (return (* 2 x)))))
'some-state)
```

⇒ 4

⇒ some-state

**mlet** *monad* ((*var mval*) ...) *body* ... [Syntaxe Scheme]

**mlet\*** *monad* ((*var mval*) ...) *body* ... [Syntaxe Scheme]

Lie les variables *var* aux valeurs monadiques *mval* dans *body*, une séquence d'expressions. Comme avec l'opérateur de liaison, on peut réfléchir comme si on « ouvrait » la valeur non-monadique « contenue » dans *mval* et comme si on faisait en sorte que *var* se réfère à cette valeur pure, non-monadique, dans la portée de *body*. La forme (*var -> val*) lie *var* à la valeur « normale » *val*, comme **let**. L'opération de liaison a lieu en séquence de la gauche vers la droite. La dernière expression de *body* doit être une expression monadique et son résultat deviendra le résultat de **mlet** ou **mlet\*** lorsque lancé dans la *monad*.

**mlet\*** est à **mlet** ce que **let\*** est à **let** (voir Section “Local Bindings” dans *GNU Guile Reference Manual*).

**mbegin** *monad mexp* ... [Système Scheme]

Lie *mexp* et les expressions monadiques suivantes en séquence, et renvoie le résultat de la dernière expression. Chaque expression dans la séquence doit être une expression monadique.

Cette procédure est similaire à **mlet**, sauf que les valeurs de retour des expressions monadiques sont ignorées. Dans ce sens, elle est analogue à **begin**, mais appliqué à des expressions monadiques.

**mwhen** *condition mexp0 mexp\** ... [Système Scheme]

Lorsque la *condition* est vraie, évalue la séquence des expressions monadiques *mexp0..mexp\** comme dans un **mbegin**. Lorsque la *condition* est fausse, renvoie **\*unspecified\*** dans la monade actuelle. Chaque expression dans la séquence doit être une expression monadique.

**munless** *condition mexp0 mexp\** ... [Système Scheme]

Lorsque la *condition* est fausse, évalue la séquence des expressions monadiques *mexp0..mexp\** comme dans un **mbegin**. Lorsque la *condition* est vraie, renvoie **\*unspecified\*** dans la monade actuelle. Chaque expression dans la séquence doit être une expression monadique.

Le module (**guix monads**) fournit la *monade d'état* qui permet à une valeur supplémentaire — l'état — d'être enfilée à travers les appels de procédures.

**%state-monad** [Variable Scheme]

La monade d'état. les procédure dans la monade d'état peuvent accéder et modifier l'état qui est enfilé.

Considérez l'exemple ci-dessous. La procédure `square` renvoie une valeur dans la monade d'état. Elle renvoie le carré de son argument, mais incrémente aussi la valeur actuelle de l'état :

```
(define (square x)
  (mlet %state-monad ((count (current-state)))
    (mbegin %state-monad
      (set-current-state (+ 1 count))
      (return (* x x)))))

(run-with-state (sequence %state-monad (map square (iota 3))) 0)
⇒ (0 1 4)
⇒ 3
```

Lorsque c'est « lancé » à travers `%state-monad`, nous obtenons cette valeur d'état supplémentaire, qui est le nombre d'appels au `square`.

`current-state` [Procédure monadique]

Renvoie l'état actuel dans une valeur monadique.

`set-current-state value` [Procédure monadique]

Initialise l'état actuel à *value* et renvoie l'état précédent dans une valeur monadique.

`state-push value` [Procédure monadique]

Pousse *value* sur l'état actuel, qui est supposé être une liste, et renvoie l'état précédent dans une valeur monadique.

`state-pop` [Procédure monadique]

Récupère (pop) une valeur dans l'état actuel et la renvoie comme une valeur monadique. L'état est supposé être une liste.

`run-with-state mval [state]` [Procédure Scheme]

Lance la valeur monadique *mval* avec *state* comme valeur initiale. Renvoie deux valeurs : la valeur du résultat et l'état du résultat.

L'interface principale avec la monade du dépôt, fournit par le module (`guix store`), est la suivante.

`%store-monad` [Variable Scheme]

La monade du dépôt — un alias pour `%state-monad`.

Les valeurs dans la monade du dépôt encapsulent les accès vers le dépôt. Lorsque son effet est nécessaire, une valeur de la monade du dépôt doit être « évaluée » en la passant à la procédure `run-with-store` (voir ci-dessous).

`run-with-store store mval [#:guile-for-build] [#:system (%current-system)]` [Procédure Scheme]

Lance *mval*, une valeur monadique dans la monade du dépôt, dans *store*, une connexion ouvert au dépôt.

`text-file name text [references]` [Procédure monadique]

Renvoie une valeur monadique correspondant au nom de fichier dans le dépôt du fichier contenant *text*, une chaîne de caractères. *references* est une liste d'éléments du dépôt auxquels le fichier texte en résultat se réfère ; c'est la liste vide par défaut.



**binary-file** *name data* [*references*] [Procédure monadique]

Renvoie une valeur monadique correspondant au nom de fichier absolu dans le dépôt du fichier contenant *data*, un vecteur d'octets. *references* est une liste d'éléments du dépôt auxquels le fichier binaire en résultat se réfère ; c'est la liste vide par défaut.

**interned-file** *file* [*name*] [*#:recursive?* *#t*] [*#:select?* (*const #t*)] [Procédure monadique]

Renvoie le nom de *file* une fois ajouté au dépôt. Utilise *name* comme nom dans le dépôt ou le nom de fichier de *file* si *name* est omis.

Lorsque *recursive?* est vraie, le contenu de *file* est ajouté récursivement ; si *file* désigne un fichier simple et que *recursive?* est vrai, son contenu est ajouté et ses bits de permissions sont préservés.

Lorsque *recursive?* est vraie, appelle (*select? file stat*) pour chaque répertoire où *file* est le nom de fichier absolu de l'entrée et *stat* est le résultat de *lstat* ; à l'exception des entrées pour lesquelles *select?* ne renvoie pas vrai.

L'exemple ci-dessous ajoute un fichier au dépôt, sous deux noms différents :

```
(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")

Le module (*guix packages*) exporte les procédures monadiques liées aux paquets suivantes :

**package-file** *package* [*file*] [*#:system* (*%current-system*)] [*#:target* *#f*] [*#:output* "out"] [Procédure monadique]

Renvoie une valeur monadique qui contient le nom de fichier absolu de *file* dans le répertoire *output* de *package*. Lorsque *file* est omis, renvoie le nom du répertoire *output* de *package*. Lorsque *target* est vrai, l'utilise comme un triplet de cible pour la compilation croisée.

Notez que cette procédure ne permet *pas* de construire *package*. Ainsi, le résultat peut ou non désigner un fichier existant. Nous vous recommandons de ne pas utiliser cette procédure si vous ne savez pas ce que vous faites.

**package->derivation** *package* [*system*] [Procédure monadique]

**package->cross-derivation** *package target* [*system*] [Procédure monadique]

Version monadique de *package-derivation* et *package-cross-derivation* (voir Section 9.2 [Définition des paquets], page 105).

## 9.12 G-Expressions

On a donc des « dérivations » qui représentent une séquence d'actions de construction à effectuer pour produire un élément du dépôt (voir Section 9.10 [Dérivations], page 159). Ces actions de construction sont effectuées lorsqu'on demande au démon de construire effectivement les dérivations ; elles sont lancées par le démon dans un conteneur (voir Section 2.5 [Invoquer *guix-daemon*], page 17).

Ça ne devrait pas vous surprendre, mais nous aimons écrire ces actions de construction en Scheme. Lorsqu'on fait ça, on fini avec deux *strates* de code Scheme<sup>6</sup> : le « code hôte » — le code qui définit les paquets, parle au démon, etc. — et le « code côté construction » — le code qui effectue effectivement les actions de construction, comme créer des répertoires, invoquer `make`, etc. (voir Section 9.6 [Phases de construction], page 143).

Pour décrire une dérivation et ses actions de construction, on a typiquement besoin d'intégrer le code de construction dans le code hôte. Ça revient à manipuler le code de construction comme de la donnée, et l'homoiconicité de Scheme — le code a une représentation directe en tant que donnée — est très utile pour cela. Mais on a besoin de plus que le mécanisme de `quasiquote` en Scheme pour construire des expressions de construction.

Le module (`guix gexp`) implémente les *G-expressions*, une forme de S-expression adaptée aux expressions de construction. Les G-expression, ou *gexps*, consistent en gros en trois formes syntaxiques : `gexp`, `ungexp` et `ungexp-splicing` (ou plus simplement : `#~`, `#$` et `#$@`), qui sont comparable à `quasiquote`, `unquote` et `unquote-splicing` respectivement (voir Section “Expression Syntax” dans *GNU Guile Reference Manual*). Cependant il y a des différences majeures :

- Les Gexps sont conçues pour être écrites dans un fichier et être lancées ou manipulées par d'autres processus.
- Lorsqu'un objet de haut-niveau comme un paquet ou une dérivation est unquotée dans une `gexp`, le résultat est comme si le nom de fichier de son résultat avait été introduit.
- Les `gexps` transportent des informations sur les paquets ou les dérivations auxquels elles se réfèrent, et ces dépendances sont automatiquement ajoutées comme des entrées du processus de construction qui les utilise.

Ce mécanisme n'est pas limité aux paquets et aux objets de dérivation : Des *compilateurs* capables d' « abaisser » d'autres objets de haut niveau vers des dérivations ou des fichiers dans le dépôt peuvent être définis, de sorte que ces objets peuvent également être insérés dans des `gexps`. Par exemple, un type utile d'objets de haut niveau pouvant être insérés dans un `gexp` est celui des « objets de type fichier », qui permettent d'ajouter facilement des fichiers au dépôt et d'y faire référence dans des dérivations et autres (voir `local-file` et `plain-file` ci-dessous).

Pour illustrer cette idée, voici un exemple de `gexp` :

```
(define build-exp
  #~(begin
      (mkdir #$output)
      (chdir #$output)
      (symlink (string-append #$coreutils "/bin/ls")
               "list-files")))
```

Cette `gexp` peut être passée à `gexp->derivation` ; on obtient une dérivation qui construit un répertoire contenant exactement un lien symbolique à `/gnu/store/...-coreutils-8.22/bin/ls` :

<sup>6</sup> Le terme de *strate* dans ce contexte a été inventé par Manuel Serrano et ses collègues dans le contexte de leurs travaux sur Hop. Oleg Kiselyov, qui a écrit des essais perspicaces ainsi que du code sur le sujet (<http://okmij.org/ftp/meta-programming/#meta-scheme>), utilise le terme de « mise en scène » pour ce genre de génération de code.

```
(gexp->derivation "the-thing" build-exp)
```

Comme on pourrait s'y attendre, la chaîne `"/gnu/store/...-coreutils-8.22"` est substituée à la place de la référence au paquet `coreutils` dans le code de construction final, et `coreutils` est automatiquement devenu une entrée de la dérivation. De même, `#$output` (équivalent à `(ungexp output)`) est remplacé par une chaîne de caractères contenant le nom du répertoire de la sortie de la dérivation.

Dans le contexte d'une compilation croisée, il est utile de distinguer entre des références à la construction *native* d'un paquet — qui peut être lancé par l'hôte — et des références à la construction croisée d'un paquet. Pour cela, `#+` joue le même rôle que `#$`, mais référence une construction native d'un paquet :

```
(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")
```

Dans l'exemple ci-dessus, la construction native de `coreutils` est utilisée, pour que `ln` puisse effectivement être lancé sur l'hôte ; mais ensuite la construction croisée d'`emacs` est utilisée.

Une autre fonctionnalité, ce sont les *modules importés* : parfois vous voudriez pouvoir utiliser certains modules Guile de « l'environnement hôte » dans la `gexp`, donc ces modules devraient être importés dans « l'environnement de construction ». La forme `with-imported-modules` vous permet d'exprimer ça :

```
(let ((build (with-imported-modules '((guix build utils))
  #~(begin
    (use-modules (guix build utils)
      (mkdir-p (string-append #$output "/bin"))))))
  (gexp->derivation "empty-dir"
    #~(begin
      #$build
      (display "success!\n")
      #t)))
```

Dans cet exemple, le module `(guix build utils)` est automatiquement récupéré dans l'environnement de construction isolé de notre `gexp`, pour que `(use-modules (guix build utils))` fonctionne comme on s'y attendrait.

Typiquement, vous voudriez que la *closure* complète du module soit importé — c.-à-d. le module lui-même et tous les modules dont il dépend — plutôt que seulement le module ; sinon, une tentative de chargement du module échouera à cause des modules dépendants manquants. La procédure `source-module-closure` calcule la closure d'un module en cherchant dans ses en-têtes sources, ce qui est pratique dans ce cas :

```
(use-modules (guix modules)) ;pour 'source-module-closure'
```

```
(with-imported-modules (source-module-closure
```

```

      '((guix build utils)
       (gnu build image)))
(gexp->derivation "something-with-vms"
 #~(begin
     (use-modules (guix build utils)
                  (gnu build image))
     ...)))

```

Dans la même idée, parfois vous pouvez souhaiter importer non seulement des modules en Scheme pur, mais aussi des « extensions » comme des liaisons Guile de bibliothèques C ou d'autres paquet « complets ». Disons que vous voulez utiliser le paquet `guile-json` du côté de la construction, voici comme procéder :

```

(use-modules (gnu packages guile)) ;pour 'guile-json'

(with-extensions (list guile-json)
 (gexp->derivation "something-with-json"
 #~(begin
     (use-modules (json))
     ...)))

```

La forme syntaxique pour construire des gexps est résumée ci-dessous.

```

#~ exp [Syntaxe Scheme]
(gexp exp) [Syntaxe Scheme]

```

Renvoie une G-expression contenant *exp*. *exp* peut contenir une ou plusieurs de ces formes :

```

#$obj
(ungexp obj)

```

Introduit une référence à *obj*. *obj* peut être d'un des types supportés, par exemple un paquet ou une dérivation, auquel cas la forme `ungexp` est remplacée par le nom de fichier de sa sortie — p. ex. `"/gnu/store/...-coreutils-8.22`.

Si *obj* est une liste, elle est traversée et les références aux objets supportés sont substitués de manière similaire.

Si *obj* est une autre gexp, son contenu est inséré et ses dépendances sont ajoutées à celle de la gexp qui l'entoure.

Si *obj* est un autre type d'objet, il est inséré tel quel.

```

#$obj:output
(ungexp obj output)

```

Cette forme est similaire à la précédente, mais se réfère explicitement à la sortie *output* de l'objet *obj* — c'est utile lorsque *obj* produit plusieurs sorties (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59).

```

#+obj
#+obj:output
(ungexp-native obj)
(ungexp-native obj output)
    Comme ungexp, mais produit une référence à la construction native de
    obj lorsqu'elle est utilisée dans une compilation croisée.

#$output[:output]
(ungexp output [output])
    Insère une référence à la sortie output de la dérivation, ou à la sortie
    principale lorsque output est omis.
    Cela ne fait du sens que pour les gexps passées à gexp->derivation.

#$@lst
(ungexp-splicing lst)
    Comme au dessus, mais recolle (splice) le contenu de lst dans la liste qui
    la contient.

#+@lst
(ungexp-native-splicing lst)
    Comme au dessus, mais se réfère à la construction native des objets listés
    dans lst.

```

Les G-expressions créées par `gexp` ou `#~` sont des objets d'exécution du type `gexp?` (voir ci-dessous).

`with-imported-modules modules body...` [Syntaxe Scheme]

Marque les gexps définies dans *body...* comme requérant *modules* dans leur environnement d'exécution.

Chaque élément dans *module* peut être le nom d'un module, comme (`guix build utils`) ou le nom d'un module suivi d'une flèche, suivie d'un objet simili-fichier :

```

`((guix build utils)
  (guix gcrypt)
  ((guix config) => ,(scheme-file "config.scm"
                                #~(define-module ...))))

```

Dans l'exemple au dessus, les deux premiers modules sont récupérés dans le chemin de recherche, et le dernier est créé à partir d'un objet simili-fichier.

Cette forme a une portée *lexicale* : elle a un effet sur les gexp directement définies dans *body...*, mais pas sur celles définies dans des procédures appelées par *body...*

`with-extensions extensions body...` [Syntaxe Scheme]

Marque les gexps définies dans *body...* comme requérant *extensions* dans leur environnement de construction et d'exécution. *extensions* est typiquement une liste d'objets paquets comme définis dans le module (`gnu packages guile`).

Concrètement, les paquets listés dans *extensions* sont ajoutés au chemin de chargement lors de la compilation des modules importés dans *body...* ; ils sont aussi ajoutés au chemin de chargement de la gexp renvoyée par *body...*

`gexp? obj` [Procédure Scheme]

Renvoie `#t` si `obj` est une G-expression.

Les G-expressions sont conçues pour être écrites sur le disque, soit en tant que code pour construire une dérivation, soit en tant que fichier normal dans le dépôt. Les procédures monadiques suivantes vous permettent de faire cela (voir Section 9.11 [La monade du dépôt], page 162, pour plus d'information sur les monads).

`gexp->derivation name exp` [#:system [Procédure monadique] (%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] [#:substituable ? #t] [#:properties '()] [#:guile-for-build #f]

Renvoie une dérivation `name` qui exécute `exp` (un `gexp`) avec `guile-for-build` (une dérivation) sur `system` ; `exp` est stocké dans un fichier appelé `script-name`. Lorsque `target` est vrai, il est utilisé comme le triplet cible de compilation croisée pour les paquets auxquels `exp` fait référence.

`modules` est devenu obsolète en faveur de `with-imported-modules`. Sa signification est de rendre `modules` disponibles dans le contexte d'évaluation de `exp` ; `modules` est une liste de noms de modules Guile qui sont cherchés dans `module-path` pour les copier dans le dépôt, les compiler et les rendre disponibles dans le chemin de chargement pendant l'exécution de `exp` — p. ex. ((guix build utils) (guix build gnu-build-system)).

`effective-version` détermine la chaîne à utiliser lors d'ajout d'extensions de `exp` (voir `with-extensions`) au chemin de recherche — p. ex. "2.2".

`graft?` détermine si les paquets référencés par `exp` devraient être greffés si possible.

Lorsque `references-graphs` est vrai, il doit s'agir d'une liste de tuples de la forme suivante :

```
(file-name package)
(file-name package output)
(file-name derivation)
(file-name derivation output)
(file-name store-item)
```

La partie droite des éléments de `references-graphs` est automatiquement transformée en une entrée du processus de construction `exp`. Dans l'environnement de construction, chaque `file-name` contient le graphe des références de l'élément correspondant, dans un format texte simple.

`allowed-references` doit soit être `#f`, soit une liste de noms de sorties ou de paquets. Dans ce dernier cas, la liste dénote les éléments du dépôt auxquels le résultat a le droit de faire référence. Toute référence à un autre élément du dépôt conduira à une erreur à la construction. Comme pour `disallowed-references`, qui peut lister des éléments qui ne doivent pas être référencés par les sorties.

*deprecation-warnings* détermine s'il faut afficher les avertissement d'obsolescence à la compilation de modules. Il peut valoir `#f`, `t` ou `'detailed`.

Les autres arguments sont les mêmes que pour *derivation* (voir Section 9.10 [Dérivations], page 159).

Les procédures *local-file*, *plain-file*, *computed-file*, *program-file* et *scheme-file* ci-dessous renvoient des *objets simili-fichiers*. C'est-à-dire, lorsqu'ils sont unquotés dans une G-expression, ces objets donnent un fichier dans le dépôt. Considérez cette G-expression :

```
#~(system* #$(file-append glibc "/sbin/nscd") "-f"
     #$(local-file "/tmp/my-nscd.conf"))
```

Ici, l'effet est « d'internaliser » `/tmp/my-nscd.conf` en le copiant dans le dépôt. Une fois étendu, par exemple via `gexp->derivation`, la G-expression se réfère à cette copie dans `/gnu/store` ; ainsi, modifier ou supprimer le fichier dans `/tmp` n'a aucun effet sur ce que fait la G-expression. *plain-file* peut être utilisé de la même manière ; elle est seulement différente par le fait que le contenu du fichier est passé directement par une chaîne de caractères.

**local-file** *file* [*name*] [*#:recursive ? #f*] [*#:select ? (const #t)*] [Procédure Scheme]

Renvoie un objet représentant le fichier local *file* à ajouter au magasin ; cet objet peut être utilisé dans un `gexp`. Si *file* est une chaîne littérale désignant un nom de fichier relatif, il est recherché par rapport au fichier source où il apparaît ; si *file* n'est pas une chaîne littérale, il est recherché par rapport au répertoire de travail courant au moment de l'exécution. *file* sera ajouté au dépôt sous *name*—par défaut le nom de base de *file*.

Lorsque *recursive?* est vraie, le contenu de *file* est ajouté récursivement ; si *file* désigne un fichier simple et que *recursive?* est vrai, son contenu est ajouté et ses bits de permissions sont préservés.

Lorsque *recursive?* est vraie, appelle (*select? file stat*) pour chaque répertoire où *file* est le nom de fichier absolu de l'entrée et *stat* est le résultat de `lstat` ; à l'exception des entrées pour lesquelles *select?* ne renvoie pas vrai.

C'est la version déclarative de la procédure monadique *interned-file* (voir Section 9.11 [La monade du dépôt], page 162).

**plain-file** *name content* [Procédure Scheme]

Renvoie un objet représentant un fichier texte nommé *name* avec pour contenu *content* (une chaîne de caractères ou un vecteur d'octets) à ajouter un dépôt.

C'est la version déclarative de *text-file*.

**computed-file** *name gexp* [*#:local-build ? #t*] [*#:options*] [Procédure Scheme]

Renvoie un objet représentant l'élément du dépôt *name*, un fichier ou un répertoire calculé par *gexp*. Lorsque *local-build?* est vrai (par défaut), la dérivation est construite localement. *options* est une liste d'arguments supplémentaires à passer à `gexp->derivation`.

C'est la version déclarative de `gexp->derivation`.

**gexp->script** *name exp* [#:guile (*default-guile*)] [Procédure monadique]  
 [#:module-path %load-path] [#:system (%current-system)] [#:target #f]  
 Renvoie un script exécutable *name* qui exécute *exp* en utilisant *guile*, avec les modules importés de *exp* dans son chemin de recherche. Recherchez les modules de *exp* dans *module-path*.

L'exemple ci-dessous construit un script qui invoque simplement la commande `ls` :

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

(gexp->script "list-files"
  #~(execl #$(file-append coreutils "/bin/ls")
    "ls"))
```

Lorsqu'elle est « lancée » à travers le dépôt (voir Section 9.11 [La monade du dépôt], page 162), on obtient une dérivation qui produit un fichier exécutable `/gnu/store/...-list-files` qui ressemble à :

```
#!/gnu/store/...-guile-2.0.11/bin/guile -ds
!#
(execl "/gnu/store/...-coreutils-8.22"/bin/ls" "ls")
```

**program-file** *name exp* [#:guile #f] [#:module-path %load-path] [Procédure Scheme]

Renvoie un objet représentant un élément du dépôt *name* qui lance *gexp*. *guile* est le paquet Guile à utiliser pour exécuter le script. Les modules importés par *gexp* sont recherchés dans *module-path*.

C'est la version déclarative de `gexp->script`.

**gexp->file** *name exp* [#:set-load-path? #t] [Procédure monadique]  
 [#:module-path %load-path] [#:splice? #f] [#:guile (*default-guile*)]

Renvoie une dérivation qui construit un fichier *name* contenant *exp*. Lorsque *splice?* est vrai, *exp* est considéré comme une liste d'expressions qui seront splicée dans le fichier qui en résulte.

Lorsque *set-load-path?* est vrai, émet du code dans le fichier de résultat pour initialiser `%load-path` et `%load-compiled-path` pour prendre en compte les modules importés de *exp*. Les modules de *exp* sont trouvés dans *module-path*.

Le fichier qui en résulte retient les références à toutes les dépendances de *exp* ou un sous-ensemble.

**scheme-file** *name gexp* [#:splice? #f] [#:set-load-path? #t] [Procédure Scheme]

Renvoie un objet représentant le fichier Scheme *name* qui contient *exp*.

C'est la version déclarative de `gexp->file`.

**text-file\*** *name text* ... [Procédure monadique]

Renvoie une valeur monadique qui construit un fichier texte contenant *text*. *text* peut lister, en plus de chaînes de caractères, des objet de n'importe quel type qui peut être utilisé dans une *gexp* : des paquets, des dérivations, des fichiers objet locaux, etc. Le fichier du dépôt qui en résulte en retient toutes les références.



Cette variante devrait être préférée à `text-file` lorsque vous souhaitez créer des fichiers qui référencent le dépôt. Cela est le cas typiquement lorsque vous construisez un fichier de configuration qui contient des noms de fichiers du dépôt, comme ceci :

```
(define (profile.sh)
  ;; Renvoie le nom d'un script shell dans le dépôt qui initialise
  ;; la variable d'environnement « PATH ».
  (text-file* "profile.sh"
    "export PATH=" coreutils "/bin:"
    grep "/bin:" sed "/bin\n"))
```

Dans cet exemple, le fichier `/gnu/store/...-profile.sh` qui en résulte référence `coreutils`, `grep` et `sed`, ce qui les empêche d'être glanés tant que le script est accessible.

`mixed-text-file name text ...` [Procédure Scheme]

Renvoie un objet représentant le fichier du dépôt `name` contenant `text`. `text` est une séquence de chaînes de caractères et de fichiers simili-objets, comme dans :

```
(mixed-text-file "profile"
  "export PATH=" coreutils "/bin:" grep "/bin")
```

C'est la version déclarative de `text-file*`.

`file-union name files` [Procédure Scheme]

Renvoie un `<computed-file>` qui construit un répertoire qui contient tous les fichiers de `files`. Chaque élément de `files` doit être une paire où le premier élément est le nom de fichier à utiliser dans le nouveau répertoire et le second élément est une gexp dénotant le fichier cible. Voici un exemple :

```
(file-union "etc"
  `(("hosts" ,(plain-file "hosts"
    "127.0.0.1 localhost"))
  ("bashrc" ,(plain-file "bashrc"
    "alias ls='ls --color=auto'"))))■
```

Cela crée un répertoire `etc` contenant ces deux fichiers.

`directory-union name things` [Procédure Scheme]

Renvoie un répertoire qui est l'union de `things`, où `things` est une liste d'objets simili-fichiers qui dénotent des répertoires. Par exemple :

```
(directory-union "guile+emacs" (list guile emacs))
```

crée un répertoire qui est l'union des paquets `guile` et `emacs`.

`file-append obj suffix ...` [Procédure Scheme]

Renvoie un objet simili-fichier qui correspond à la concaténation de `obj` et `suffix` où `obj` est un objet abaissable et chaque `suffix` est une chaîne de caractères.

Par exemple, considérez cette gexp :

```
(gexp->script "run-uname"
  #~(system* #$(file-append coreutils
    "/bin/uname")))
```

On peut obtenir le même effet avec :

```
(gexp->script "run-uname"
```

```
#~(system* (string-append #coreutils
                        "/bin/uname")))
```

Il y a une différence cependant : dans le cas `file-append`, le script qui en résulte contient le nom de fichier absolu comme une chaîne de caractère alors que dans le deuxième cas, le script contient une expression (`string-append ...`) pour construire le nom de fichier à l'exécution.

```
let-system system body... [Syntaxe Scheme]
```

```
let-system (system target) body... [Syntaxe Scheme]
```

Lier `system` au système actuellement visé—par exemple, "x86\_64-linux"—dans `body`.

Dans le second cas, lier également `target` à la cible de compilation croisée actuelle—un triplet GNU tel que "arm-linux-gnueabi"—ou `#f` si nous ne faisons pas de compilation croisée.

`let-system` est utile occasionnellement dans le cas où l'objet raccordé au `gexp` dépend de la cible sur le système cible, comme dans cet exemple :

```
#~(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 "dunno!")))))
  "-net" "user" #image)
```

```
with-parameters ((parameter value) ...) exp [Syntaxe Scheme]
```

Cette macro est similaire à la forme `parameterize` pour les *paramètres* liés dynamiquement (voir Section "Parameters" dans *GNU Guile Reference Manual*). La principale différence est qu'il prend effet lorsque l'objet de type fichier renvoyé par `exp` est abaissé à un élément de dérivation ou de stockage.

Une utilisation typique de `with-parameters` consiste à forcer le système en vigueur pour un objet donné :

```
(with-parameters (%current-system "i686-linux"))
  coreutils)
```

L'exemple ci-dessus renvoie un objet qui correspond à la version i686 de Coreutils, quelle que soit la valeur actuelle de `%current-système`.

Bien sûr, en plus de `gexps` incluses dans le code « hôte », certains modules contiennent des outils de construction. Pour savoir facilement qu'ils sont à utiliser dans la strate de construction, ces modules sont gardés dans l'espace de nom (`guix build ...`).

En interne, les objets de haut-niveau sont *abaissés*, avec leur compilateur, soit en des dérivation, soit en des objets du dépôt. Par exemple, abaisser un paquet crée une dérivation, et abaisser un `plain-file` crée un élément du dépôt. Cela est effectué par la procédure monadique `lower-object`.

`lower-object obj [system] [#:target #f]` [Procédure monadique]

Renvoie comme valeur dans `%store-monad` la dérivation ou l'élément de dépôt correspondant à `obj` pour `system`, en effectuant une compilation croisée pour `target` si `target` est vrai. `obj` doit être un objet auquel est associé un compilateur `gexp`, tel qu'un `<package>`.

`gexp->approximate-sexp gexp` [Procédure]

Il peut parfois être utile de convertir une G-expression en S-expression. Par exemple, certains outils d'analyse statique de style (dits « linters », voir Section 10.8 [Invoquer `guix lint`], page 213) vérifient les phases de compilation des paquets pour détecter des problèmes potentiels. Cette conversion peut être réalisée avec cette procédure. Toutefois, certaines informations peuvent être perdues au cours de l'opération. Plus spécifiquement, les objets abaissables seront remplacés silencieusement par un objet arbitraire – pour l'instant la liste `(*approximate*)`, mais cela pourrait changer.

### 9.13 Invoquer `guix repl`

La commande `guix repl` facilite la programmation de Guix dans Guile en lançant une boucle Guile `read-eval-print`. (REPL) pour la programmation interactive (voir Section “Using Guile Interactively” dans *GNU Guile Reference Manual*), ou en exécutant des scripts Guile (voir Section “guile” dans *GNU Guile Reference Manual*). Par rapport au simple lancement de la commande `guile`, `guix repl` garantit que tous les modules Guix et toutes ses dépendances sont disponibles dans le chemin de recherche.

La syntaxe générale est :

```
guix repl option fichier args]
```

Lorsqu'un argument `file` est fourni, `file` est exécuté comme un script Guile :

```
guix repl mon-script.scm
```

Pour passer des arguments au script, utilisez `--` pour éviter qu'ils ne soient interprétés comme des arguments pour `guix repl` lui-même :

```
guix repl -- mon-script.scm --input=toto.txt
```

Pour rendre exécutable un script directement depuis le shell, en utilisant l'exécutable `guix` qui se trouve dans le chemin de recherche utilisateur-`rice`, ajoutez les deux lignes suivantes en haut du script :

```
#!/usr/bin/env -S guix repl --
!#
```

En l'absence d'un argument de nom de fichier, une REPL guile est lancé, ce qui permet de l'utiliser de manière interactive (voir Section 9.14 [Utiliser Guix de manière interactive], page 177) :

```
$ guix repl
scheme@(guile-user)> ,use (gnu packages base)
scheme@(guile-user)> coreutils
$1 = #<package coreutils@8.29 gnu/packages/base.scm:327 3e28300>
```

En plus, `guix repl` implémente un protocole REPL simple lisible par une machine à utiliser avec (`guix inferior`), un dispositif pour interagir avec des *inférieurs*, des processus séparés qui font tourner une version potentiellement différente de Guix.

Les options disponibles sont les suivantes :

```
--type=type
-t type    Démarrer un REPL du type donné, qui peut être l'un de ces types :

    guile    C'est la valeur par défaut. Elle démarre un REPL Guile standard
             fonctionnel.

    machine  Démarre un REPL qui utilise le protocole lisible par machine. C'est
             le protocole que parle le module (guix inferior).

--listen=extrémité
    Par défaut, guix repl lit depuis l'entrée standard et écrit sur la sortie standard.
    Lorsque cette option est passée, il écouterait plutôt les connexions sur endpoint.
    Voici un exemple d'options valides :

    --listen=tcp:37146
        Accepte les connexions sur localhost, sur le port 31.

    --listen=unix:/tmp/socket
        Accepte les connexions sur le socket Unix-domain /tmp/socket.

--load-path=répertoire
-L répertoire
    Ajoute répertoire au début du chemin de recherche de module de paquets (voir
    Section 9.1 [Modules de paquets], page 104).

    Cela permet à des utilisateurs de définir leur propres paquets et les rendre
    disponibles au script ou au REPL.

-q    Inhiber le chargement du fichier ~/.guile. Par défaut, ce fichier de configura-
    tion est chargé lors de la création d'un fichier guile. REPL.
```

## 9.14 Utiliser Guix de manière interactive

La commande `guix repl` vous donne accès à une *boucle-lecture-évaluation-affichage* (BLÉA ou REPL en anglais) chaleureuse et amicale (voir Section 9.13 [Invoquer `guix repl`], page 176). Si vous commencez la programmation avec Guix — pour définir vos propres paquets, écrire des manifestes, définir des services pour le système Guix ou Guix Home, etc — vous la trouverez sans doute pratique pour jouer avec vos idées sur la REPL.

Si vous utilisez Emacs, le moyen le plus pratique pour cela est d'utiliser Geiser (voir Section 22.3 [La configuration parfaite], page 671), mais vous n'avez pas besoin d'utiliser Emacs pour profiter de la REPL. Lorsque vous utilisez `guix repl` ou `guile` dans un terminal, nous vous recommandons d'utiliser Readline pour la complétion et Colorized pour avoir une sortie colorée. Pour cela, vous pouvez lancer :

```
guix install guile guile-readline guile-colorized
... puis créer un fichier .guile dans votre répertoire personnel, contenant ceci :

(use-modules (ice-9 readline) (ice-9 colorized))

(activate-readline)
(activate-colorized)
```

La REPL vous permet d'évaluer du code Scheme ; vous tapez une expression Scheme sur l'invite, et la REPL affiche ce en quoi elle s'évalue :

```
$ guix repl
scheme@(guix-user)> (+ 2 3)
$1 = 5
scheme@(guix-user)> (string-append "a" "b")
$2 = "ab"
```

Ça devient plus intéressant quand vous commencez à jouer avec Guix sur la REPL. La première chose à faire est « d'importer » le module (`guix`), qui vous donne accès à la partie principale de l'interface de programmation, et peut-être à un ensemble de modules Guix utiles. Vous pouvez taper (`use-modules (guix)`), ce qui est du code Scheme valide pour importer un module (voir Section “Using Guile Modules” dans *manuel de référence de GNU Guile*), mais la REPL fournit la *commande* `use` comme raccourci (voir Section “REPL Commands” dans *manuel de référence de GNU Guile*) :

```
scheme@(guix-user)> ,use (guix)
scheme@(guix-user)> ,use (gnu packages base)
```

Remarquez que les commandes de la REPL sont introduites par une virgule. Une commande REPL comme `use` n'est pas du code Scheme valide, elle est interprétée de manière spéciale par la REPL.

Guix étend la REPL Guile avec des commandes supplémentaires pour votre confort. Parmi celles-ci, la commande `build` est pratique : elle s'assure que l'objet simili-fichier donné est construit, en le construisant si nécessaire, et renvoie le nom de fichier de sa sortie. Dans l'exemple ci-dessous, nous construisons les paquets `coreutils` et `grep`, ainsi qu'un « fichiers calculé » (voir Section 9.12 [G-Expressions], page 166), et nous utilisons la procédure `scandir` pour lister les fichiers du répertoire `/bin` de Grep :

```
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))
construction de /gnu/store/...-x.drv...
$4 = "/gnu/store/...-x"
scheme@(guix-user)> ,use(ice-9 ftw)
scheme@(guix-user)> (scandir (string-append $3 "/bin"))
$5 = ( "." ".." "egrep" "fgrep" "grep")
```

À plus bas niveau, une commande utile est `lower` : elle prend un objet simili-fichier et l'« abaisse » en une dérivation (voir Section 9.10 [Dérivations], page 159) ou un fichier du dépôt :

```
scheme@(guix-user)> ,lower grep
$6 = #<derivation /gnu/store/...-grep-3.6.drv => /gnu/store/...-grep-3.6 7f0e639115f0>
scheme@(guix-user)> ,lower (plain-file "x" "Hello!")
$7 = "/gnu/store/...-x"
```

La liste complète des commandes de la REPL se trouvent en tapant `,help guix` et est donnée ci-dessous pour référence.

- build *objet*** [commande REPL]  
Abaisse *objet* et le construit s'il n'est pas déjà construit, et renvoie le nom de fichier de ses sorties.
- lower *objet*** [commande REPL]  
Abaisse *objet* en une dérivation ou un nom de fichier du dépôt et le renvoie.
- verbosity *niveau*** [commande REPL]  
Change la verbosité des construction à *niveau*.  
C'est similaire à l'option en ligne de commande `--verbosity` (voir Section 10.1.1 [Options de construction communes], page 180) : le niveau 0 signifie le silence total, le niveau 1 montre les événements de construction et les niveaux plus élevés affichent les journaux de construction.
- run-in-store *exp*** [commande REPL]  
Lance *exp*, une expression monadique dans la monade du dépôt. Voir Section 9.11 [La monade du dépôt], page 162, pour plus d'information.
- enter-store-monad** [commande REPL]  
Entre dans une nouvelle REPL pour évaluer des expressions monadiques (voir Section 9.11 [La monade du dépôt], page 162). Vous pouvez quitter la REPL « interne » en tapant `,q`.

## 10 Utilitaires

Cette section décrit les utilitaires en ligne de commande de Guix. certains sont surtout faits pour les personnes qui écrivent de nouvelles définitions de paquets tandis que d'autres sont plus utiles pour une utilisation générale. Ils complètent l'interface de programmation Scheme de Guix d'une manière pratique.

### 10.1 Invoquer guix build

La commande `guix build` construit des paquets ou des dérivations et leurs dépendances et affiche les chemins du dépôt qui en résulte. Remarquez qu'elle ne modifie pas le profil de l'utilisateur — c'est le travail de la commande `guix package` (voir Section 6.2 [Invoquer guix package], page 44). Ainsi, elle est surtout utile pour les personnes qui développent la distribution.

La syntaxe générale est :

```
guix build options package-or-derivation...
```

Par exemple, la commande suivante construit la dernière version d'Emacs et de Guile, affiche leur journaux de construction et enfin affiche les répertoires des résultats :

```
guix build emacs guile
```

De même, la commande suivante construit tous les paquets disponibles :

```
guix build --quiet --keep-going \  
$(guix package -A | awk '{ print $1 "@" $2 }')
```

*package-or-derivation* peut être soit le nom d'un paquet trouvé dans la distribution logicielle comme `coreutils`, soit `coreutils@8.20`, soit une dérivaison comme `/gnu/store/...-coreutils-8.19.drv`. Dans le premier cas, la commande cherchera un paquet avec le nom correspondant (et éventuellement la version) dans les modules de la distribution GNU (voir Section 9.1 [Modules de paquets], page 104).

Autrement, l'option `--expression` peut être utilisée pour spécifier une expression Scheme qui s'évalue en un paquet ; c'est utile lorsqu'il est nécessaire de faire la distinction entre plusieurs paquets ou variantes de paquets portant le même nom.

Il peut y avoir aucune, une ou plusieurs *options*. Les options disponibles sont décrites dans les sous-sections ci-dessous.

#### 10.1.1 Options de construction communes

Un certain nombre d'options qui contrôlent le processus de construction sont communes avec `guix build` et les autres commandes qui peuvent générer des constructions, comme `guix package` ou `guix archive`. Voici ces options :

`--load-path=répertoire`

`-L répertoire`

Ajoute *répertoire* au début du chemin de recherche de module de paquets (voir Section 9.1 [Modules de paquets], page 104).

Cela permet à des utilisateurs de définir leur propres paquets et les rendre disponibles aux outils en ligne de commande.

**--keep-failed**

**-K** Garde l'arborescence de construction des constructions en échec. Ainsi, si une construction échoue, son arborescence de construction est préservée dans `/tmp`, dans un répertoire dont le nom est affiché à la fin du journal de construction. Cela est utile pour déboguer des échecs de construction. Voir Section 10.1.4 [Débogage des échecs de construction], page 193, pour des astuces sur la manière de déboguer des problèmes de construction.

Cette option implique `--no-offload`, et elle n'a pas d'effet quand elle est connectée à un démon distant avec une `guix://` URI (voir Section 9.9 [Le dépôt], page 157).

**--keep-going**

**-k** Continue lorsque certaines dérivations échouent ; ne s'arrête que lorsque toutes les constructions ont soit réussies, soit échouées.

Le comportement par défaut est de s'arrêter dès qu'une des dérivations spécifiées échoue.

**--dry-run**

**-n** Ne pas construire les dérivations.

**--fallback**

Lorsque la substitution d'un binaire pré-compilé échoue, construit les paquets localement à la place (voir Section 6.3.6 [Échec de substitution], page 58).

**--substitute-urls=urls**

Considère *urls* comme une liste d'URL de sources de substituts séparés par des espaces, et remplace la liste par défaut d'URL de `guix-daemon` (voir [guix-daemon URLs], page 18).

Cela signifie que les substituts peuvent être téléchargés depuis *urls*, tant qu'ils sont signés par une clef autorisée par l'administrateur système (voir Section 6.3 [Substituts], page 54).

Lorsque *urls* est la chaîne vide, cela a pour effet de désactiver la substitution.

**--no-substitutes**

Ne pas utiliser de substitut pour les résultats de la construction. C'est-à-dire, toujours construire localement plutôt que de permettre le téléchargement de binaires pré-construits (voir Section 6.3 [Substituts], page 54).

**--no-grafts**

Ne pas « greffer » les paquets. En pratique, cela signifie que les mises à jour des paquets disponibles comme des greffes ne sont pas appliquées. Voir Chapitre 19 [Mises à jour de sécurité], page 657, pour plus d'information sur les greffes.

**--rounds=n**

Construit chaque dérivation *n* fois d'affilé, et renvoie une erreur si les constructions consécutives ne sont pas identiques bit-à-bit.

Cela est une manière utile pour détecter des processus de construction non déterministes. Les processus de construction non déterministes sont problématiques car ils rendent pratiquement impossible la *vérification* par les



utilisateurs de l'authenticité de binaires tiers. Voir Section 10.12 [Invoquer guix challenge], page 226, pour plus d'informations.

Lorsqu'utilisé avec `--keep-failed`, la sortie différente est gardée dans le dépôt sous `/gnu/store/...-check`. Cela rend plus facile l'étude des différences entre les deux résultats.

`--no-offload`

N'essaye pas de décharger les constructions vers d'autres machines (voir Section 2.4.2 [Réglages du déchargement du démon], page 12). C'est-à-dire que tout sera construit localement au lieu de décharger les constructions à une machine distante.

`--max-silent-time=secondes`

Lorsque le processus de construction ou de substitution restent silencieux pendant plus de *secondes*, le terminer et rapporter une erreur de construction.

Par défaut, le paramètre du démon est pris en compte (voir Section 2.5 [Invoquer guix-daemon], page 17).

`--timeout=secondes`

De même, lorsque le processus de construction ou de substitution dure plus de *secondes*, le terminer et rapporter une erreur de construction.

Par défaut, le paramètre du démon est pris en compte (voir Section 2.5 [Invoquer guix-daemon], page 17).

`-v [niveau]`

`--verbosity=niveau`

Utiliser le *niveau* de verbosité, en tant qu'entier. 0 signifie qu'aucune sortie n'est produite, 1 signifie une sortie silencieuse ; 2 est similaire à 1 mais affiche aussi les URL des téléchargements ; 3 montre tous les journaux de construction sur la sortie d'erreur standard.

`--cores=n`

`-c n` Permet d'utiliser jusqu'à *n* cœurs du CPU pour la construction. La valeur spéciale 0 signifie autant de cœurs que possible.

`--max-jobs=n`

`-M n` Permettre au maximum à *n* de construire des jobs en parallèle. Voir Section 2.5 [Invoquer guix-daemon], page 17, pour plus de détails sur cette option et l'option équivalente `guix-daemon`.

`--debug=niveau`

Produire une sortie de débogage qui provient du démon de construction. *niveau* doit être un entier entre 0 et 5 ; plus grand est ce nombre, plus verbeuse sera la sortie. Indiquer un niveau de 4 ou plus peut être utile pour déboguer des problèmes d'installation avec le démon de construction.

Sous le capot, `guix build` est surtout une interface à la procédure `package-derivation` du module (`guix packages`), et à la procédure `build-derivations` du module (`guix derivations`).

En plus des options passées explicitement par la ligne de commande, `guix build` et les autres commandes `guix` qui peuvent effectuer des constructions prennent en compte la variable d'environnement `GUIX_BUILD_OPTIONS`.

`GUIX_BUILD_OPTIONS` [Variable d'environnement]

Les utilisateurs peuvent définir cette variable à une liste d'options de la ligne de commande qui seront automatiquement utilisées par `guix build` et les autres commandes `guix` qui peuvent effectuer des constructions, comme dans l'exemple suivant :

```
$ export GUIX_BUILD_OPTIONS="--no-substitutes -c 2 -L /toto/titi"
```

Ces options sont analysées indépendamment, et le résultat est ajouté aux options de la ligne de commande analysées.

### 10.1.2 Options de transformation de paquets

Un autre ensemble d'options de la ligne de commande supportés par `guix build` et aussi `guix package` sont les *options de transformation de paquets*. Ce sont des options qui rendent possible la définition de *variantes de paquets* — par exemple, des paquets construit à partir de sources différentes. C'est une manière simple de créer des paquets personnalisés à la volée sans avoir à taper les définitions de variantes de paquets (voir Section 9.2 [Définition des paquets], page 105).

Les options de transformation des paquets sont préservées dans les mises à jour : `guix upgrade` tente d'appliquer aux paquets mis à jour les options de transformation initialement utilisées lors de la création du profil.

Les options disponibles sont énumérées ci-dessous. La plupart des commandes les prennent en charge, ainsi qu'une option `--help-transform` qui liste toutes les options disponibles et un synopsis (ces options ne sont pas affichées dans la sortie `--help` par souci de concision).

`--tune[=cpu]`

Utilise les versions des paquets marqués comme « réglables » optimisées pour *cpu*. Lorsque *cpu* est `native`, ou s'il est omis, règle pour le CPU sur lequel la commande `guix` est lancée.

Les noms de *cpu* valides sont ceux reconnus par le compilateur sous-jacent, par défaut la collection de compilateurs de GNU (GCC). Sur les processeurs x86\_64, cela comprend les noms des CPU comme `nehalem`, `haswell` et `skylake` (voir Section "x86 Options" dans *Using the GNU Compiler Collection (GCC)*).

Au fur et à mesure de la sortie de nouvelles générations de CPU, de nouvelles instructions sont ajoutées à l'ensemble d'instruction standard de l'architecture (ISA), en particulier des instructions pour le calcul parallèle instruction-unique/données-multiples (SIMD). Par exemple, alors que les CPU Core2 et Skylake implémentent tous deux l'ISA x86\_64, seul ce dernier prend en charge les instructions SIMD AVX2.

Le principal bénéfice attendu de `--tune` est que les programmes peuvent utiliser ces fonctionnalités SIMD *et* qu'ils n'ont pas déjà un mécanisme pour choisir le bon code optimisé à l'exécution. Les paquets qui possèdent la propriété `tunable?` sont considérés comme des *paquets réglables* par l'option `--tune` ; une définition de paquet avec la bonne propriété ressemble à ceci :

```
(package
```

```
(name "hello-simd")
;; ...

;; Ce paquet peut bénéficier des extensions SIMD donc
;; on le marque « réglable ».
(properties '((tunable? . #t)))
```

Les autres paquets ne sont pas considérés comme réglables. Cela permet à Guix d'utiliser des binaires génériques dans le cas où le réglage pour un CPU particulier ne donnera probablement aucun avantage.

Les paquets réglés sont construits avec `-march=CPU` ; sous le capot, l'option `-march` est passée à l'enveloppe par une enveloppe du compilateur. Comme la machine de construction peut ne pas être capable de lancer du code pour la micro-architecture CPU cible, la suite de tests n'est pas lancée lors de la construction d'un paquet réglé.

Pour réduire les reconstructions au minimum, les paquets réglés sont *greffés* sur les paquets qui en dépendent (voir Chapitre 19 [Mises à jour de sécurité], page 657). Ainsi, utiliser `--no-grafts` annule l'effet de `--tune`.

Nous appelons cette technique le *multi-versionnement des paquets* : on peut construire plusieurs variantes d'un paquet réglable, une pour chaque variante de CPU. C'est la contrepartie à gros grain du *multi-versionnement fonctionnel* implémenté par la chaîne d'outils de GNU (voir Section “Function Multiversioning” dans *Using the GNU Compiler Collection (GCC)*).

```
--with-source=source
--with-source=paquet=source
--with-source=paquet@version=source
```

Utilise *source* comme la source de *paquet*, et *version* comme son numéro de version. *source* doit être un nom de fichier ou une URL, comme pour `guix download` (voir Section 10.3 [Invoquer guix download], page 195).

Lorsque *paquet* est omis, la commande utilisera le nom de paquet spécifié par la base de *source* — p. ex. si *source* est `/src/guix-2.0.10.tar.gz`, le paquet correspondant est `guile`.

De même, lorsque *version* est omis, la chaîne de version est inférée à partir de *source* ; dans l'exemple précédent, il s'agit de `2.0.10`.

Cette option permet aux utilisateurs d'essayer des versions des paquets différentes de celles fournies par la distribution. L'exemple ci-dessous télécharge `ed-1.7.tar.gz` depuis un miroir GNU et l'utilise comme source pour le paquet `ed` :

```
guix build ed --with-source=mirror://gnu/ed/ed-1.4.tar.gz
```

En tant que développeur·euse, `--with-source` permet de tester facilement des versions bêta, et même de vérifier leur impact sur les paquets qui en dépendent :

```
guix build elogind --with-source=.../shepherd-0.9.0rc1.tar.gz
... ou pour construire un dépôt de gestion de version dans un environnement vierge :
```

```
$ git clone git://git.sv.gnu.org/guix.git
$ guix build guix --with-source=guix@1.0=./guix
```

#### `--with-input=paquet=remplaçant`

Remplace la dépendance sur *paquet* par une dépendance à *remplaçant*. *paquet* doit être un nom de paquet et *remplaçant* doit être une spécification de paquet comme *guile* ou *guile@1.8*.

Par exemple, la commande suivante construit Guix, mais remplace sa dépendance à la version stable actuelle de Guile par une dépendance à une ancienne version de Guile, *guile@2.0* :

```
guix build --with-input=guile=guile@2.0 guix
```

C'est un remplacement récursif profond. Donc dans cet exemple, à la fois *guix* et ses dépendances *guile-json* (qui dépend aussi de *guile*) sont reconstruits avec *guile@2.0*.

Cette option est implémentée avec la procédure Scheme `package-input-rewriting` (voir Section 9.2 [Définition des paquets], page 105).

#### `--with-graft=paquet=remplaçant`

Cette option est similaire à `--with-input` mais avec une différence importante : plutôt que de reconstruire la chaîne de dépendance complète, *remplaçant* est construit puis greffé sur les binaires qui référençaient initialement *paquet*. Voir Chapitre 19 [Mises à jour de sécurité], page 657, pour plus d'information sur les greffes.

Par exemple, la commande ci-dessous greffe la version 3.5.4 de GnuTLS sur Wget et toutes ses dépendances, en remplaçant les références à la version actuelle de GnuTLS à laquelle ils se réfèrent actuellement :

```
guix build --with-graft=gnutls=gnutls@3.5.4 wget
```

Cela a l'avantage d'être bien plus rapide que de tout reconstruire. Mais il y a un piège : cela ne fonctionne que si *paquet* et *remplaçant* sont strictement compatibles — par exemple, s'ils fournissent une bibliothèque, l'interface binaire applicative (ABI) de ces bibliothèques doivent être compatibles. Si *remplaçant* est incompatible avec *paquet*, alors le paquet qui en résulte peut devenir inutilisable. À utiliser avec précaution !

#### `--with-debug-info=paquet`

Construire *paquet* de manière à préserver ses informations de débogage et les greffer sur les paquets qui en dépendent. Cela est utile si *paquet* ne fournit pas déjà les informations de débogage sous forme de sortie `debug` (voir Chapitre 17 [Installer les fichiers de débogage], page 652).

Par exemple, supposons que vous subissiez un plantage de Inkscape et que vous vouliez voir ce qui se passe dans GLib, une bibliothèque au fond du graphe de dépendance d'Inkscape. GLib n'a pas de sortie `debug`, donc le débogage est difficile. Heureusement, vous reconstruisez la GLib avec les informations de débogage et vous l'installez dans Inkscape :

```
guix install inkscape --with-debug-info=glib
```

Seule GLib doit être recompilée, ce qui prend un temps raisonnable. Voir Chapitre 17 [Installer les fichiers de débogage], page 652, pour plus d'informations.

**Remarque:** Sous le capot, cette option fonctionne en passant par le ‘#:strip-binaries ? #f’ au système de construction du paquet qui nous intéresse (voir Section 9.5 [Systèmes de construction], page 124). La plupart des systèmes de compilation supportent cette option, mais certains ne le font pas. Dans ce cas, une erreur se produit.

De même, si un paquet C/C++ est construit sans `-g` (ce qui est rarement le cas), les informations de débogage resteront indisponibles même si `#:strip-binaries?` est faux.

`--with-c-toolchain=package=toolchain`

Cette option modifie la compilation de *package* et de tout ce qui en dépend afin qu’ils soient construits avec *toolchain* au lieu de la chaîne d’outils GNU par défaut pour C/C++.

Regardez cet exemple :

```
guix build octave-cli \
  --with-c-toolchain=fftw=gcc-toolchain@10 \
  --with-c-toolchain=fftwf=gcc-toolchain@10
```

La commande ci-dessus construit une variante des paquets `fftw` et `fftwf` en utilisant la version 10 de `gcc-toolchain` au lieu de la chaîne d’outils par défaut, puis construit une variante de l’interface en ligne de commande GNU Octave en les utilisant. GNU Octave lui-même est également construit avec `gcc-toolchain@10`.

Cet autre exemple construit la bibliothèque Hardware Locality (`hwloc`) et ses dépendances jusqu’à `intel-mpi-benchmarks` avec le compilateur Clang C :

```
guix build --with-c-toolchain=hwloc=clang-toolchain \
  intel-mpi-benchmarks
```

**Remarque:** Il peut y avoir des incompatibilités d’interface binaire d’application (ABI) entre les chaînes d’outils. Cela est particulièrement vrai pour la bibliothèque standard C++ et les bibliothèques de support d’exécution telles que celle d’OpenMP. En reconstruisant toutes les dépendances avec la même chaîne d’outils, `--with-c-toolchain` minimise les risques d’incompatibilité mais ne peut pas les éliminer entièrement. Choisissez *package* judicieusement.

`--with-git-url=paquet=url`

Construire *paquet* depuis le dernier commit de la branche `master` du dépôt sur *url*. Les sous-modules Git du dépôt sont récupérés, récursivement.

Par exemple, la commande suivante construit la bibliothèque Python NumPy avec le dernier commit de la branche `master` de Python lui-même :

```
guix build python-numpy \
  --with-git-url=python=https://github.com/python/cpython
```

Cette option peut aussi être combinée avec `--with-branch` ou `--with-commit` (voir plus bas).

Évidemment, comme cela utilise le dernier commit d'une branche donnée, le résultat d'une telle commande varie avec le temps. Néanmoins c'est une manière pratique pour reconstruire des piles logicielles entières avec le dernier commit d'un ou plusieurs paquets. C'est particulièrement pratique dans le contexte d'une intégration continue.

Les clones sont gardés dans un cache dans `~/.cache/guix/checkouts` pour accélérer les accès consécutifs au même dépôt. Vous pourriez vouloir le nettoyer de temps en temps pour récupérer de l'espace disque.

#### `--with-branch=paquet=branche`

Construire *paquet* à partir du dernier commit de la *branche*. Si le champ *source* de *paquet* est une origine avec la méthode `git-fetch` (voir Section 9.2.2 [référence de origin], page 113) ou un objet `git-checkout`, l'URL du dépôt est récupérée à partir de cette *source*. Sinon, vous devez utiliser `--with-git-url` pour spécifier l'URL du dépôt Git.

Par exemple, la commande suivante construit `guile-sqlite3` à partir du dernier commit de sa branche `master`, puis construit `guix` (qui en dépend) et `cuirass` (qui dépend de `guix`) avec cette construction spécifique de `guile-sqlite3` :

```
guix build --with-branch=guile-sqlite3=master cuirass
```

#### `--with-commit=paquet=commit`

Cela est similaire à `--with-branch`, sauf qu'elle construite à partir de *commit* au lieu du sommet d'une branche. *commit* doit être un identifiant SHA1 de commit Git valide, un tag ou un identifiant dans le style de `git describe` comme `1.0-3-gabc123`.

#### `--with-patch=paquet=fichier`

Ajoute *fichier* à la liste des correctifs appliqués à *paquet*, où *paquet* est une spécification comme `python@3.8` ou `glibc`. *fichier* doit contenir un correctif ; il est appliqué avec les drapeaux spécifiés dans l'*origin* de *paquet* (voir Section 9.2.2 [référence de origin], page 113), qui par défaut inclus `-p1` voir Section "patch Directories" dans *Comparing and Merging Files*).

Par exemple, la commande ci-dessous reconstruit Coreutils avec la bibliothèque C de GNU (glibc) corrigée avec le correctif donné :

```
guix build coreutils --with-patch=glibc=./glibc-frob.patch
```

Dans cet exemple, glibc lui-même ainsi que tout ce qui mène à Coreutils dans le graphe des dépendances est reconstruit.

#### `--with-latest=paquet`

Alors vous aimez les toutes dernières technologies ? Cette option est faite pour vous ! Elle remplace les occurrence de *paquet* dans le graphe des dépendances avec sa toute dernière version en amont, telle que rapportée par `guix refresh` (voir Section 10.6 [Invoquer guix refresh], page 205).

Elle fait cela en déterminant la dernière version publiée en amont (si possible), en la téléchargeant et en l'authentifiant *si* elle propose une signature OpenPGP.

Par exemple, la commande ci-dessous construit Guix avec la dernière version de Guile-JSON :

```
guix build guix --with-latest=guile-json
```

Il y a des limites. Déjà, dans les cas où l'outil ne peut pas ou ne sait pas comment authentifier le code source, vous risquez de charger du code malveillant ; un avertissement est émis dans ce cas. Ensuite, cette option change simplement la source utilisée dans les définitions existantes des paquets, ce qui n'est pas toujours suffisant : il peut y avoir des dépendances supplémentaires qui doivent être ajoutées, des correctifs à appliquer, et plus généralement tout le travail d'assurance qualité que les développeurs de Guix font habituellement sera absent.

On vous aura prévenu ! Dans tous les autres cas, c'est une méthode pour rester à la hauteur qui a du punch ! Nous vous encourageons à soumettre des correctifs pour les définitions des paquets quand vous aurez testé une mise à jour (voir Chapitre 22 [Contribuer], page 668).

```
--without-tests=paquet
```

Construire *paquet* sans lancer la suite de tests. Cela peut être utile dans les situations où vous voulez éviter la longue série de tests d'un paquet intermédiaire, ou si une suite de tests de paquet échoue dans un mode non déterministe. Il doit être utilisé avec précaution car l'exécution de la suite de tests est un bon moyen de s'assurer qu'un paquet fonctionne comme prévu.

L'arrêt des tests conduit à un autre élément du dépôt. Par conséquent, lorsque vous utilisez cette option, tout ce qui dépend de *paquet* doit être reconstruit, comme dans cet exemple :

```
guix install --without-tests=python python-notebook
```

La commande ci-dessus installe *python-notebook* par dessus *python*, construit sans exécuter sa suite de tests. Pour ce faire, elle reconstruit également tout ce qui dépend de *python*, y compris *python-notebook* lui-même.

En interne, `--without-tests` repose sur le changement de l'option `#:tests?` de la phase `check` d'un paquet (voir Section 9.5 [Systèmes de construction], page 124). Notez que certains paquets utilisent une phase `check` personnalisée qui ne respecte pas le paramètre `#:tests? #f`. Par conséquent, `--without-tests` n'a aucun effet sur ces paquets.

Vous vous demandez comme faire la même chose dans du code Scheme, par exemple dans votre manifeste, ou comme écrire votre propre transformation de paquets ? Voir Section 9.3 [Définition de variantes de paquets], page 116, pour un aperçu des interfaces de programmation disponibles.

### 10.1.3 Options de construction supplémentaires

Les options de la ligne de commande ci-dessous sont spécifiques à `guix build`.

```
--quiet
```

```
-q
```

Construire silencieusement, sans afficher les journaux de construction ; c'est équivalent à `--verbosity=0`. À la fin, le journal de construction est gardé dans `/var` (ou similaire) et on peut toujours l'y trouver avec l'option `--log-file`.

```
--file=fichier
-f fichier
```

Construit le paquet, la dérivation ou l'objet simili-fichier en lequel le code dans *file* s'évalue (voir Section 9.12 [G-Expressions], page 166).

Par exemple, *file* peut contenir une définition de paquet comme ceci (voir Section 9.2 [Définition des paquets], page 105) :

```
(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
      "0ssi1wpaf7plaswqqjwigppsg5fyh99vdlb9kz17c9lng89ndq1i"))))
  (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+))
```

*file* peut également contenir une représentation JSON d'une ou plusieurs définitions de paquets. L'exécution de `guix build -f` sur `hello.json` avec le contenu suivant entraînerait la construction des paquets `myhello` et `greeter` :

```
[
  {
    "name": "myhello",
    "version": "2.10",
    "source": "mirror://gnu/hello/hello-2.10.tar.gz",
    "build-system": "gnu",
    "arguments": {
      "tests?": false
    }
  },
  {
    "home-page": "https://www.gnu.org/software/hello/",
    "synopsis": "Hello, GNU world: An example GNU package",
    "description": "GNU Hello prints a greeting.",
    "license": "GPL-3.0+",
    "native-inputs": ["gettext"]
  }
]
```



```

    "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`

Construire tous les paquets listés dans un *manifest* donné (voir [profile-manifest], page 48).

`--expression=expr`

`-e expr` Construit le paquet ou la dérivation en lequel *expr* s'évalue.

Par exemple, *expr* peut être `(@ (gnu packages guile) guile-1.8)`, qui désigne sans ambiguïté cette variante spécifique de la version 1.8 de Guile.

Autrement, *exp* peut être une G-expression, auquel cas elle est utilisée comme un programme de construction passé à `gexp->derivation` (voir Section 9.12 [G-Expressions], page 166).

Enfin, *expr* peut se référer à une procédure monadique à au moins un argument (voir Section 9.11 [La monade du dépôt], page 162). La procédure doit renvoyer une dérivation comme une valeur monadique, qui est ensuite lancée à travers `run-with-store`.

`--source`

`-S` Construit les dérivation source des paquets, plutôt que des paquets eux-mêmes.

Par exemple, `guix build -S gcc` renvoie quelque chose comme `/gnu/store/...-gcc-4.7.2.tar.bz2`, qui est l'archive des sources de GCC.

L'archive des sources renvoyée est le résultat de l'application des correctifs et des extraits de code éventuels spécifiés dans le champ `origin` du paquet (voir Section 9.2 [Définition des paquets], page 105).

Comme avec les autres dérivation, le résultat de la construction des sources peut être vérifié avec l'option `--check` (voir [vérification de la construction], page 192). C'est utile pour valider que les source (éventuellement déjà construites ou substituées, donc en cache) correspondent au hash déclaré.

Notez que `guix build -S` compile seulement les sources des paquets spécifiés. Les sources de dépendances statiquement liées ne sont pas incluses et sont en elles-mêmes insuffisantes pour reproduire les paquets.

**--sources**

Récupère et renvoie la source de *package-or-derivation* et toute ses dépendances, récursivement. C'est pratique pour obtenir une copie locale de tous les codes sources requis pour construire *packages*, ce qui vous permet de les construire plus tard même sans accès au réseau. C'est une extension de l'option **--source** et elle peut accepter l'un des arguments facultatifs suivants :

**package** Cette valeur fait que l'option **--sources** se comporte comme l'option **--source**.

**all** Construit les dérivations des sources de tous les paquets, dont les sources qui pourraient être listées dans **inputs**. C'est la valeur par défaut.

```
$ guix build --sources tzdata
The following derivations will be built:
/gnu/store/...-tzdata2015b.tar.gz.drv
/gnu/store/...-tzcode2015b.tar.gz.drv
```

**transitive**

Construire les dérivations des sources de tous les paquets, ainsi que toutes celles des entrées transitives des paquets. On peut par exemple utiliser cette option pour précharger les sources des paquets pour les construire plus tard hors ligne.

```
$ guix build --sources=transitive tzdata
The following derivations will be built:
/gnu/store/...-tzcode2015b.tar.gz.drv
/gnu/store/...-findutils-4.4.2.tar.xz.drv
/gnu/store/...-grep-2.21.tar.xz.drv
/gnu/store/...-coreutils-8.23.tar.xz.drv
/gnu/store/...-make-4.1.tar.xz.drv
/gnu/store/...-bash-4.3.tar.xz.drv
...
```

**--system=ystème****-s système**

Essayer de construire pour *system* — p. ex. **i686-linux** — au lieu du type de système de l'hôte. La commande **guix build** vous permet de répéter cette option plusieurs fois, auquel cas elle construit pour tous les systèmes spécifiés ; les autres commandes ignorent les options **-s** supplémentaires.

**Remarque:** Le drapeau **--system** est utilisé pour une compilation *native* et ne doit pas être confondu avec une compilation croisée. Voir **--target** ci-dessous pour des informations sur la compilation croisée.

Un exemple d'utilisation de ceci sur les systèmes Linux, qui peut émuler différentes personnalités. Par exemple, passer **--system=i686-linux** sur un système **x86\_64-linux** ou **--system=armhf-linux** sur un système **aarch64-linux** vous permet de construire des paquets dans un environnement 32-bit complet.

**Remarque:** La possibilité de construire pour un système `armhf-linux` est activé sans condition sur les machines `aarch64-linux`, bien que certaines puces `aarch64` n'en soient pas capables, comme les ThunderX.

De même, lorsque l'émulation transparente avec QEMU et `binfmt_misc` est activée (voir Section 12.9.29 [Services de virtualisation], page 504), vous pouvez construire pour n'importe quel système pour lequel un gestionnaire QEMU `binfmt_misc` est installé.

Les constructions pour un autre système que celui de la machine que vous utilisez peuvent aussi être déchargées à une machine distante de la bonne architecture. Voir Section 2.4.2 [Réglages du déchargement du démon], page 12, pour plus d'information sur le déchargement.

- `--target=triplet`  
Effectuer une compilation croisée pour *triplet* qui doit être un triplet GNU valide, comme "`aarch64-linux-gnu`" (voir Section "Specifying Target Triplets" dans *Autoconf*).
- `--list-systems`  
Liste tous les systèmes pris en charge, qui peuvent être passés en argument à `--system`.
- `--list-targets`  
Liste toutes les cibles prises en charge, qui peuvent être passées en argument à `--target`.
- `--check` Reconstruit les *package-or-derivation*, qui sont déjà disponibles dans le dépôt et lève une erreur si les résultats des constructions ne sont pas identiques bit-à-bit. Ce mécanisme vous permet de vérifier si les substituts précédemment installés sont authentiques (voir Section 6.3 [Substituts], page 54) ou si le résultat de la construction d'un paquet est déterministe. Voir Section 10.12 [Invoquer guix challenge], page 226, pour plus d'informations et pour les outils. Lorsqu'utilisé avec `--keep-failed`, la sortie différente est gardée dans le dépôt sous `/gnu/store/...-check`. Cela rend plus facile l'étude des différences entre les deux résultats.
- `--repair` Essaie de réparer les éléments du dépôt spécifiés, s'ils sont corrompus, en les téléchargeant ou en les construisant à nouveau. Cette opération n'est pas atomique et donc restreinte à l'utilisateur `root`.
- `--derivations`
- `-d` Renvoie les chemins de dérivation, et non les chemins de sortie, des paquets donnés.
- `--root=fichier`
- `-r fichier`  
Fait de *fichier* un lien symbolique vers le résultat, et l'enregistre en tant que racine du ramasse-miettes.

En conséquence, les résultats de cette invocation de `guix build` sont protégés du ramasse-miettes jusqu'à ce que *fichier* soit supprimé. Lorsque cette option est omise, les constructions sont susceptibles d'être glanées.

#### `--log-file`

Renvoie les noms des journaux de construction ou les URL des *package-orientation* donnés ou lève une erreur si les journaux de construction sont absents.

Cela fonctionne indépendamment de la manière dont les paquets ou les dérivations sont spécifiées. Par exemple, les invocations suivantes sont équivalentes :

```
guix build --log-file $(guix build -d guile)
guix build --log-file $(guix build guile)
guix build --log-file guile
guix build --log-file -e '(@ (gnu packages guile) guile-2.0)'
```

Si un journal n'est pas disponible localement, à moins que `--no-substitutes` ne soit passé, la commande cherche un journal correspondant sur l'un des serveurs de substituts (tels que spécifiés avec `--substitute-urls`).

Donc par exemple, imaginons que vous souhaitez voir le journal de construction de GDB sur `aarch64`, mais que vous n'avez qu'une machine `x86_64` :

```
$ guix build --log-file gdb -s aarch64-linux
https://ci.guix.gnu.org/log/...-gdb-7.10
```

Vous pouvez accéder librement à une vaste bibliothèque de journaux de construction !

### 10.1.4 Débogage des échecs de construction

Lors de la définition d'un nouveau paquet (voir Section 9.2 [Définition des paquets], page 105), vous passerez probablement du temps à déboguer et modifier la construction jusqu'à ce que ça marche. Pour cela, vous devez effectuer les commandes de construction vous-même dans un environnement le plus proche possible de celui qu'utilise le démon de construction.

Pour cela, la première chose à faire est d'utiliser l'option `--keep-failed` ou `-K` de `guix build`, qui gardera la construction échouée de l'arborescence dans `/tmp` ou le répertoire spécifié par `TMPDIR` (voir Section 10.1.1 [Options de construction communes], page 180).

À partir de là, vous pouvez vous déplacer dans l'arborescence de construction et sourcer le fichier `environment-variables`, qui contient toutes les variables d'environnement qui étaient définies lorsque la construction a échoué. Disons que vous déboguez un échec de construction dans le paquet `toto` ; une session typique ressemblerait à cela :

```
$ guix build toto -K
... build fails
$ cd /tmp/guix-build-toto.drv-0
$ source ./environment-variables
$ cd toto-1.2
```

Maintenant, vous pouvez invoquer les commandes comme si vous étiez le démon (presque) et corriger le processus de construction.

Parfois il arrive que, par exemple, les tests d'un paquet réussissent lorsque vous les lancez manuellement mais échouent quand ils sont lancés par le démon. Cela peut arriver parce que le démon tourne dans un conteneur où, contrairement à notre environnement au-dessus, l'accès réseau est indisponible, `/bin/sh` n'existe pas, etc. (voir Section 2.4.1 [Réglages de l'environnement de construction], page 11).

Dans ce cas, vous pourriez avoir besoin de lancer le processus de construction dans un conteneur similaire à celui que le démon crée :

```
$ guix build -K toto
...
$ cd /tmp/guix-build-toto.drv-0
$ guix shell --no-grafts -C -D toto strace gdb
[env]# source ./environment-variables
[env]# cd toto-1.2
```

Ici, `guix shell -C` crée un conteneur et démarre un nouveau shell à l'intérieur (voir Section 8.1 [Invoquer `guix shell`], page 83). La partie `strace gdb` ajoute les commandes `strace` et `gdb` dans le conteneur, ce qui pourrait s'avérer utile pour le débogage. L'option `--no-grafts` s'assure qu'on obtienne le même environnement, avec des paquets non greffés (voir Chapitre 19 [Mises à jour de sécurité], page 657, pour plus d'informations sur les greffes).

Pour obtenir un conteneur plus proche de ce qui serait utilisé par le démon de construction, on peut enlever `/bin/sh` :

```
[env]# rm /bin/sh
```

(Ne vous inquiétez pas, c'est sans danger : tout cela se passe dans un conteneur jetable créé par `guix shell`.)

La commande `strace` n'est probablement pas dans le chemin de recherche, mais on peut lancer :

```
[env]# $GUIX_ENVIRONMENT/bin/strace -f -o log make check
```

De cette manière, non seulement vous aurez reproduit les variables d'environnement utilisées par le démon, mais vous lancerez aussi le processus de construction dans un conteneur similaire à celui utilisé par le démon.

## 10.2 Invoquer `guix edit`

Tant de paquets, tant de fichiers source ! La commande `guix edit` facilite la vie des utilisateurs et des empaqueteurs en plaçant leur éditeur sur le fichier source qui contient la définition des paquets spécifiés. Par exemple :

```
guix edit gcc@4.9 vim
```

lance le programme spécifié dans la variable d'environnement `VISUAL` ou dans la variable d'environnement `EDITOR` pour visionner la recette de GCC 4.9.3 et celle de Vim.

Si vous utilisez une copie du dépôt Git de Guix (voir Section 22.1 [Construire depuis Git], page 668), ou que vous avez créé vos propres paquets dans `GUIX_PACKAGE_PATH` (voir Section 9.1 [Modules de paquets], page 104), vous pourrez modifier les recettes des paquets. Sinon, vous pourrez examiner les recettes en lecture-seule des paquets actuellement dans le dépôt.

Au lieu de `GUIX_PACKAGE_PATH`, l'option de la ligne de commande `--load-path=directory` (ou en bref `-L directory`) vous permet d'ajouter *directory* au début du chemin de recherche du module de paquet et donc de rendre vos propres paquets visibles.

### 10.3 Invoquer guix download

Lorsqu'on écrit une définition de paquet, on a généralement besoin de télécharger une archive des sources, calculer son hash SHA256 et écrire ce hash dans la définition du paquet (voir Section 9.2 [Définition des paquets], page 105). L'outil `guix download` aide à cette tâche : il télécharge un fichier à l'URL donné, l'ajoute au dépôt et affiche à la fois son nom dans le dépôt et son hash SHA56.

Le fait que le fichier téléchargé soit ajouté au dépôt économise la bande passante : quand on construit ensuite le paquet nouvellement défini avec `guix build`, l'archive des sources n'a pas besoin d'être à nouveau téléchargée puisqu'elle se trouve déjà dans le dépôt. C'est aussi une manière pratique de garder des fichiers temporairement, qui pourront ensuite être supprimés (voir Section 6.5 [Invoquer guix gc], page 60).

La commande `guix download` supporte les mêmes URI que celles utilisées dans les définitions de paquets. En particulier, elle supporte les URI `mirror://`. Les URI `http` (HTTP sur TLS) sont supportées *si* les liaisons Guile de GnuTLS sont disponibles dans l'environnement de l'utilisateur ; si elle ne sont pas disponibles, une erreur est renvoyée. Voir Section "Guile Preparations" dans *GnuTLS-Guile*, pour plus d'informations.

`guix download` vérifie les certificats du serveur HTTPS en chargeant les autorités de certification X.509 depuis le répertoire vers lequel pointe la variable d'environnement `SSL_CERT_DIR` (voir Section 12.11 [Certificats X.509], page 572), à moins que `--no-check-certificate` ne soit utilisé.

Les options suivantes sont disponibles :

`--hash=algorithme`

`-H algorithme`

Calcule un hash en utilisant l'*algorithme* spécifié. Voir Section 10.4 [Invoquer guix hash], page 196, pour plus d'informations.

`--format=fmt`

`-f fmt` Écrit le hash dans le format spécifié par *fmt*. Pour plus d'informations sur les valeurs valides pour *fmt*, voir Section 10.4 [Invoquer guix hash], page 196.

`--no-check-certificate`

Ne pas valider les certificats HTTPS des serveurs.

Lorsque vous utilisez cette option, vous n'avez *absolument aucune garanti* que vous communiquez avec le serveur authentique responsable de l'URL donnée, ce qui vous rend vulnérable à des attaques de « l'homme du milieu ».

`--output=fichier`

`-o fichier`

Enregistre le fichier téléchargé dans *fichier* plutôt que de l'ajouter au dépôt.

## 10.4 Invoquer guix hash

La commande `guix hash` calcule le hash d'un fichier. C'est surtout un outil pour simplifier la vie des contributeur-riche-s à la distribution : elle calcule le hash cryptographique d'un ou plusieurs fichiers, qui peut être utilisé dans la définition d'un paquet (voir Section 9.2 [Définition des paquets], page 105).

La syntaxe générale est :

```
guix hash option fichier ...
```

Lorsque *fichier* est - (un tiret), `guix hash` calcul le hash des données lues depuis l'entrée standard. `guix hash` a les options suivantes :

`--hash=algorithme`

`-H algorithme`

Calcule un hash en utilisant l' *algorithme* spécifié, `sha256` par défaut.

*algorithme* doit être le nom de l' algorithme d'un hash cryptographique supporté par Libgcrypt *via* Guile-Gcrypt—par exemple, `sha512` ou `sha3-256` (voir Section “Hash Functions” dans *Guile-Gcrypt Reference Manual*).

`--format=fmt`

`-f fmt` Écrit le hash dans le format spécifié par *fmt*.

Formats supportés : `base64`, `nix-base32`, `base32`, `base16` (hex et hexadecimal peuvent être utilisés).

Si l'option `--format` n'est pas spécifiée, `guix hash` affichera le hash en `nix-base32`. Cette représentation est utilisée dans les définitions des paquets.

`--recursive`

`-r` L'option `--recursive` est obsolète et remplacée par `--serializer=nar` (voir plus bas) ; `-r` reste accepté car c'est un raccourci pratique.

`--serializer=type`

`-S type` Calcule le hash sur *fichier* avec la sérialisation *type*.

*type* peut être l'une des valeurs suivantes :

`none` C'est la valeur par défaut : elle calcule le hash du contenu d'un fichier.

`nar` Calcul le hash d'une « archive normalisée » (ou « nar ») contenant *fichier*, dont ses enfants si c'est un répertoire. Certaines métadonnées de *fichier* fait partie de l'archive ; par exemple lorsque *fichier* est un fichier normal, le hash est différent que le *fichier* soit exécutable ou non. Les métadonnées comme un horodatage n'ont aucun impact sur le hash (voir Section 6.10 [Invoquer guix archive], page 71, pour plus de détails sur le format nar).

`git` Calcule le hash d'un fichier ou d'un répertoire comme une arborescence Git, suivant la même méthode que le système de contrôle de version Git.

`--exclude-vcs`

`-x` Conjointement avec `--recursive`, exclut les répertoires de système de contrôle de version (`.bzd`, `.git`, `.hg`, etc.).

Par exemple, voici comment calculer le hash d'un dépôt Git, ce qui est utile avec la méthode `git-fetch` (voir Section 9.2.2 [référence de origin], page 113) :

```
$ git clone http://example.org/toto.git
$ cd toto
$ guix hash -x --serializer=nar .
```

## 10.5 Invoquer `guix import`

La commande `guix import` est utile pour les gens qui voudraient ajouter un paquet à la distribution avec aussi peu de travail que possible — une demande légitime. La commande connaît quelques dépôts logiciels d'où elle peut « importer » des métadonnées de paquets. Le résultat est une définition de paquet, ou un modèle de définition, dans le format reconnu par Guix (voir Section 9.2 [Définition des paquets], page 105).

La syntaxe générale est :

```
guix import importer options...
```

`importer` spécifie la source depuis laquelle importer des métadonnées de paquets, et `options` spécifie un identifiant de paquet et d'autres options spécifiques à `importer`.

Certains des importeurs s'appuient sur la capacité d'exécuter la commande `gpgv`. Pour ceux-ci, GnuPG doit être installé dans `$PATH` ; exécuter `guix install gnupg` si nécessaire.

Actuellement, les « importeurs » disponibles sont :

`gnu` Importe des métadonnées d'un paquet GNU donné. Cela fournit un modèle pour la dernière version de ce paquet GNU, avec le hash de son archive, le synopsis et la description canonique.

Les informations supplémentaires comme les dépendances du paquet et sa licence doivent être renseignées manuellement.

Par exemple, la commande suivante renvoie une définition de paquets pour GNU Hello :

```
guix import gnu hello
```

Les options spécifiques sont :

```
--key-download=politique
```

Comme pour `guix refresh`, spécifie la politique de gestion des clefs OpenPGP manquantes lors de la vérification de la signature d'un paquet. Voir Section 10.6 [Invoquer `guix refresh`], page 205.

`pypi` Importe des métadonnées depuis l'index des paquets Python (<https://pypi.python.org/>). Les informations sont récupérées à partir de la description en JSON disponible sur [pypi.python.org](https://pypi.python.org) et inclus généralement toutes les informations utiles, dont les dépendances des paquets. Pour une efficacité maximale, il est recommandé d'installer l'utilitaire `unzip`, pour que l'importateur puisse dézipper les wheels Python et récupérer les informations contenues à l'intérieur.



La commande ci-dessous importe les métadonnées de la dernière version du paquet Python `itsdangerous` :

```
guix import pypi itsdangerous
```

Vous pouvez aussi demander une version spécifique :

```
guix import pypi itsdangerous@1.1.0
```

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

`gem` Importe des métadonnées de RubyGems (<https://rubygems.org/>). Les informations sont récupérées au format JSON disponible sur [rubygems.org](https://rubygems.org) et inclut les informations les plus utiles, comme les dépendances à l'exécution. Il y a des cependant quelques restrictions. Les métadonnées ne distinguent pas synopsis et description, donc la même chaîne est utilisée pour les deux champs. En plus, les détails des dépendances non Ruby requises pour construire des extensions natives sont indisponibles et laissé en exercice à l'empaqueteur.

La commande ci-dessous importe les métadonnées pour le paquet Ruby `rails` :

```
guix import gem rails
```

Vous pouvez aussi demander une version spécifique :

```
guix import gem rails@7.0.4
```

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

`minetest` Importe des métadonnées de ContentDB (<https://content.minetest.net>). Les informations sont récupérées au format JSON disponible à travers l'API de ContentDB (<https://content.minetest.net/help/api/>) et inclus les informations les plus utiles, dont les dépendances. Il y a cependant quelques limitations. L'information de licence est souvent incomplète. Le hash de commit est parfois oublié. Les descriptions sont au format Markdown mais Guix utilise plutôt Texinfo. Les packs de textures et les jeux ne sont pas pris en charge.

La commande ci-dessous importe les métadonnées pour le mod Mesecons de Jeija :

```
guix import minetest Jeija/mesecons
```

Vous pouvez aussi ne pas indiquer le nom de l'auteur :

```
guix import minetest mesecons
```

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

**cpan** Importe des métadonnées de MetaCPAN (<https://www.metacpan.org/>). Les informations sont récupérées au format JSON disponible à travers l'API de MetaCPAN (<https://fastapi.metacpan.org/>) et inclus les informations les plus utiles, comme les dépendances des modules. L'information sur les licences doit être vérifiée avec attention. Si Perl est disponible dans le dépôt, alors l'utilitaire `corelist` sera utiliser pour exclure les modules du cœur de la distribution Perl de la liste des dépendances.

La commande ci-dessous importe des métadonnées pour le module `Acme::Boolean` Perl :

```
guix import cpan Acme::Boolean
```

**cran** Importe des métadonnées de CRAN (<https://cran.r-project.org/>), le dépôt central de l'environnement statistique et graphique GNU R (<https://r-project.org/>).

L'information est extraite du fichier `DESCRIPTION` du paquet.

La commande ci-dessous importe les métadonnées du paquet Cairo R :

```
guix import cran Cairo
```

Vous pouvez aussi demander une version spécifique :

```
guix import cran rasterVis@0.50.3
```

Lorsque l'option `--recursive` est utilisée, l'importateur traversera le graphe des dépendances du paquet en amont récursivement et générera des expressions de paquets pour tous ceux qui ne sont pas déjà dans Guix.

Lorsque vous ajoutez `--style=specification`, l'importateur générera des définitions de paquets dont les entrées sont les spécifications des paquets au lieu de références au variables des paquets. C'est utile lorsque des définitions de paquets doivent être ajoutées à des modules utilisateurs, comme la liste des modules de paquets n'a pas besoin d'être changée. La valeur par défaut est `--style=variable`.

Lorsque `--archive=bioconductor` est ajoutée, les métadonnées sont importées depuis Bioconductor (<https://www.bioconductor.org/>), un répertoire de paquets R pour l'analyse et la compréhension de données génomiques volumineuses en bioinformatique.

Les informations sont extraites du fichier `DESCRIPTION` du paquet contenu dans l'archive du paquet.

La commande ci-dessous importe les métadonnées pour le paquet R `GenomicRanges` :

```
guix import cran --archive=bioconductor GenomicRanges
```

Enfin, vous pouvez aussi importer des paquets R qui n'ont pas encore été publiés sur CRAN ou Bioconductor tant qu'ils ne sont pas dans le dépôt Git. Utilisez `--archive=git` suivi par l'URL du dépôt Git :

```
guix import cran --archive=git https://github.com/immunogenomics/harmony
```

**texlive** Importe les informations de paquets TeX à partir de la base de données TeX Live pour les paquets TeX qui font partie de la distribution TeX Live (<https://www.tug.org/texlive/>).

Les informations sur les paquets sont obtenues à partir de la base de données des paquets de TeX Live, un fichier texte brut inclus dans le paquets `texlive-bin`. Le code source est téléchargé depuis plusieurs emplacements dans le dépôt SVN du projet TeX Live.

La commande ci-dessous importe les métadonnées du paquet TeX `fontspec` :

```
guix import texlive fontspec
```

`json` Importe des métadonnées d'un fichier JSON local. Considérez l'exemple suivant d'une définition de paquet au format JSON :

```
{
  "name": "hello",
  "version": "2.10",
  "source": "mirror://gnu/hello/hello-2.10.tar.gz",
  "build-system": "gnu",
  "home-page": "https://www.gnu.org/software/hello/",
  "synopsis": "Hello, GNU world: An example GNU package",
  "description": "GNU Hello prints a greeting.",
  "license": "GPL-3.0+",
  "native-inputs": ["gettext"]
}
```

Les noms des champs sont les mêmes que pour les enregistrements de `<package>` (Voir Section 9.2 [Définition des paquets], page 105). Les référence à d'autres paquets sont fournies comme des listes JSON de chaînes de spécifications de paquets comme `guile` ou `guile@2.0`.

L'importateur supporte aussi une définition plus explicite des sources avec les champs habituels pour les enregistrements `<origin>` :

```
{
  ...
  "source": {
    "method": "url-fetch",
    "uri": "mirror://gnu/hello/hello-2.10.tar.gz",
    "sha256": {
      "base32": "0ssi1wpaf7plawqqjwigppsg5fyh99vdlb9kzl7c9lng89ndqi"
    }
  }
  ...
}
```

La commande ci-dessous lit les métadonnées du fichier JSON `hello.json` et renvoie une expression de paquet :

```
guix import json hello.json
```

`hackage` Importe les métadonnées de l'archive de paquets centrale de la communauté Haskell, Hackage (<https://hackage.haskell.org/>). Les informations sont récupérées depuis les fichiers Cabal et incluent toutes les informations utiles, dont les dépendances des paquets.

Les options spécifiques sont :

```

--stdin
-s          Lit un fichier Cabal depuis l'entrée standard.

--no-test-dependencies
-t          N'inclut pas les dépendances requises uniquement par les suites de
           tests.

--cabal-environment=alist
-e alist  alist est une alist Scheme qui définit l'environnement dans lequel
           les conditions de Cabal sont évaluées. Les clefs acceptées sont :
           os, arch, impl et une représentation sous forme de chaîne de car-
           actères du nom d'un drapeau. La valeur associée à un drapeau doit
           être le symbole true ou false. La valeur associée aux autres clefs
           doivent se conformer avec la définition du format de fichiers Cabal.
           La valeur par défaut associée avec les clefs os, arch et impl sont
           respectivement 'linux', 'x86_64' et 'ghc'.

--recursive
-r          Traverse le graphe des dépendances du paquet amont donné et
           génère les expressions de paquets de tous ceux qui ne sont pas déjà
           dans Guix.

```

La commande ci-dessous importe les métadonnées pour la dernière version du paquet HTTP Haskell sans inclure les dépendances de test et en spécifiant la valeur du drapeau 'network-uri' comme **false** :

```
guix import hackage -t -e "'(\\\"network-uri\\\" . false))" HTTP
```

Une version spécifique du paquet peut éventuellement être spécifiée en faisant suivre le nom du paquet par un arobase et un numéro de version comme dans l'exemple suivant :

```
guix import hackage mtl@2.1.3.1
```

**stackage** L'importateur **stackage** est une enveloppe autour de l'importateur **hackage**. Il prend un nom de paquet, recherche la version incluse dans une version au support étendu (LTS) de Stackage (<https://www.stackage.org>) et utilise l'importateur **hackage** pour récupérer les métadonnées. Remarquez que c'est à vous de choisir une version LTS compatible avec le compilateur GHC utilisé par Guix.

Les options spécifiques sont :

```

--no-test-dependencies
-t          N'inclut pas les dépendances requises uniquement par les suites de
           tests.

--lts-version=version
-l version
           version est la version LTS désirée. Si elle est omise, la dernière
           version est utilisée.

```

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

La commande ci-dessous importe les métadonnées pour le paquet HTTP Haskell inclus dans la version 7.18 de LTS Stackage :

```
guix import stackage --lts-version=7.18 HTTP
```

`elpa` Importe les métadonnées du dépôt de paquets ELPA (Emacs Lisp Package Archive) (voir Section “Packages” dans *The GNU Emacs Manual*).

Les options spécifiques sont :

`--archive=repo`

`-a repo` *repo* identifie le dépôt d’archive depuis lequel récupérer les informations. Actuellement les dépôts supportés et leurs identifiants sont :

- GNU (<https://elpa.gnu.org/packages>), qu’on peut choisir avec l’identifiant `gnu`. C’est la valeur par défaut.

Les paquets de `elpa.gnu.org` avec l’une des clefs contenues dans le porte-clef GnuPG `share/emacs/25.1/etc/package-keyring.gpg` (ou similaire) dans le paquet `emacs` (voir Section “Package Installation” dans *The GNU Emacs Manual*).

- NonGNU (<https://elpa.nongnu.org/nongnu/>), qu’on peut choisir avec l’identifiant `nongnu`.

- MELPA-Stable (<https://stable.melpa.org/packages>), qu’on peut sélectionner avec l’identifiant `melpa-stable`.

- MELPA (<https://melpa.org/packages>), qu’on peut sélectionner avec l’identifiant `melpa`.

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

`crate` Importer les métadonnées du dépôt de paquets Rust crates.io (<https://crates.io>), comme dans cet exemple :

```
guix import crate blake2-rfc
```

L’importeur `crate` vous permet aussi de spécifier une version de chaînes de caractères :

```
guix import crate constant-time-eq@0.1.0
```

Les options supplémentaires comprennent :

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

**elm** Importer les métadonnées du dépôt de paquets Elm `package.elm-lang.org` (<https://package.elm-lang.org>), comme dans cet exemple :

```
guix import elm elm-explorations/webgl
```

L'importeur Elm vous permet aussi de spécifier une version de chaînes de caractères :

```
guix import elm elm-explorations/webgl@1.1.3
```

Les options supplémentaires comprennent :

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

**opam** Importe les métadonnées du répertoire de paquets OPAM (<https://opam.ocaml.org/>) utilisé par la communauté OCaml.

Les options supplémentaires comprennent :

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

`--repo` Par défaut on recherche les paquets dans le dépôt OPAM officiel. Cette option, qui peut être utilisée plusieurs fois en même temps, vous permet d'ajouter d'autres dépôts dans lesquelles on recherchera les paquets. Les arguments valide sont :

- le nom d'un dépôt connu — cela peut être `opam`, `coq` (équivalent à `coq-released`), `coq-core-dev`, `coq-extra-dev` ou `grew`.
- l'URL d'un dépôt, telle qu'attendue par la commande `opam repository add` (par exemple, l'URL équivalent au nom `opam` plus haut serait <https://opam.ocaml.org>).
- le chemin vers une copie locale d'un dépôt (un répertoire contenant un sous-répertoire `packages/`).

Les dépôts sont passés à cette commande par ordre de préférence. Les dépôts supplémentaires ne remplaceront pas le dépôt `opam` par défaut, qui est toujours gardé en dernier recours.

Ainsi, remarquez que les versions ne sont pas comparées entre dépôts. Le premier dépôt (de gauche à droite) qui a au moins une version d'un paquet donné prendra le pas sur les autres, et la version importée sera la plus récente trouvée *dans ce dépôt uniquement*.

**go** Importe les métadonnées d'un module Go avec `proxy.golang.org` (<https://proxy.golang.org>).

```
guix import go gopkg.in/yaml.v2
```

Il est possible d'utiliser la spécification d'un paquet avec le suffixe `@VERSION` pour importer une version spécifique.

Les options supplémentaires comprennent :

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

`--pin-versions`

Lorsque vous utilisez cette option, l'importateur préserve les versions exactes des modules Go de dépendance au lieu d'utiliser leur dernière version disponible. C'est utile si vous essayez d'importer un paquet qui dépend récursivement de versions antérieures de lui-même pour la construction. Lorsque vous utilisez ce mode, le symbole du paquet sera créé en ajoutant la version à son nom, pour que plusieurs version du même paquet puissent coexister.

`egg` Importe les métadonnées pour des « eggs » CHICKEN (<https://wiki.call-cc.org/eggs>). L'information est récupérée dans les fichiers `PACKAGE.egg` trouvés dans le dépôt git `eggs-5-all` (<git://code.call-cc.org/eggs-5-all>). Toutefois, il ne fournit pas toutes les informations dont vous avez besoin, il n'y a pas de champ “description” et les licences utilisées ne sont pas toujours précises (BSD est souvent employée à la place de BSD-N).

```
guix import egg sourcehut
```

Vous pouvez aussi demander une version spécifique :

```
guix import egg arrays@1.0
```

Les options supplémentaires comprennent :

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

`hexpm` Importer les métadonnées du dépôt de paquets Erlang et Elixir `hex.pm` `hex.pm` (<https://hex.pm>), comme dans cet exemple :

```
guix import hexpm stun
```

L'importateur essaye de déterminer le système de construction utilisé par le paquet.

L'importateur `hexpm` vous permet aussi de spécifier une version de chaînes de caractères :

```
guix import hexpm cf@0.3.0
```

Les options supplémentaires comprennent :

`--recursive`

`-r` Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.

La structure du code de `guix import` est modulaire. Il serait utile d'avoir plus d'importateurs pour d'autres formats de paquets et votre aide est la bienvenue sur ce sujet (voir Chapitre 22 [Contribuer], page 668).

## 10.6 Invoquer `guix refresh`

La commande `guix refresh` s'adresse avant tout aux personnes qui écrivent des paquets. En tant qu'utilisateur·rice, vous pourriez être intéressé·e par l'option `--with-latest` qui peut vous conférer les superpouvoirs de mettre à jour les paquets, et qui est construite à partir de `guix refresh` (voir Section 10.1.2 [Options de transformation de paquets], page 183). Par défaut, `guix refresh` rapporte les paquets fournis par la distribution qui sont en retard par rapport aux dernières versions disponibles en amont, comme ceci :

```
$ guix refresh
gnu/packages/gettext.scm:29:13: gettext serait mis à jour de 0.18.1.1 à 0.18.2.1■
gnu/packages/glib.scm:77:12: glib serait mis à jour de 2.34.3 à 2.37.0
```

On peut aussi spécifier des paquets à prendre en compte, auquel cas un avertissement est émis pour les paquets qui ne sont pas mis à jour :

```
$ guix refresh coreutils guile guile-ssh
gnu/packages/ssh.scm:205:2 : avertissement : aucun gestionnaire de mise à jour pour gu
gnu/packages/guile.scm:136:12 : guile serait mis à jour de 2.0.12 à 2.0.13■
```

`guix refresh` navigue le dépôt amont de chaque paquet et détermine le numéro de version le plus élevé parmi les versions publiées. La commande sait comment mettre à jour certains types de paquets : les paquets GNU, les paquets ELPA, etc. — voir la documentation pour `--type` ci-dessous. Il y a beaucoup de paquet cependant pour lesquels il manque une méthode pour déterminer si une nouvelle version est disponible en amont. Cependant, le mécanisme est extensible, alors n'hésitez pas à nous contacter pour ajouter une nouvelle méthode !

`--recursive`

Considère les paquets spécifiés et tous les paquets dont ils dépendent.

```
$ 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
...
```

Parfois les noms en amont diffèrent du nom de paquet utilisé par Guix et `guix refresh` a besoin d'un peu d'aide. La plupart des gestionnaires de mise à jour prennent en compte la propriété `upstream-name` dans les définitions de paquets, ce qui peut être utilisé à cette fin :

```
(define-public network-manager
  (package
    (name "network-manager")
    ;; ...
    (properties '((upstream-name . "NetworkManager")))))
```



Quand l'option `--update` est utilisée, elle modifie les fichiers sources de la distribution pour mettre à jour les numéros de version et les hash de l'archive source de ces recettes de paquets (voir Section 9.2 [Définition des paquets], page 105). Pour ce faire, il télécharge la dernière archive source de chaque paquet et sa signature OpenPGP associée, authentifiant l'archive téléchargée par rapport à sa signature en utilisant `gpgv`, et enfin calcule son hash—notez que `GnuPG` doit être installé et dans `$PATH` ; lancez `guix install gnupg` si nécessaire.

Quand la clé publique utilisée pour signer l'archive est manquante depuis le trousseau de clés utilisateur-`rice`, une tentative est faite de la récupérer automatiquement depuis un serveur de clé publique ; en cas de succès, la clé est ajoutée au trousseau de l'utilisateur-`rice` ; sinon, `guix refresh` renvoie une erreur.

Les options suivantes sont supportées :

`--expression=expr`

`-e expr` Considérer le paquet évalué par `expr`.

C'est utile pour précisément se référer à un paquet, comme dans cet exemple :

```
guix refresh -l -e '(@@ (gnu packages commencement) glibc-final)'
```

Cette commande liste les dépendances de la libc « finale » (presque tous les paquets).

`--update`

`-u` Met à jour les fichiers source de la distribution (les recettes de paquets) en place. Cette option est généralement utilisée depuis une copie du dépôt git de Guix (voir Section 22.2 [Lancer Guix avant qu'il ne soit installé], page 670) :

```
$ ./pre-inst-env guix refresh -s non-core -u
```

Voir Section 9.2 [Définition des paquets], page 105, pour plus d'information sur les définitions des paquets.

`--select=[subset]`

`-s subset` Choisi tous les paquets dans `subset`, entre `core` et `non-core`.

Le sous-ensemble `core` se réfère à tous les paquets du cœur de la distribution — c.-à-d. les paquets qui sont utilisés pour construire « tout le reste ». Cela comprend `GCC`, `libc`, `Binutils`, `Bash`, etc. Habituellement, changer l'un de ces paquets dans la distribution implique de reconstruire tous les autres. Ainsi, ces mises à jour sont une nuisance pour les utilisateurs, en terme de temps de compilation et de bande passante utilisés pour effectuer la mise à jour.

Le sous-ensemble `non-core` se réfère au reste des paquets. C'est habituellement utile dans les cas où une mise à jour des paquets du cœur serait dérangeante.

`--manifest=fichier`

`-m fichier`

Choisi tous les paquets du manifeste dans `file`. C'est utile pour vérifier qu'aucun des paquets du manifeste utilisateur ne peut être mis à jour.

`--type=updater`

`-t updater`

Chois uniquement les paquets pris en charge par `updater` (éventuellement une liste de gestionnaires de mise à jour séparés par des virgules). Actuellement, `updater` peut être l'une des valeurs suivantes :

<code>gnu</code>	le gestionnaire de mise à jour pour les paquets GNU ;
<code>savannah</code>	le gestionnaire de mise à jour pour les paquets hébergés sur Savannah ( <a href="https://savannah.gnu.org">https://savannah.gnu.org</a> ) ;
<code>sourceforge</code>	le gestionnaire de mise à jour pour les paquets hébergés sur SourceForge ( <a href="https://sourceforge.net">https://sourceforge.net</a> ) ;
<code>gnome</code>	le gestionnaire de mise à jour pour les paquets GNOME ;
<code>kde</code>	le gestionnaire de mise à jour pour les paquets KDE ;
<code>xorg</code>	le gestionnaire de mise à jour pour les paquets X.org ;
<code>kernel.org</code>	le gestionnaire de mise à jour pour les paquets hébergés sur kernel.org ;
<code>egg</code>	le gestionnaire de mise à jour pour les paquets Egg ( <a href="https://wiki.call-cc.org/eggs/">https://wiki.call-cc.org/eggs/</a> ) ;
<code>elpa</code>	le gestionnaire de mise à jour pour les paquets ELPA ( <a href="https://elpa.gnu.org/">https://elpa.gnu.org/</a> ) ;
<code>cran</code>	le gestionnaire de mise à jour pour les paquets CRAN ( <a href="https://cran.r-project.org/">https://cran.r-project.org/</a> ) ;
<code>bioconductor</code>	le gestionnaire de mise à jour pour les paquets Bioconductor ( <a href="https://www.bioconductor.org/">https://www.bioconductor.org/</a> ) ;
<code>cpan</code>	le gestionnaire de mise à jour pour les paquets CPAN ( <a href="https://www.cpan.org/">https://www.cpan.org/</a> ) ;
<code>pypi</code>	le gestionnaire de mise à jour pour les paquets PyPI ( <a href="https://pypi.python.org">https://pypi.python.org</a> ).
<code>gem</code>	le gestionnaire de mise à jour pour les paquets RubyGems ( <a href="https://rubygems.org">https://rubygems.org</a> ).
<code>github</code>	le gestionnaire de mise à jour pour les paquets GitHub ( <a href="https://github.com">https://github.com</a> ).
<code>hackage</code>	le gestionnaire de mise à jour pour les paquets Hackage ( <a href="https://hackage.haskell.org">https://hackage.haskell.org</a> ).
<code>stackage</code>	le gestionnaire de mise à jour pour les paquets Stackage ( <a href="https://www.stackage.org">https://www.stackage.org</a> ).
<code>crate</code>	le gestionnaire de mise à jour pour les paquets Crates ( <a href="https://crates.io">https://crates.io</a> ).
<code>launchpad</code>	le gestionnaire de mise à jour pour les paquets Launchpad ( <a href="https://launchpad.net">https://launchpad.net</a> ).

**generic-html**

un gestionnaire de mise à jour qui visite la page HTML où l'archive des sources est hébergée, lorsque c'est possible.

**generic-git**

un gestionnaire de mise à jour générique pour les paquets hébergés sur des dépôts Git. Il essaye d'analyser les noms de tag Git de manière astucieuse, mais s'il ne parvient pas à analyser le nom d'un tag et à comparer les tags correctement, il est possible de définir les propriétés suivantes pour un paquet.

- **release-tag-prefix** : une expression régulière pour repérer le préfixe dans le nom d'un tag.
- **release-tag-suffix** : une expression régulière pour repérer le suffixe dans le nom d'un tag.
- **release-tag-version-delimiter** : une chaîne de caractères utilisée comme délimitation dans le nom d'un tag pour séparer les numéros de version.
- **accept-pre-releases** : par défaut, le gestionnaire de mise à jour ignorera les pré-versions ; pour lui faire chercher aussi les pré-versions, assignez à cette propriété la valeur **#t**.

```
(package
  (name "toto")
  ;; ...
  (properties
    '((release-tag-prefix . "^release0-")
      (release-tag-suffix . "[a-z]?$")
      (release-tag-version-delimiter . ":"))))
```

Par exemple, la commande suivante ne vérifie que les mises à jour des paquets Emacs hébergés sur [elpa.gnu.org](http://elpa.gnu.org) et les paquets CRAN :

```
$ guix refresh --type=elpa,cran
gnu/packages/statistics.scm:819:13 : r-testthat serait mis à jour de 0.10.0
gnu/packages/emacs.scm:856:13 : emacs-auctex serait mis à jour de 11.88.6 à
```

**--list-updaters**

Liste les gestionnaires de mises à jour disponibles et quitte (voir **--type** ci-dessus).

Pour chaque gestionnaire, affiche le pourcentage de paquets qu'il couvre ; à la fin, affiche le pourcentage de paquets couverts par tous les gestionnaires.

En plus, on peut passer à **guix refresh** un ou plusieurs noms de paquets, comme dans cet exemple :

```
$ ./pre-inst-env guix refresh -u emacs idutils gcc@4.8
```

La commande ci-dessus met à jour spécifiquement les paquets **emacs** et **idutils**. L'option **--select** n'aurait aucun effet dans ce cas. Vous voudrez aussi sans doute mettre à jour les définitions qui correspondent aux paquets installés dans votre profil :

```
$ ./pre-inst-env guix refresh -u \
```

```
$ (guix package --list-installed | cut -f1)
```

Pour déterminer s'il faut mettre à jour un paquet, il est parfois pratique de savoir quels paquets seraient affectés par la mise à jour pour pouvoir vérifier la compatibilité. Pour cela l'option suivante peut être utilisée avec un ou plusieurs noms de paquets passés à `guix refresh` :

`--list-dependent`

-1 Liste les paquets de plus haut-niveau qui devraient être reconstruits après la mise à jour d'un ou plusieurs paquets.

Voir Section 10.10 [Invoquer `guix graph`], page 218, pour des informations sur la manière de visualiser la liste des paquets dépendant d'un autre.

Sachez que l'option `--list-dependent` n'a qu'une approche *approximative* sur les reconstructions qui seraient nécessaires à la suite d'une mise à jour. D'autres reconstructions peuvent être nécessaires dans certaines circonstances.

```
$ guix refresh --list-dependent flex
```

```
Construire les 120 paquets suivants s'assure que les 213 paquets dépendants sont recon  
hop@2.4.0 emacs-geiser@0.13 notmuch@0.18 mu@0.9.9.5 cflow@1.4 idutils@4.6 ...■
```

La commande ci-dessus liste un ensemble de paquets qui peuvent être construits pour vérifier la compatibilité d'une mise à jour de `flex`.

`--list-transitive`

Lister tous les paquets dont un paquet ou plus dépendent.

```
$ guix refresh --list-transitive flex
```

```
flex@2.6.4 depends on the following 25 packages: perl@5.28.0 help2man@1.47.6  
bison@3.0.5 indent@2.2.10 tar@1.30 gzip@1.9 bzip2@1.0.6 xz@5.2.4 file@5.33 .
```

La commande ci-dessus liste un ensemble de paquets qui, lorsqu'ils sont modifiés, causent la reconstruction de `flex`.

Les options suivante peuvent être utilisées pour personnaliser les opérations avec GnuPG :

`--gpg=commande`

Utilise *commande* comme la commande de GnuPG 2.x. *commande* est recherchée dans `PATH`.

`--keyring=fichier`

Utilise *fichier* comme porte-clefs pour les clefs amont. *fichier* doit être dans le *format keybox*. Les fichiers Keybox ont d'habitude un nom qui fini par `.kbx` et GNU Privacy Guard (GPG) peut manipuler ces fichiers (voir Section "kbxutil" dans *Using the Privacy Guard*, pour plus d'informations sur un outil pour manipuler des fichiers keybox).

Lorsque cette option est omise, `guix refresh` utilise `~/.config/guix/upstream/trustedkeys.kbx` comme trousseau pour les clés de signature en amont. Les signatures OpenPGP sont vérifiées avec ces clés ; les clés manquantes sont aussi téléchargées dans ce trousseau (voir `--key-download` plus bas).

Vous pouvez exporter les clefs de votre porte-clefs GPG par défaut dans un fichier keybox avec une commande telle que :

```
gpg --export rms@gnu.org | kbxutil --import-openpgp >> mykeyring.kbx■
```

De même, vous pouvez récupérer des clefs dans un fichier keybox spécifique comme ceci :

```
gpg --no-default-keyring --keyring mykeyring.kbx \
  --recv-keys 3CE464558A84FDC69DB40CFB090B11993D9AEBB5
```

Voir Section “GPG Configuration Options” dans *Using the GNU Privacy Guard* pour plus d’informations sur l’option `--keyring` de GPG.

#### `--key-download=politique`

Gère les clefs OpenPGP manquantes d’après la *politique*, qui peut être l’une des suivantes :

**always** Toujours télécharger les clefs manquantes depuis un serveur de clefs et les ajouter au porte-clefs de l’utilisateur.

**never** Ne jamais essayer de télécharger les clefs OpenPGP manquante. Quitter à la place.

#### **interactive**

Lorsqu’on rencontre un paquet signé par une clef OpenPGP inconnue, demander à l’utilisateur s’il souhaite la télécharger ou non. C’est le comportement par défaut.

#### `--key-server=host`

Utiliser *host* comme serveur de clefs OpenPGP lors de l’importe d’une clef publique.

#### `--load-path=répertoire`

#### `-L répertoire`

Ajoute *répertoire* au début du chemin de recherche de module de paquets (voir Section 9.1 [Modules de paquets], page 104).

Cela permet à des utilisateurs de définir leur propres paquets et les rendre disponibles aux outils en ligne de commande.

Le gestionnaire de mises à jour `github` utilise GitHub API (<https://developer.github.com/v3/>) pour faire des requêtes sur les nouvelles versions. Lorsqu’elle est utilisé de manière répétée, par exemple lorsque vous vérifiez tous les paquets, GitHub finira par refuser de répondre à d’autres requêtes de l’API. Par défaut 60 requêtes à l’heure sont autorisées, et une vérification complète de tous les paquets GitHub dans Guix demande bien plus que cela. L’authentification avec GitHub à travers l’utilisation d’un jeton d’API lève ces limites. Pour utiliser un jeton de l’API, initialisez la variable d’environnement `GUIX_GITHUB_TOKEN` avec un jeton que vous vous serez procuré sur <https://github.com/settings/tokens> ou autrement.

## 10.7 Invoquer `guix style`

La commande `guix style` aide aussi bien les utilisateur·ices que les auteur·ices de paquets à présenter leurs définitions de paquets et leurs fichiers de configuration suivant la toute dernière mode. Elle peut aussi bien reformater des fichiers entiers, avec l’option `--whole-file`, qu’appliquer des *règles de style* spécifiques à des définitions de paquets individuelles. La commande fournit actuellement les règles stylistiques suivantes :

- formater les définitions de paquets suivant les conventions du projet (voir Section 22.5.4 [Formatage du code], page 683) ;
- réécrire les entrées d'un paquet dans le « nouveau style » expliqué plus bas.

La manière dont les entrées des paquets sont écrites est en phase de transition (voir Section 9.2.1 [référence de package], page 108, pour plus d'information sur les entrées des paquets). Jusqu'à la version 1.3.0, les entrées des paquets étaient écrites avec l'« ancien style », où chaque entrée était associée à une étiquette explicite, la plupart du temps le nom du paquet :

```
(package
  ;; ...
  ;; L'« ancien style » (obsolète).
  (inputs `(("libunistring" ,libunistring)
            ("libffi" ,libffi))))
```

Aujourd'hui, l'ancien style est obsolète et le style de préférence ressemble à ceci :

```
(package
  ;; ...
  ;; Le « nouveau style ».
  (inputs (list libunistring libffi)))
```

De la même manière, l'utilisation de `alist-delete` et compagnie pour manipuler les entrées est maintenant obsolète en faveur de `modify-inputs` (voir Section 9.3 [Définition de variantes de paquets], page 116, pour plus d'information sur `modify-inputs`).

Dans la grande majorité des cas, c'est un changement purement mécanique de la syntaxe qui ne provoque même pas de reconstruction du paquet. Lancer `guix style -S inputs` peut le faire pour vous, que vous travailliez sur des paquets dans Guix ou dans un canal externe.

La syntaxe générale est :

```
guix style [options] paquet...
```

Cela fait analyser et réécrire la définition des *paquet...* à `guix style` ou, si *paquet* est omis, de *tous* les paquets. Les options `--styling` ou `-S` vous permettent de choisir la règle de style, la règle par défaut est `format` — voir plus bas.

Pour reformater des fichiers sources complets, la syntaxe est :

```
guix style --whole-file fichier...
```

Les options disponibles sont listées ci-dessous.

`--dry-run`

`-n` Montre les emplacement dans les fichiers sources qui seraient modifiés mais sans les modifier.

`--whole-file`

`-f` Reformate entièrement les fichiers donnés. Dans ce cas, les arguments suivants sont interprétés comme des noms de fichiers (plutôt que des noms de paquets), et l'option `--styling` n'a aucun effet.

Par exemple, voici comme vous pouvez reformater votre configuration de système d'exploitation (vous devez avoir les permissions en écriture sur le fichier) :

```
guix style -f /etc/config.scm
```

**--styling=règle**

**-S règle** Applique la *règle*, l'une des règles de style suivantes :

**format** Formate les définitions de paquets données — c'est la règle de style par défaut. Par exemple, la personne qui crée un paquet et qui utilise le Guix du checkout (voir Section 22.2 [Lancer Guix avant qu'il ne soit installé], page 670) peut vouloir reformater la définition du paquet Coreutils de cette manière :

```
./pre-inst-env guix style coreutils
```

**inputs** Réécrit les entrées du paquet en le « nouveau style », décrit plus haut. Vous pouvez réécrire les entrées d'un paquet *toto* dans votre propre canal de cette manière :

```
guix style -L ~/mon/canal -S inputs toto
```

La réécriture s'effectue de manière prudente : les commentaires sont préservés et le processus s'arrête s'il n'arrive pas à comprendre le code du champ *inputs*. L'option **--input-simplification** décrite plus bas fournit un contrôle plus fin sur les entrées qui devraient être simplifiées.

**--list-stylings**

**-l** Liste et décrit les règles de style puis quitte.

**--load-path=répertoire**

**-L répertoire**

Ajoute *répertoire* au début du chemin de recherche de module de paquets (voir Section 9.1 [Modules de paquets], page 104).

**--expression=expr**

**-e expr** Reformate le paquet évalué par *expr*.

Par exemple, lancer :

```
guix style -e '(@ (gnu packages gcc) gcc-5)'
```

modifie le style de la définition du paquet *gcc-5*.

**--input-simplification=politique**

Lorsque vous utilisez la règle de style *inputs*, avec **'-S inputs'**, cette option spécifie la politique de simplification des entrées du paquet dans le cas où les étiquettes des entrées ne correspondent pas au nom du paquet. La *politique* peut être l'une des suivantes :

**silent** Simplifie les entrées seulement si le changement est « silencieux », ce qui signifie que le paquet n'a pas besoin d'être reconstruit (sa dérivation reste inchangée).

**safe** Simplifie les entrées seulement si c'est « sûr » : le paquet peut avoir besoin d'être reconstruit, mais on est sûr que le changement n'aura aucun effet observable.

**always** Simplifie les entrées même si les étiquettes des entrées ne correspondent pas au nom du paquet, et même si le changement peut avoir un effet observable.

La valeur par défaut est `silent`, ce qui signifie que les simplifications des entrées n'entraîne aucune reconstruction de paquet.

## 10.8 Invoquer `guix lint`

La commande `guix lint` est conçue pour aider les développeur·euses à éviter des erreurs communes et à utiliser un style cohérent lors de l'écriture de recettes de paquets. Elle lance des vérifications sur un ensemble de paquets donnés pour trouver des erreurs communes dans leur définition. Les *vérifieurs* disponibles comprennent (voir `--list-checkers` pour une liste complète) :

**synopsis**

**description**

Vérifie certaines règles typographiques et stylistiques dans les descriptions et les synopsis.

**inputs-should-be-native**

Identifie les entrées qui devraient sans doute plutôt être des entrées natives.

**source**

**home-page**

**mirror-url**

**github-url**

**source-file-name**

Sonde les URL `home-page` et `source` et rapporte celles qui sont invalides. Suggère une URL en `mirror://` lorsque c'est possible. Si l'URL de `source` redirige vers une URL GitHub, recommande d'utiliser l'URL GitHub. Vérifie que le nom du fichier `source` a un sens, p. ex. qu'il ne s'agisse pas juste d'un numéro de version ou « `git-checkout` », sans avoir déclaré un `file-name` (voir Section 9.2.2 [référence de origin], page 113).

**source-unstable-tarball**

Analyse l'URL `source` pour déterminer si une archive de GitHub est autogénérée ou s'il s'agit d'une archive de publication. Malheureusement les archives autogénérées de GitHub sont parfois régénérées.

**dérivation**

Vérifiez que la dérivation des paquets donnés peut être calculée avec succès pour tous les systèmes pris en charge (voir Section 9.10 [Dérivations], page 159).

**profile-collisions**

Vérifiez si l'installation des paquets donnés dans un profil risque d'entraîner des collisions. Des collisions se produisent lorsque plusieurs paquets portant le même nom mais avec une version différente ou un nom de fichier de stockage différent sont propagés. Voir Section 9.2.1 [référence de package], page 108, pour plus d'informations sur les entrées propagées.

**archivage**

Vérifie si le code source du paquet est archivé à Software Heritage (<https://www.softwareheritage.org>).

Lorsque le code source non archivé provient d'un système de contrôle de version (VCS)—par exemple, il est obtenu avec `git-fetch`, envoyez au Software





```
"CVE-2011-5244")))))
```

**formatting**

Avertit de problèmes de formatage du code source évidents : des espaces en fin de ligne, des tabulations, etc.

**input-labels**

Rapport les étiquettes de l'ancien style qui ne correspondent pas au nom du paquet. Cela a pour but d'aider à migrer de « l'ancien style ». Voir Section 9.2.1 [référence de package], page 108, pour plus d'information sur les entrées des paquets et les styles d'entrées. Voir Section 10.7 [Invoquer guix style], page 210, pour la manière de migrer vers le nouveau style.

La syntaxe générale est :

```
guix lint options package...
```

Si aucun paquet n'est donné par la ligne de commande, tous les paquets seront vérifiés. Les *options* peuvent contenir aucune ou plus des options suivantes :

**--list-checkers**

**-l** Liste et décrit tous les vérificateurs disponibles qui seront lancés sur les paquets puis quitte.

**--checkers**

**-c** N'active que les vérificateurs spécifiés dans une liste de noms séparés par des virgules parmi la liste renvoyée par **--list-checkers**.

**--exclude**

**-x** Ne désactive que les vérifications spécifiés dans une liste de noms séparés par des virgules, en utilisant les noms de **--list-checkers**.

**--expression=expr**

**-e expr** Considérer le paquet évalué par *expr*.

C'est utile pour désigner des paquets sans ambiguïté, comme dans cet exemple :

```
guix lint -c archival -e '(@ (gnu packages guile) guile-3.0)'
```

**--no-network**

**-n** N'active que les vérificateurs qui ne dépendent pas d'un accès réseau.

**--load-path=répertoire****-L répertoire**

Ajoute *répertoire* au début du chemin de recherche de module de paquets (voir Section 9.1 [Modules de paquets], page 104).

Cela permet à des utilisateurs de définir leur propres paquets et les rendre disponibles aux outils en ligne de commande.

## 10.9 Invoquer guix size

La commande **guix size** aide à dresser un profil de l'utilisation de l'espace disque par les paquets. Il est facile de négliger l'impact d'une dépendance supplémentaire ajoutée à un paquet, ou l'impact de l'utilisation d'une sortie unique pour un paquet qui pourrait être

facilement séparé (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59). Ce sont ces problèmes que `guix size` peut typiquement mettre en valeur.

On peut passer un ou plusieurs spécifications de paquets à la commande, comme `gcc@4.8` ou `guile:debug`, ou un nom de fichier dans le dépôt. Regardez cet exemple :

```
$ guix size coreutils
store item                                total    self
/gnu/store/...-gcc-5.5.0-lib              60.4    30.1  38.1%
/gnu/store/...-glibc-2.27                 30.3    28.8  36.6%
/gnu/store/...-coreutils-8.28             78.9    15.0  19.0%
/gnu/store/...-gmp-6.1.2                  63.1     2.7   3.4%
/gnu/store/...-bash-static-4.4.12         1.5     1.5   1.9%
/gnu/store/...-acl-2.2.52                 61.1     0.4   0.5%
/gnu/store/...-attr-2.4.47                60.6     0.2   0.3%
/gnu/store/...-libcap-2.25                60.5     0.2   0.2%
total: 78.9 MiB
```

Les éléments du dépôt listés ici constituent la *clôture transitive* de Coreutils — c.-à-d. Coreutils et toutes ses dépendances, récursivement — comme ce qui serait renvoyé par :

```
$ guix gc -R /gnu/store/...-coreutils-8.23
```

Ici, la sortie possède trois colonnes à côté de chaque élément du dépôt. La première colonne, nommée « total », montre la taille en mébioctet (Mio) de la clôture de l'élément du dépôt — c'est-à-dire sa propre taille plus la taille de ses dépendances. La colonne suivante, nommée « lui-même », montre la taille de l'élément lui-même. La dernière colonne montre le ration de la taille de l'élément lui-même par rapport à celle de tous les éléments montrés.

Dans cet exemple, on voit que la clôture de Coreutils pèse 79 Mio, dont la plupart est dû à la libc et aux bibliothèques à l'exécution de GCC (ce n'est pas un problème en soit que la libc et les bibliothèques de GCC représentent une grande part de la clôture parce qu'elles sont toujours disponibles sur le système de toute façon).

Comme la commande accepte également les noms de fichiers de dépôt, il est facile d'évaluer la taille d'un résultat de construction :

```
guix size $(guix system build config.scm)
```

Lorsque les paquets passés à `guix size` sont disponibles dans le dépôt<sup>1</sup>, `guix size` demande au démon de déterminer ses dépendances, et mesure sa taille dans le dépôt, comme avec du `-ms --apparent-size` (voir Section “du invocation” dans *GNU Coreutils*).

Lorsque les paquets donnés ne sont *pas* dans le dépôt, `guix size` rapporte les informations en se basant sur les substituts disponibles (voir Section 6.3 [Substituts], page 54). Cela permet de profiler l'utilisation du disque des éléments du dépôt même s'ils ne sont pas sur le disque, mais disponibles à distance.

Vous pouvez aussi spécifier plusieurs noms de paquets :

```
$ guix size coreutils grep sed bash
store item                                total    self
/gnu/store/...-coreutils-8.24             77.8    13.8  13.4%
```

<sup>1</sup> Plus précisément, `guix size` cherche les variantes *non greffées* des paquets donnés, tels qu'ils sont renvoyés par `guix build paquet --no-graft`. Voir Chapitre 19 [Mises à jour de sécurité], page 657, pour des informations sur les greffes

```

/gnu/store/...-grep-2.22          73.1    0.8   0.8%
/gnu/store/...-bash-4.3.42       72.3    4.7   4.6%
/gnu/store/...-readline-6.3     67.6    1.2   1.2%
...
total: 102.3 MiB

```

Dans cet exemple on voit que la combinaison des quatre paquets prend 102.3 Mio en tout, ce qui est bien moins que la somme des clôtures puisqu'ils ont beaucoup de dépendances en commun.

En regardant le profil renvoyé par `guix size`, vous pouvez vous demander pourquoi un paquet donné apparaît dans le profil. Pour le comprendre, vous pouvez utiliser `guix graph --path -t references` pour afficher le chemin le plus court entre les deux paquets (voir Section 10.10 [Invoquer `guix graph`], page 218).

Les options disponibles sont :

`--substitute-urls=urls`

Utilise les informations de substituts de *urls*. Voir [client-substitute-urls], page 181.

`--sort=clef`

Trie les lignes en fonction de la *clef*, l'une des options suivantes :

`lui-même` la taille de chaque élément (par défaut) ;

`closure` la taille totale de la clôture de l'élément.

`--map-file=fichier`

Écrit un schéma de l'utilisation du disque au format PNG dans *fichier*.

Pour l'exemple au-dessus, le schéma ressemble à ceci :



Cette option requiert l'installation de Guile-Charting (<https://wingolog.org/software/guile-charting/>) et qu'il soit visible dans le chemin de recherche des modules Guile. Lorsque ce n'est pas le cas, `guix size` plante en essayant de le charger.

`--system=système`

`-s système`

Considère les paquets pour *système* — p. ex. `x86_64-linux`.

`--load-path=répertoire`

`-L répertoire`

Ajoute *répertoire* au début du chemin de recherche de module de paquets (voir Section 9.1 [Modules de paquets], page 104).

Cela permet à des utilisateurs de définir leur propres paquets et les rendre disponibles aux outils en ligne de commande.

## 10.10 Invoque guix graph

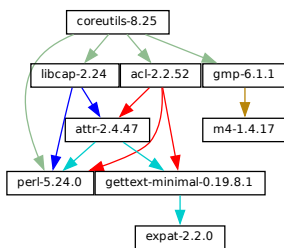
Les paquets et leurs dépendances forment un *graphe*, plus précisément un graphe orienté acyclique (DAG). Il peut vite devenir difficile d'avoir une représentation mentale du DAG d'un paquet, donc la commande `guix graph` fournit une représentation visuelle du DAG. Par défaut, `guix graph` émet un représentation du DAG dans le format d'entrée de Graphviz (<https://www.graphviz.org/>), pour que sa sortie puisse être passée directement à la commande `dot` de Graphviz. Elle peut aussi émettre une page HTML avec du code Javascript pour afficher un « diagramme d'accords » dans un navigateur Web, grâce à la bibliothèque `d3.js` (<https://d3js.org/>), ou émettre des requêtes Cypher pour construire un graphe dans une base de donnée de graphes supportant le langage de requêtes openCypher (<https://www.opencypher.org/>). Avec l'option `--path`, elle affiche simplement le chemin le plus court entre deux paquets. La syntaxe générale est :

```
guix graph options paquet...
```

Par exemple, la commande suivante génère un fichier PDF représentant le DAG du paquet pour GNU Core Utilities, qui montre ses dépendances à la compilation :

```
guix graph coreutils | dot -Tpdf > dag.pdf
```

La sortie ressemble à ceci :



Joli petit graphe, non ?

Vous pouvez trouver plus amusant de naviguer dans le graphe interactivement avec `xdot` (du paquet `xdot`) :

```
guix graph coreutils | xdot -
```

Mais il y a plus qu'un seul graphe ! Celui au-dessus est concis : c'est le graphe des objets paquets, en omettant les entrées implicites comme GCC, libc, grep, etc. Il est souvent utile d'avoir ces graphes concis, mais parfois on veut voir plus de détails. `guix graph` supporte plusieurs types de graphes, qui vous permettent de choisir le niveau de détails :

**package** C'est le type par défaut utilisé dans l'exemple plus haut. Il montre le DAG des objets paquets, sans les dépendances implicites. C'est concis, mais omet pas mal de détails.

**reverse-package**

Cela montre le DAG *inversé* des paquets. Par exemple :

```
guix graph --type=reverse-package ocaml
```

... crée le graphe des paquets qui dépendent *explicitement* d'OCaml (si vous êtes aussi intéressé-e dans le cas où OCaml représente une dépendance implicite, voir **reverse-bag** ci-dessous).

Remarquez que pour les paquets du cœur de la distribution, cela crée des graphes énormes. Si vous voulez seulement voir le nombre de paquets qui dépendent d'un paquet donné, utilisez **guix refresh --list-dependent** (voir Section 10.6 [Invoquer guix refresh], page 205).

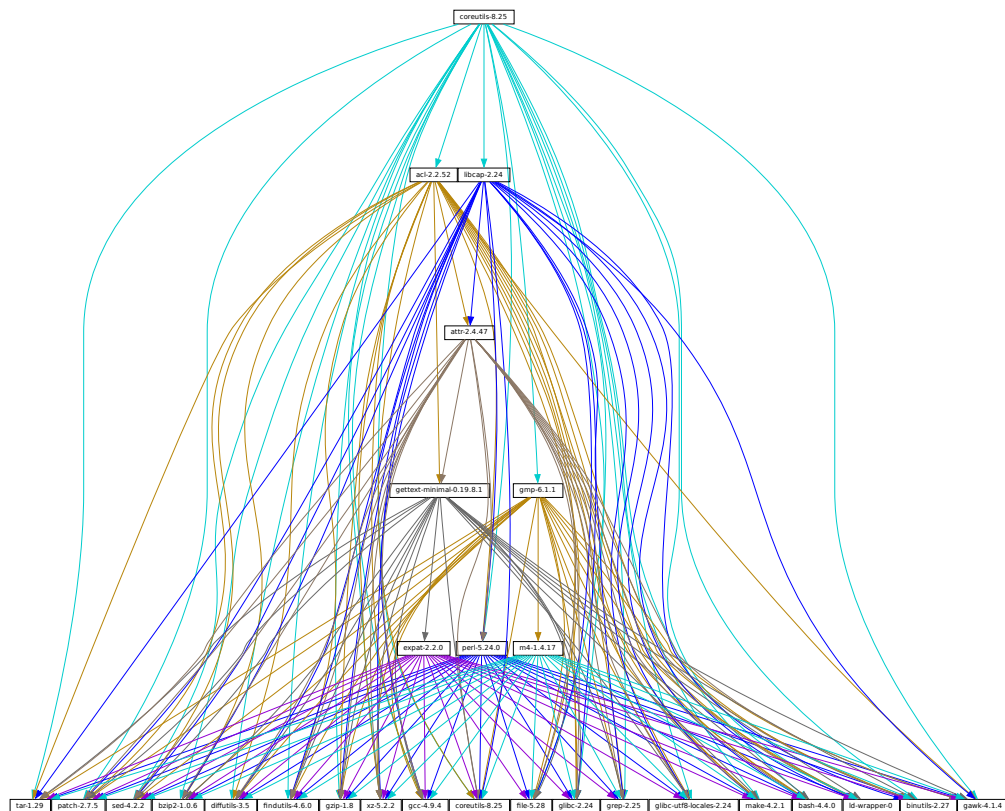
**bag-emerged**

C'est le DAG du paquet, *avec* les entrées implicites.

Par exemple, la commande suivante :

```
guix graph --type=bag-emerged coreutils
```

... montre ce graphe plus gros :



En bas du graphe, on voit toutes les entrées implicites de *gnu-build-system* (voir Section 9.5 [Systèmes de construction], page 124).

Maintenant, remarquez que les dépendances de ces entrées implicites — c'est-à-dire les *dépendances de bootstrap* (voir Chapitre 20 [Bootstrapping], page 659) — ne sont pas affichées, pour rester concis.

**bag** Comme **bag-emerged** mais cette fois inclus toutes les dépendances de bootstrap.

**bag-with-origins**

Comme **bag**, mais montre aussi les origines et leurs dépendances.

**reverse-bag**

Cela montre le DAG *inverse* des paquets. Contrairement à **reverse-package**, il montre aussi les dépendance implicites. Par exemple :

```
guix graph -t reverse-bag dune
```

... crée le graphe des tous les paquets qui dépendent de Dune, directement ou indirectement. Comme Dune est une dépendance *implicite* de nombreux paquets *via dune-build-system*, cela montre un plus grand nombre de paquets, alors que **reverse-package** en montrerait très peu, voir aucun.

**dérivation**

C'est la représentation la plus détaillée : elle montre le DAG des dérivations (voir Section 9.10 [Dérivations], page 159) et des éléments du dépôt. Comparé à la représentation ci-dessus, beaucoup plus de nœuds sont visibles, dont les scripts de construction, les correctifs, les modules Guile, etc.

Pour ce type de graphe, il est aussi possible de passer un nom de fichier `.drv` à la place d'un nom de paquet, comme dans :

```
guix graph -t derivation $(guix system build -d my-config.scm)
```

**module** C'est le graphe des *modules de paquets* (voir Section 9.1 [Modules de paquets], page 104). Par exemple, la commande suivante montre le graphe des modules de paquets qui définissent le paquet `guile` :

```
guix graph -t module guile | xdot -
```

Tous les types ci-dessus correspondent aux *dépendances à la construction*. Le type de graphe suivant représente les *dépendances à l'exécution* :

**references**

C'est le graphe des *references* d'une sortie d'un paquet, telles que renvoyées par `guix gc --references` (voir Section 6.5 [Invoquer guix gc], page 60).

Si la sortie du paquet donnée n'est pas disponible dans le dépôt, `guix graph` essaiera d'obtenir les informations sur les dépendances à travers les substituts.

Vous pouvez aussi passer un nom de fichier du dépôt plutôt qu'un nom de paquet. Par exemple, la commande ci-dessous produit le graphe des références de votre profile (qui peut être gros !)

```
guix graph -t references $(readlink -f ~/.guix-profile)
```

**referrers**

C'est le graphe des *référents* d'un élément du dépôt, tels que renvoyés par `guix gc --referrers` (voir Section 6.5 [Invoquer guix gc], page 60).

Cela repose exclusivement sur les informations de votre dépôt. Par exemple, supposons que Inkscape est actuellement disponible dans 10 profils sur votre

machine ; `guix graph -t referrers inkscape` montrera le graphe dont la racine est Inkscape avec 10 profils qui y sont liés.

Cela peut aider à déterminer ce qui empêche un élément du dépôt d'être glané.

Souvent, le graphe du paquet qui vous intéresse ne tient pas sur votre écran, et de toute façon tout ce que vous voulez savoir c'est *pourquoi* ce paquet dépend en fait d'un paquet apparemment sans rapport. L'option `--path` indique à `guix graph` d'afficher le chemin le plus court entre deux paquets (ou dérivations, ou éléments du dépôt, etc.) :

```
$ 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
```

Parfois vous voudrez quand même visualiser le graphe mais préférerez le recouper afin qu'il puisse être affiché. Un moyen pour ce faire est l'option `--max-depth` (ou `-M`), qui vous laisse spécifier la profondeur maximale du graphe. Dans l'exemple ci-dessous, nous visualisons seulement `libreoffice` et les nœuds dont la distance à `libreoffice` est au plus 2 :

```
guix graph -M 2 libreoffice | xdot -f fdp -
```

Attention hein, c'est encore un sacré plat de spaghetti, mais au moins `dot` peut en faire un rendu rapide et il est possible de le parcourir un peu.

Les options disponibles sont les suivantes :

```
--type=type
-t type    Produit un graphe en sortie de type type où type doit être l'un des types
           au-dessus.

--list-types
           Liste les types de graphes supportés.

--backend=moteur
-b moteur Produit un graphe avec le moteur choisi.

--list-backends
           Liste les moteurs de graphes supportés.
           Actuellement les moteurs disponibles sont Graphviz et d3.js.

--path     Affiche le chemin le plus court entre deux nœuds du type spécifié par --type.
           L'exemple ci-dessous montre le chemin le plus court entre libreoffice et llvm
           correspondant aux références de libreoffice :
```

```
$ guix graph --path -t references libreoffice llvm
```



```

/gnu/store/...-libreoffice-6.4.2.2
/gnu/store/...-libepoxy-1.5.4
/gnu/store/...-mesa-19.3.4
/gnu/store/...-llvm-9.0.1
--expression=expr
-e expr    Considérer le paquet évalué par expr.
            C'est utile pour précisément se référer à un paquet, comme dans cet exemple :
            guix graph -e '(@@ (gnu packages commencement) gnu-make-final)'
--system=système
-s système
            Affiche le graphe pour système — p. ex. i686-linux.
            Le graphe de dépendance des paquets est la plupart du temps indépendant de
            l'architecture, mais il y a quelques parties qui dépendent de l'architecture que
            cette option vous permet de visualiser.
--load-path=répertoire
-L répertoire
            Ajoute répertoire au début du chemin de recherche de module de paquets (voir
            Section 9.1 [Modules de paquets], page 104).
            Cela permet à des utilisateurs de définir leur propres paquets et les rendre
            disponibles aux outils en ligne de commande.

```

En plus de cela, `guix graph` prend en charge toutes les options habituelles de transformation des paquets (voir Section 10.1.2 [Options de transformation de paquets], page 183). Il est ainsi facile de voir l'effet d'une transformation de réécriture de graphe telle que `--with-input`. Par exemple, la commande ci-dessous produit le graphe de `git` une fois que `openssl` a été remplacé par `libressl` partout dans le graphe :

```
guix graph git --with-input=openssl=libressl
```

Tant de possibilités, tant de plaisir !

## 10.11 Invoquer `guix publish`

Le but de `guix publish` est de vous permettre de partager facilement votre dépôt avec d'autres personnes qui peuvent ensuite l'utiliser comme serveur de substituts (voir Section 6.3 [Substituts], page 54).

Lorsque `guix publish` est lancé, il crée un serveur HTTP qui permet à n'importe qui avec un accès réseau d'y récupérer des substituts. Cela signifie que toutes les machines qui font tourner Guix peuvent aussi agir comme une ferme de construction, puisque l'interface HTTP est compatible avec Cuirass, le logiciel derrière la ferme de construction `ci.guix.gnu.org`.

Pour des raisons de sécurité, chaque substitut est signé, ce qui permet aux destinataires de vérifier leur authenticité et leur intégrité (voir Section 6.3 [Substituts], page 54). Parce que `guix publish` utilise la clé de signature du système, qui n'est lisible que par l'administrateur système, il doit être lancé en `root` ; l'option `--user` lui fait abandonner les privilèges de `root` dès le début.

La paire de clés pour les signatures doit être générée avant de lancer `guix publish`, avec `guix archive --generate-key` (voir Section 6.10 [Invoquer `guix archive`], page 71).

Lorsque vous passez l'option `--advertise`, le serveur annonce sa disponibilité sur le réseau local en utilisant le DNS multicast (mDNS) et le protocole de découverte de service DNS (DNS-SD), actuellement via Guile-Avahi (voir *Using Avahi in Guile Scheme Programs*).

La syntaxe générale est :

```
guix publish options...
```

Lancer `guix publish` sans arguments supplémentaires lancera un serveur HTTP sur le port 8080 :

```
guix publish
```

`guix publish` peut aussi être démarré avec le protocole d'« activation par socket » de `systemd` (voir Section “Service De- and Constructors” dans *le manuel de GNU Shepherd*).

Une fois qu'un serveur de publication a été autorisé, le démon peut télécharger des substituts à partir de lui. Voir Section 6.3.3 [Récupérer des substituts d'autres serveurs], page 56.

Par défaut, `guix publish` compresse les archives à la volée quand il les sert. Ce mode « à la volée » est pratique puisqu'il ne demande aucune configuration et est disponible immédiatement. Cependant, lorsqu'il s'agit de servir beaucoup de clients, nous recommandons d'utiliser l'option `--cache`, qui active le cache des archives avant de les envoyer aux clients — voir les détails plus bas. La commande `guix weather` fournit un manière pratique de vérifier ce qu'un serveur fournit (voir Section 10.15 [Invoquer `guix weather`], page 231).

En bonus, `guix publish` sert aussi un miroir adressé par le contenu des fichiers source référencés dans les enregistrements `origin` (voir Section 9.2.2 [référence de `origin`], page 113). Par exemple, en supposant que `guix publish` tourne sur `example.org`, l'URL suivante renverra le fichier brut `hello-2.10.tar.gz` avec le hash SHA256 donné (représenté sous le format `nix-base32`, voir Section 10.4 [Invoquer `guix hash`], page 196) :

```
http://example.org/file/hello-2.10.tar.gz/sha256/0ssi1...ndq1i
```

Évidemment, ces URL ne fonctionnent que pour des fichiers dans le dépôt ; dans les autres cas, elles renvoient une erreur 404 (« Introuvable »).

Les journaux de construction sont disponibles à partir des URL `/log` comme ceci :

```
http://example.org/log/gwspk...-guile-2.2.3
```

Lorsque `guix-daemon` est configuré pour sauvegarder les journaux de construction compressés, comme c'est le cas par défaut (voir Section 2.5 [Invoquer `guix-daemon`], page 17), les URL `/log` renvoient le journal compressé tel-quel, avec une en-tête `Content-Type` et/ou `Content-Encoding` appropriée. Nous recommandons de lancer `guix-daemon` avec `--log-compression=gzip` parce que les navigateurs web les décompressent automatiquement, ce qui n'est pas le cas avec la compression `bzip2`.

Les options suivantes sont disponibles :

```
--port=port
```

```
-p port    Écoute les requêtes HTTP sur le port.
```

```
--listen=hôte
```

```
    Écoute sur l'interface réseau de hôte. Par défaut, la commande accepte les connexions de n'importe quelle interface.
```

`--user=utilisateur`

`-u utilisateur`

Charge les privilèges de *utilisateur* le plus vite possible — c.-à-d. une fois que la socket du serveur est ouverte et que la clef de signature a été lue.

`--compression[=méthode[:niveau]]`

`-C [méthode[:niveau]]`

Compresser les données en utilisant les *method* et *level* donnés. *method* est l'un des codes `lzip`, `zstd` ou `gzip` ; lorsque *method* est omis, `gzip` est utilisé.

Lorsque *level* est égal à zéro, désactivez la compression. La plage de 1 à 9 correspond à différents niveaux de compression : 1 est le plus rapide, et 9 est le meilleur (gourmand en CPU). Le niveau par défaut est 3.

Habituellement, `lzip` compresse notablement mieux que `gzip` pour une légère augmentation de l'utilisation du CPU. voir benchmarks on the lzip Web page ([https://nongnu.org/lzip/lzip\\_benchmark.html](https://nongnu.org/lzip/lzip_benchmark.html)). Cependant, `lzip` permet un faible débit à la décompression (de l'ordre de 50 Mio/s sur du matériel récent), ce qui peut être le facteur limitant pour quelqu'un qui télécharge sur un réseau rapide.

Le ratio de compression de `zstd` est entre celui de `lzip` et celui de `gzip` ; son avantage principal est sa high vitesse de décompression (<https://facebook.github.io/zstd/>).

À moins que `--cache` ne soit utilisé, la compression se fait à la volée et les flux compressés ne sont pas cachés. Ainsi, pour réduire la charge sur la machine qui fait tourner `guix publish`, c'est une bonne idée de choisir un niveau de compression faible, de lancer `guix publish` derrière un serveur de cache ou d'utiliser `--cache`. Utilise `--cache` à l'avantage qu'il permet à `guix publish` d'ajouter l'en-tête `HTTP Content-Length` à sa réponse.

Cette option peut être répétée, auquel cas chaque substitut est compressé en utilisant toutes les méthodes sélectionnées, et toutes sont annoncées. Cette option est utile lorsque les utilisateur-riche-s ne supportent pas toutes les méthodes de compression : il-elle-s peuvent sélectionner celle qu'elles supportent.

`--cache=répertoire`

`-c répertoire`

Cache les archives et les métadonnées (les URL `.narinfo`) dans *répertoire* et ne sert que les archives dans ce cache.

Lorsque cette option est omise, les archives et les métadonnées sont créées à la volée. Cela réduit la bande passante disponible, surtout quand la compression est activée puisqu'elle pourrait être limitée par le CPU. Un autre inconvénient au mode par défaut est que la taille des archives n'est pas connue à l'avance, donc `guix publish` n'ajoute pas l'en-tête `Content-Length` à ses réponses, ce qui empêche les clients de savoir la quantité de données à télécharger.

À l'inverse, lorsque `--cache` est utilisée, la première requête pour un élément du dépôt (via une URL `.narinfo`) déclenche une tâche de fond pour créer l'archive — en calculant son `.narinfo` et en compressant l'archive au besoin. Une fois l'archive en cache dans *répertoire*, les requêtes suivantes réussissent et

sont servies directement depuis le cache, ce qui garanti que les clients ont la meilleure bande passante possible.

Cette première requête `.narinfo` renvoie quand même 200, si l'élément du dépôt est « assez petit », sous la limite de contournement du cache — voir `--cache-bypass-threshold` plus bas. De cette manière, les clients n'ont pas besoin d'attendre que l'archive soit prête. Pour les éléments plus gros, la première requête `.narinfo` renvoie 404, ce qui signifie que les clients doivent attendre jusqu'à ce que l'archive soit prête.

Le processus de création est effectué par des threads de travail. Par défaut, un thread par cœur du CPU est créé, mais cela peut être personnalisé. Voir `--workers` plus bas.

Lorsque l'option `--ttl` est utilisée, les entrées cachées sont automatiquement supprimées lorsqu'elles expirent.

#### `--workers=N`

Lorsque `--cache` est utilisée, demande l'allocation de  $N$  thread de travail pour créer les archives.

#### `--ttl=ttl`

Produit des en-têtes HTTP `Cache-Control` qui annoncent une durée de vie (TTL) de  $ttl$ .  $ttl$  peut dénoter une durée : `5d` signifie 5 jours, `1m` signifie un mois, etc.

Cela permet au Guix de l'utilisateur de garder les informations en cache pendant  $ttl$ . Cependant, remarquez que `guix publish` ne garanti pas lui-même que les éléments du dépôt qu'il fournit seront toujours disponible pendant la durée  $ttl$ .

En plus, lorsque `--cache` est utilisée, les entrées cachées qui n'ont pas été demandé depuis  $ttl$  et n'ont pas d'élément correspondant dans le dépôt peuvent être supprimées.

#### `--negative-ttl=ttl`

Produit de la même manière des en-têtes HTTP `Cache-Control` pour annoncer la durée de vie (TLL) d'une recherche *négative* — des entrées du dépôt manquantes, pour lesquelles une erreur 404 est renvoyée. Par défaut, aucun TTL négatif n'est annoncé.

Ce paramètre peut aider à ajuster la charge du serveur et la latence des substitués en demandant aux clients coopératifs d'être plus ou moins patients quand un élément du dépôt manque.

#### `--cache-bypass-threshold=taille`

Avec `--cache`, les éléments du dépôt plus petits que  $taille$  sont immédiatement disponibles, même s'ils ne sont pas en cache.  $taille$  est une taille en octets, ou peut utiliser un suffixe comme `M` pour les mégaoctets, etc. La valeur par défaut est `10M`.

Le « contournement du cache » vous permet de réduire le délai de publication pour les clients au prix d'utilisation accrue des I/O et du CPU sur le serveur : en fonction des accès clients, ces éléments peuvent être créés plusieurs fois avant qu'une copie ne soit disponible dans le cache.

Augmenter la limite peut être utile pour les sites qui ont peu d'utilisateurs, ou pour garantir que les utilisateurs reçoivent les substituts même pour les éléments qui ne sont pas populaires.

`--nar-path=chemin`

Utilise *chemin* comme préfixe des URL de fichier « nar » (voir Section 6.10 [Invoquer guix archive], page 71).

Par défaut, les nars sont présents à l'URL comme `/nar/gzip/...-coreutils-8.25`. Cette option vous permet de changer la partie `/nar` en *chemin*.

`--public-key=fichier`

`--private-key=fichier`

Utilise les *fichiers* spécifiques comme pair de clefs utilisées pour signer les éléments avant de les publier.

Les fichiers doivent correspondre à la même pair de clefs (la clef privée est utilisée pour signer et la clef publique est seulement ajouté aux métadonnées de la signature). Ils doivent contenir les clefs dans le format s-expression canonique produit par `guix archive --generate-key` (voir Section 6.10 [Invoquer guix archive], page 71). Par défaut, `/etc/guix/signing-key.pub` et `/etc/guix/signing-key.sec` sont utilisés.

`--repl[=port]`

`-r [port]` Crée un serveur REPL Guile (voir Section “REPL Servers” dans *GNU Guile Reference Manual*) sur *pport* (37146 par défaut). C'est surtout utile pour déboguer un serveur `guix publish` qui tourne.

Activer `guix publish` sur un système Guix est vraiment une seule ligne : instanciez simplement un service `guix-publish-service-type` dans le champs `services` de votre déclaration `operating-system` (voir [guix-publish-service-type], page 281).

Si vous utilisez plutôt Guix sur une « distro étrangère », suivez ces instructions :

- Si votre distro hôte utilise le système d'init `systemd` :
 

```
# ln -s ~root/.guix-profile/lib/systemd/system/guix-publish.service \
    /etc/systemd/system/
# systemctl start guix-publish && systemctl enable guix-publish
```
- Si votre distribution hôte utilise le système d'initialisation `Upstart` :
 

```
# ln -s ~root/.guix-profile/lib/upstart/system/guix-publish.conf /etc/init/
# start guix-publish
```
- Sinon, procédez de manière similaire avec votre système d'init de votre distro.

## 10.12 Invoquer guix challenge

Est-ce que les binaires fournis par ce serveur correspondent réellement au code source qu'il dit avoir construit ? Est-ce que le processus de construction d'un paquet est déterministe ? Ce sont les question auxquelles la commande `guix challenge` essaye de répondre.

La première question est évidemment importante : avant d'utiliser un serveur de substituts (voir Section 6.3 [Substituts], page 54), il vaut mieux *vérifier* qu'il fournit les bons binaires et donc le *défier*. La deuxième est ce qui permet la première : si les constructions

des paquets sont déterministes alors des constructions indépendantes du paquet devraient donner le même résultat, bit à bit ; si un serveur fournit un binaire différent de celui obtenu localement, il peut être soit corrompu, soit malveillant.

On sait que le hash qui apparaît dans `/gnu/store` est le hash de toutes les entrées du processus qui construit le fichier ou le répertoire — les compilateurs, les bibliothèques, les scripts de construction, etc. (voir Chapitre 1 [Introduction], page 1). En supposant que les processus de construction sont déterministes, un nom de fichier dans le dépôt devrait correspondre exactement à une sortie de construction. `guix challenge` vérifie si il y a bien effectivement une seule correspondance en comparant les sorties de plusieurs constructions indépendantes d'un élément du dépôt donné.

La sortie de la commande ressemble à :

```
$ guix challenge \
  --substitute-urls="https://ci.guix.gnu.org https://guix.example.org" \
  openssl git pius coreutils grep
mise à jour des substituts depuis 'https://ci.guix.gnu.org'... 100.0%
mise à jour des substituts depuis 'https://guix.example.org'... 100.0%
le contenu de /gnu/store/...-openssl-1.0.2d diffère :
  empreinte locale : 0725122r5jnzazaacncwsvp9kgf42266ayyp814v7djxs7nk963q
  https://ci.guix.gnu.org/nar/...-openssl-1.0.2d : 0725122r5jnzazaacncwsvp9kgf42266ayyp814v7djxs7nk963q
  https://guix.example.org/nar/...-openssl-1.0.2d : 1zy4fmaaocnrzzajkdn3f5gmjk754b43qkq471lbyak9z0qjyim
  fichiers différents :
    /lib/libcrypto.so.1.1
    /lib/libssl.so.1.1

le contenu de /gnu/store/...-git-2.5.0 diffère :
  empreinte locale : 00p3bmryhjxrhpn2gxs2fy0a15lnip05197205pgbk5ra395hyha
  https://ci.guix.gnu.org/nar/...-git-2.5.0 : 069nb85bv4d4a6slrwjdy8v1cn4cwspm3kdbmby81d6zckj3nq9f
  https://guix.example.org/nar/...-git-2.5.0 : 0mdqa9w1p6cml16976v4wi0sw9r4p5prkj7lzf1877wk11c9c73
  fichier différent :
    /libexec/git-core/git-fsck

le contenu de /gnu/store/...-pius-2.1.1 diffère :
  empreinte locale : 0k4v3m9z1zp8xzzizb7d8kjj72f9172xv078sq4w173vniq9ig3ax
  https://ci.guix.gnu.org/nar/...-pius-2.1.1 : 0k4v3m9z1zp8xzzizb7d8kjj72f9172xv078sq4w173vniq9ig3ax
  https://guix.example.org/nar/...-pius-2.1.1 : 1cy25x1a4fzq5rk0pmvc8xhwyffnqz95h2bpvqs2mpvlbccy0gs
  fichier différent:
    /share/man/man1/pius.1.gz

...

5 éléments du dépôt ont été analysés :
- 2 (40.0%) étaient identiques
- 3 (60.0%) étaient différents
- 0 (0.0%) étaient impossibles à évaluer
```

Dans cet exemple, `guix challenge` demande à tous les serveurs de substituts les cinq paquets spécifiés sur la ligne de commande. Il rapporte ensuite les éléments du dépôt pour lesquels les serveurs obtiennent un résultat différent de la construction locale (si elle existe) ou différent entre eux ; ici, les lignes ‘`empreinte locale`’ indiquent que le résultat d’une construction locale était disponible pour chacun des paquets et montre son empreinte.

Dans l’exemple, `guix.example.org` obtient toujours une réponse différente. Inversement, `ci.guix.gnu.org` est d’accord avec les constructions locale, sauf dans le cas de Git. Cela peut indiquer que le processus de construction de Git est non-déterministe, ce qui

signifie que sa sortie diffère en fonction de divers choses que Guix ne contrôle pas parfaitement, malgré l'isolation des constructions (voir Section 6.1 [Fonctionnalités], page 43). Les sources les plus communes de non-déterminisme comprennent l'ajout d'horodatage dans les résultats des constructions, l'inclusion de nombres aléatoires et des listes de fichiers ordonnés par numéro d'incéud. Voir <https://reproducible-builds.org/docs/>, pour plus d'informations.

Pour trouver ce qui ne va pas avec ce binaire Git, l'approche la plus facile est de lancer :

```
guix challenge git \
  --diff=diffoscope \
  --substitute-urls="https://ci.guix.gnu.org https://guix.example.org"
```

Cela invoque automatiquement `diffoscope`, qui affiche des informations détaillées sur les fichiers qui diffèrent.

Autrement, nous pouvons faire quelque chose comme dans ces lignes (voir Section 6.10 [Invoquer guix archive], page 71) :

```
$ 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
```

Cette commande montre les différences entre les fichiers qui résultent de la construction locale et des fichiers qui résultent de la construction sur `ci.guix.gnu.org` (voir Section “Overview” dans *Comparing and Merging Files*). La commande `diff` fonctionne bien avec des fichiers texte. Lorsque des fichiers binaires diffèrent cependant, `Diffoscope` (<https://diffoscope.org/>) est une meilleure option. C'est un outil qui aide à visualiser les différences entre toute sorte de fichiers.

Une fois que vous avez fait ce travail, vous pourrez dire si les différences sont dues au non-déterminisme du processus de construction ou à la malhonnêteté du serveur. Nous avons fait beaucoup d'effort pour éliminer les sources de non-déterminisme dans les paquets pour rendre plus facile la vérification des substituts, mais bien sûr, c'est un processus qui n'implique pas que Guix, mais une grande partie de la communauté des logiciels libres. Pendant ce temps, `guix challenge` est un outil pour aider à corriger le problème.

Si vous écrivez un paquet pour Guix, nous vous encourageons à vérifier si `ci.guix.gnu.org` et d'autres serveurs de substituts obtiennent le même résultat que vous avec :

```
guix challenge paquet
```

La syntaxe générale est :

```
guix challenge options argument...
```

où *argument* est une spécification de paquet comme `guile@2.0` ou `glibc:debug` ou, autrement, un nom de fichier du dépôt renvoyé, par exemple, par `guix build` ou `guix gc --list-live`.

Lorsqu'une différence est trouvée entre l'empreinte d'un élément construit localement et celle d'un substitut fourni par un serveur, ou parmi les substituts fournis par différents serveurs, la commande l'affiche comme dans l'exemple ci-dessus et sa valeur de sortie est 2 (les autres valeurs différentes de 0 indiquent d'autres sortes d'erreurs).

L'option qui compte est :

- substitute-urls=urls**  
 Considère *urls* comme la liste des URL des sources de substituts séparés par des espaces avec lesquels comparer les paquets locaux.
- diff=mode**  
 En cas d'inadéquation, montre les différences en fonction de *mode*, l'un des :
- simple** (par défaut)  
 Montrer la liste des fichiers qui diffèrent.
  - diffoscope**  
*commande*  
 Invoque Diffoscope (<https://diffoscope.org/>), en lui passant deux répertoires dont le contenu ne correspond pas.  
 Quand *command* est un nom de fichier absolu, lancez *command* au lieu de Diffoscope.
  - none**  
 Ne donnez pas plus de détails sur les différences.
- Ainsi, à moins que l'option **--diff=none** ne soit passée, **guix challenge** télécharge les éléments du store à partir des serveurs de substitut donnés afin de pouvoir les comparer.
- verbose**  
**-v** Montre des détails sur les correspondances (contenu identique) en plus des informations sur différences.

### 10.13 Invoquer guix copy

La commande **guix copy** copie des éléments du dépôt d'une machine vers le dépôt d'une autre machine à travers une connexion SSH<sup>2</sup>. Par exemple, la commande suivante copie le paquet **coreutils**, le profil utilisateur et toutes leurs dépendances sur *hôte*, en tant qu'utilisateur *utilisateur* :

```
guix copy --to=utilisateur@hôte \  

  coreutils $(readlink -f ~/.guix-profile)
```

Si certains éléments à copier sont déjà présents sur *hôte*, ils ne sont pas envoyés.

La commande ci-dessous récupère **libreoffice** et **gimp** depuis *hôte*, en supposant qu'ils y sont présents :

```
guix copy --from=hôte libreoffice gimp
```

La connexion SSH est établie avec le client Guile-SSH, qui est compatible avec OpenSSH : il prend en compte `~/.ssh/known_hosts` et `~/.ssh/config` et utilise l'agent SSH pour l'authentification.

La clef utilisée pour signer les éléments qui sont envoyés doit être acceptée par la machine distante. De même, la clef utilisée pour la machine distante depuis laquelle vous récupérez des éléments doit être dans `/etc/guix/acl` pour qu'ils soient acceptés par votre propre démon. Voir Section 6.10 [Invoquer guix archive], page 71, pour plus d'informations sur l'authentification des éléments du dépôt.

<sup>2</sup> Cette commande n'est disponible que si Guile-SSH est trouvé. Voir Section 2.2 [Prérequis], page 8, pour des détails



La syntaxe générale est :

```
guix copy [--to=spec|--from=spec] items...
```

Vous devez toujours spécifier l'une des options suivantes :

`--to=spec`

`--from=spec`

Spécifie l'hôte où envoyer ou d'où recevoir les éléments. *spec* doit être une spécification SSH comme `example.org`, `charlie@example.org` ou `charlie@example.org:2222`.

L'option *items* peut être des noms de paquets, comme `gimp` ou des éléments du dépôt comme `/gnu/store/...-idutils-4.6`.

Lorsque vous spécifiez le nom d'un paquet à envoyer, il est d'abord construit au besoin, sauf si l'option `--dry-run` est spécifiée. Les options de construction communes sont supportées (voir Section 10.1.1 [Options de construction communes], page 180).

## 10.14 Invoquer guix container

**Remarque:** À la version 1.4.0, cet outil est toujours expérimental. L'interface est sujette à changement radicaux dans le futur.

Le but de `guix container` est de manipuler des processus qui tournent dans un environnement séparé, connus sous le nom de « conteneur », typiquement créés par les commandes `guix shell` (voir Section 8.1 [Invoquer guix shell], page 83) et `guix system container` (voir Section 12.15 [Invoquer guix system], page 584).

La syntaxe générale est :

```
guix container action options...
```

*action* spécifie les opérations à effectuer avec un conteneur, et *options* spécifie les arguments spécifiques au contexte pour l'action.

Les actions suivantes sont disponibles :

**exec** Exécute une commande dans le contexte d'un conteneur lancé.

La syntaxe est :

```
guix container exec pid programme arguments...
```

*pid* spécifie le PID du conteneur lancé. *programme* spécifie le nom du fichier exécutable dans le système de fichiers racine du conteneur. *arguments* sont les options supplémentaires à passer à *programme*.

La commande suivante lance un shell de connexion interactif dans un conteneur Guix System, démarré par `guix system container` et dont le PID est 9001 :

```
guix container exec 9001 /run/current-system/profile/bin/bash --login
```

Remarquez que *pid* ne peut pas être le processus parent d'un conteneur. Ce doit être le PID 1 du conteneur ou l'un de ses processus fils.

## 10.15 Invoquer guix weather

Vous pouvez parfois grogner lorsque les substituts ne sont pas disponibles et que vous devez construire les paquets vous-même (voir Section 6.3 [Substituts], page 54). La commande `guix weather` rapporte la disponibilité des substituts sur les serveurs spécifiés pour que vous sachiez si vous allez raller aujourd’hui. Cela peut parfois être une information utile pour les utilisateurs, mais elle est surtout utile pour les personnes qui font tourner `guix publish` (voir Section 10.11 [Invoquer guix publish], page 222).

Voici un exemple :

```
$ guix weather --substitute-urls=https://guix.example.org
calcul de 5,872 dérivations de paquets pour x86_64-linux...
recherche de 6,128 éléments du dépôt sur https://guix.example.org...
mise à jour des substituts depuis 'https://guix.example.org'... 100.0%
https://guix.example.org
  43.4% substituts disponibles (2,658 sur 6,128)
  7,032.5 Mo de fichiers nar (compressés)
 19,824.2 Mo sur le disque (décompressés)
  0.030 secondes par requêtes (182.9 secondes au total)
 33.5 requêtes par seconde

 9.8% (342 sur 3,470) des éléments manquants sont dans la queue
867 constructions dans la queue
  x86_64-linux : 518 (59.7%)
  i686-linux : 221 (25.5%)
  aarch64-linux : 128 (14.8%)
vitesse de construction : 23.41 constructions par heure
  x86_64-linux : 11.16 constructions par heure
  i686-linux : 6.03 constructions par heure
  aarch64-linux : 6.41 constructions par heure
```

Comme vous pouvez le voir, elle rapporte le pourcentage des paquets pour lesquels des substituts sont disponibles sur le serveur — indépendamment du fait que les substituts soient activés, et indépendamment du fait que la clef de signature du serveur soit autorisée. Elle rapporte aussi la taille des archives compressées (« nars ») fournies par le serveur, la taille des éléments du dépôt correspondant dans le dépôt (en supposant que la déduplication soit désactivée) et la vitesse du serveur. La deuxième partie donne des statistiques sur l’intégration continue (CI), si le serveur le supporte. En plus, avec l’option `--coverage`, `guix weather` peut lister les substituts de paquets « importants » qui font défaut sur le serveur (voir plus bas).

Pour cela, `guix weather` récupère par HTTP(S) les métadonnées (*narinfos* de tous les éléments du dépôts pertinents. Comme `guix challenge`, il ignore les signatures de ces substituts, ce qui n’est pas dangereux puisque la commande ne fait que récupérer des statistiques et n’installe pas ces substituts.

La syntaxe générale est :

```
guix weather options... [paquets...]
```

Lorsque `packages` est omis, `guix weather` vérifie la disponibilité de substituts pour *all* les paquets, ou pour ceux spécifiés avec `--manifest` ; sinon, il ne prend en compte que les

paquets spécifiés. Il est également possible d'interroger des types de systèmes spécifiques avec `--system`. `guix weather` sort avec un code non nul lorsque la fraction des substituts disponibles est inférieure à 100%.

Les options disponibles sont listées ci-dessous.

`--substitute-urls=urls`

`urls` est la liste des URL des serveurs de substituts séparés par des espaces. Lorsque cette option n'est pas renseignée, l'ensemble des serveurs de substituts par défaut est utilisé.

`--system=système`

`-s système`

Effectue des requêtes pour les substituts `système` — p. ex. `aarch64-linux`. Cette option peut être répétée, auquel cas `guix weather` demandera les substituts de plusieurs types de systèmes.

`--manifest=fichier`

Plutôt que de demander des substituts pour tous les paquets, demande uniquement les paquets spécifiés dans `fichier`. `fichier` doit contenir un *manifeste* comme avec l'option `-m` de `guix package` (voir Section 6.2 [Invoquer `guix package`], page 44).

Cette option peut être répétée plusieurs fois, auquel cas les manifestes sont concaténés.

`--coverage[=count]`

`-c [count]`

Rapporte la couverture des substituts pour les paquets : liste les paquets avec au moins `count` autres paquets qui en dépendent (zéro par défaut) pour lesquels il n'y a pas de substitut. Les paquets qui en dépendent ne sont pas listés : si `b` dépend de `a` et que `a` n'a pas de substitut, seul `a` est listé, même si `b` n'a habituellement pas de substitut non plus. Le résultat ressemble à cela :

```
$ guix weather --substitute-urls=https://ci.guix.gnu.org https://bordeaux.guix.gnu.org
calcul de 8 983 dérivations de paquets pour x86_64-linux...
recherche de 9 343 éléments du dépôt sur https://ci.guix.gnu.org
https://bordeaux.guix.gnu.org...
mise à jour des substituts depuis « https://ci.guix.gnu.org https://bordeaux.guix.gnu.org
https://ci.guix.gnu.org https://bordeaux.guix.gnu.org
  64.7 % des substituts sont disponibles (6,047 sur 9,343)
...
2502 paquets ne sont pas sur « https://ci.guix.gnu.org https://bordeaux.guix.gnu.org
  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
...
```

Ce que montre cet exemple est que `kcoreaddons` et probablement les 58 paquets qui en dépendent n'ont pas de substituts sur `ci.guix.gnu.org` ; de même pour `qgpgme` et les 46 paquets qui en dépendent.

Si vous êtes un développeur de Guix, ou si vous prenez soin de cette ferme de construction, vous voudrez sans doute inspecter plus finement ces paquets : ils peuvent simplement avoir échoué à la construction.

`--display-missing`

Afficher la liste des articles du magasin pour lesquels il manque des substituts.

## 10.16 Invoquer guix processes

La commande `guix processes` peut être utile pour les développeur-euses ou les personnes qui administrent des systèmes, surtout sur des machines multi-utilisateur-riche et sur les fermes de construction : elle liste les sessions actuelles (les connexions au démon), ainsi que des informations sur les processus en question<sup>3</sup>. Voici un exemple des informations qu'elle renvoie :

```
$ sudo guix processes
SessionPID: 19002
ClientPID: 19090
ClientCommand: guix shell python

SessionPID: 19402
ClientPID: 19367
ClientCommand: guix publish -u guix-publish -p 3000 -C 9 ...

SessionPID: 19444
ClientPID: 19419
ClientCommand: cuirass --cache-directory /var/cache/cuirass ...
LockHeld: /gnu/store/...-perl-ipc-cmd-0.96.lock
LockHeld: /gnu/store/...-python-six-bootstrap-1.11.0.lock
LockHeld: /gnu/store/...-libjpeg-turbo-2.0.0.lock
ChildPID: 20495
ChildCommand: guix offload x86_64-linux 7200 1 28800
ChildPID: 27733
ChildCommand: guix offload x86_64-linux 7200 1 28800
ChildPID: 27793
ChildCommand: guix offload x86_64-linux 7200 1 28800
```

Dans cet exemple, on voit que `guix-daemon` a trois clients directs : `guix environment`, `guix publish` et l'outil d'intégration continue Cuirass ; leur identifiant de processus (PID) est donné par le champ `ClientPID`. Le champ `SessionPID` fournit le PID du sous-processus `guix-daemon` de cette session particulière.

Le champs `LockHeld` montre quels éléments du dépôt sont actuellement verrouillés par cette session, ce qui correspond aux éléments du dépôt qui sont en train d'être construits ou d'être substitués (le champ `LockHeld` n'est pas montré si `guix processes` n'est pas lancé en root). Enfin, en regardant les champs `ChildPID` et `ChildCommand`, on comprend que ces trois constructions sont déchargées (voir Section 2.4.2 [Réglages du déchargement du démon], page 12).

<sup>3</sup> Les sessions distantes, lorsque `guix-daemon` est démarré avec `--listen` en spécifiant un point d'entrée TCP, ne sont *pas* listées.

La sortie est dans le format Recutils pour qu'on puisse utiliser la commande `recsel` pour sélectionner les sessions qui nous intéressent (voir Section "Selection Expressions" dans *GNU recutils manual*). Par exemple, la commande montre la ligne de commande et le PID du client qui effectue la construction d'un paquet Perl :

```
$ sudo guix processes | \
  recsel -p ClientPID,ClientCommand -e 'LockHeld ~ "perl"'
ClientPID: 19419
ClientCommand: cuirass --cache-directory /var/cache/cuirass ...
```

Des options supplémentaires sont listées ci-dessous.

`--format=format`

`-f format` Produire la sortie dans le *format* donné, parmi :

`recutils` L'option par défaut. Elle affiche un ensemble d'enregistrement `Session` `recutils` qui incluent chacun `ChildProcess` comme champ.

`normalized`

Normalise les enregistrements de sortie en ensembles d'enregistrement (voir Section "Record Sets" dans *GNU recutils manual*). La normalisation en ensembles d'enregistrement permet de faire des jointures entre plusieurs types d'enregistrement. L'exemple ci-dessous liste le PID de chaque `ChildProcess` et le PID associé pour `Session` qui lance `ChildProcess` où la `Session` a été démarrée avec `guix build`.

```
$ guix processes --format=normalized | \
  recsel \
  -j Session \
  -t ChildProcess \
  -p Session.PID,PID \
  -e 'Session.ClientCommand ~ "guix build"'
PID: 4435
Session_PID: 4278

PID: 4554
Session_PID: 4278

PID: 4646
Session_PID: 4278
```

## 11 Architectures externes

vous pouvez cibler des ordinateurs d'architectures CPU différentes lors de la production de paquets (voir Section 6.2 [Invoquer guix package], page 44), de lots applicatif (voir Section 8.3 [Invoquer guix pack], page 96) ou de systèmes complets (voir Section 12.15 [Invoquer guix system], page 584).

GNU Guix prend en charge deux mécanismes distincts pour cibler des architectures externes :

1. Le mécanisme de compilation croisée ([https://fr.wikipedia.org/wiki/Compilateur#Compilation\\_crois%C3%A9e](https://fr.wikipedia.org/wiki/Compilateur#Compilation_crois%C3%A9e)) classique.
2. Le mécanisme de construction natif qui consiste à construire en utilisant le jeu d'instruction CPU du système externe que vous ciblez. Cela nécessite souvent de l'émulation, avec le programme QEMU par exemple.

### 11.1 Compilation croisée

Les commandes qui prennent en charge la compilation croisée proposent les options `--list-target` et `--target`.

L'option `--list-targets` liste toutes les cibles prises en charge qui peuvent être passées en argument à `--target`.

```
$ guix build --list-targets
Les cibles disponibles sont :
```

```
- aarch64-linux-gnu
- arm-linux-gnueabi
- 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
```

Les cibles sont spécifiées comme des triplets GNU (voir Section “Specifying Target Triplets” dans *Autoconf*).

Ces triplets sont passés à GCC et aux autres compilateurs sous-jacent éventuellement impliqués dans la construction d'un paquet, d'une image système et de toute autre sortie de GNU Guix.

```
$ guix build --target=aarch64-linux-gnu hello
/gnu/store/9926by9qrx91ijkhw9ndgwp4bn24g9h-hello-2.12
```

```
$ file /gnu/store/9926by9qrx91ijkhw9ndgwp4bn24g9h-hello-2.12/bin/hello
/gnu/store/9926by9qrx91ijkhw9ndgwp4bn24g9h-hello-2.12/bin/hello: ELF
64-bit LSB executable, ARM aarch64 ...
```

Le principal avantage de la compilation croisée est qu'il n'y a pas de ralentissement par rapport à l'émulation avec QEMU. Il y a cependant de plus grands risques que certains paquets échouent à compiler de manière croisée car seuls quelques utilisateurs et utilisatrices utilisent ce mécanisme de manière intensive.

## 11.2 Constructions natives

Les commandes qui prennent en charge l'imitation d'un système spécifique proposent les options `--list-systems` et `--system`.

L'option `--list-systems` liste tous les systèmes pris en charge qui peuvent être passés en argument à `--system`.

```
$ guix build --list-systems
```

```
Les systèmes disponibles sont :
```

```
- 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/cc0km35s8x2z4pmwkrqjx46i8b1i3gm-hello-2.12
```

```
$ file /gnu/store/cc0km35s8x2z4pmwkrqjx46i8b1i3gm-hello-2.12/bin/hello
/gnu/store/cc0km35s8x2z4pmwkrqjx46i8b1i3gm-hello-2.12/bin/hello: ELF
32-bit LSB executable, Intel 80386 ...
```

Dans l'exemple ci-dessus, le système actuel est *x86\_64-linux*. Le paquet *hello* est cependant construit pour le système *i686-linux*.

C'est possible parce que le jeu d'instruction CPU *i686* est un sous-ensemble de *x86\_64*, d'où le fait que les binaires ciblant *i686* puissent tourner sur *x86\_64*.

Toujours dans le contexte de l'exemple précédent, si on choisit le système *aarch64-linux* que la commande `guix build --system=aarch64-linux hello` doit construire des dérivations, une étape supplémentaire est nécessaire.

Les binaires qui ciblent *aarch64-linux* ne peuvent pas tourner directement sur un système *x86\_64-linux*. Une couche d'émulation est requise. Le démon GNU Guix peut utiliser le mécanisme `binfmt_misc` ([https://en.wikipedia.org/wiki/Binfmt\\_misc](https://en.wikipedia.org/wiki/Binfmt_misc)) du noyau Linux pour cela. En résumé, le noyau Linux peut laisser l'exécution d'un binaire qui cible une plateforme externe, ici *aarch64-linux* à un programme en espace utilisateur, typiquement un émulateur.

Il existe un service qui configure QEMU en tant que moteur pour le mécanisme `binfmt_misc` (voir Section 12.9.29 [Services de virtualisation], page 504). Sur les distributions externes basées sur Debian, l'alternative consiste à installer le paquet `qemu-user-static`.

Si le mécanisme `binfmt_misc` n'est pas correctement mis en place, la construction échouera de cette façon :

```
$ guix build --system=armhf-linux hello --check
...
unsupported-platform /gnu/store/jjn969pijv7hff62025yxpfmtc8zy0aq0-hello-2.12.drv aarch64-linux
while setting up the build environment: a `aarch64-linux' is required to
build `/gnu/store/jjn969pijv7hff62025yxpfmtc8zy0aq0-hello-2.12.drv', but
I am a `x86_64-linux'...
```

alors que, avec le mécanisme `binfmt_misc` correctement lié à QEMU, on peut s'attendre à voir :

```
$ guix build --system=armhf-linux hello --check
/gnu/store/13xz4nghg39wpymivlwghy08yzj97hlj-hello-2.12
```

Le principal avantage de la construction native par rapport à la compilation croisée est que plus de paquets seront susceptibles de se construire correctement. Cependant cela a un prix : la compilation prise en charge par QEMU est *bien plus lente* que la compilation croisée, car chaque instruction doit être émulée.

La disponibilité des substituts pour l'architecture ciblée par l'option `--system` peut atténuer ce problème. Une autre manière de le contourner est d'installer GNU Guix sur une machine dont le CPU prend en charge le jeu d'instruction cible, et de la configurer en tant que machine de déchargement (voir Section 2.4.2 [Réglages du déchargement du démon], page 12).



## 12 Configuration du système

Guix System utilise un mécanisme de configuration du système cohérent. On veut dire par là que tous les aspects de la configuration globale du système — comme la disponibilité des services système, la configuration des fuseaux horaires, des paramètres linguistiques et des comptes utilisateurs — sont déclarés à un seul endroit. Une telle *configuration système* peut être *instanciée*, c'est-à-dire entrer en vigueur.

L'un des avantages de placer toute la configuration du système sous le contrôle de Guix est de permettre les mises à jour transactionnelles du système ce qui rend possible le fait de revenir en arrière à une instanciation précédent du système, si quelque chose se passait mal avec le nouveau (voir Section 6.1 [Fonctionnalités], page 43). Un autre avantage est de rendre facile la réplication de la même configuration sur plusieurs machines différentes ou à différents moments dans le temps, sans avoir à recourir à des outils d'administrations supplémentaires au-dessus des outils du système.

Cette section décrit ce mécanisme. Tout d'abord nous nous concentrons sur le point de vue de l'administrateur système en expliquant comment le système est configuré et instancié. Ensuite nous montrons comment ce mécanisme peut être étendu, par exemple pour supporter de nouveaux services systèmes.

### 12.1 Utiliser le système de configuration

Le système d'exploitation est configuré en fournissant une déclaration `operating-system` dans un fichier qui peut être passé à la commande `guix system` (voir Section 12.15 [Invoquer `guix system`], page 584). Une configuration simple, avec les services systèmes par défaut, le noyau Linux-Libre par défaut, un disque de RAM initial et un chargeur d'amorçage ressemble à ceci :

```
;; 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)))

```

Cet exemple devrait se comprendre de lui-même. Certains champs définis ci-dessus, comme `host-name` et `bootloader` sont obligatoires. D'autres comme `packages` et `services` peuvent être omis auquel cas ils ont une valeur par défaut.

Ci-dessous nous discutons des effets de certains des champs les plus importants (voir Section 12.2 [référence de `operating-system`], page 246, pour des détails sur tous les champs disponibles) et comment *instancier* le système d'exploitation avec `guix system`.

## Bootloader

Le champ `bootloader` décrit la méthode qui sera utilisée pour démarrer votre système. Les machines basées sur les processeurs Intel peuvent démarrer dans l'ancien mode BIOS, comme dans l'exemple au-dessus. Cependant, les machines plus récentes s'appuient sur

l'UEFI (*Unified Extensible Firmware Interface*) pour démarrer. Dans ce cas, le champ `bootloader` devrait contenir quelque chose comme cela :

```
(bootloader-configuration
  (bootloader grub-efi-bootloader)
  (targets '("/boot/efi")))
```

Voir Section 12.14 [Configuration du chargeur d'amorçage], page 578, pour plus d'informations sur les options de configuration disponibles.

## Paquets visibles sur tout le système

Le champ `packages` liste les paquets qui seront visibles sur tout le système, pour tous les comptes utilisateur-`rice-s` — c.-à-d. dans la variable d'environnement `PATH` de tous les utilisateur-`rice-s` — en plus des profils utilisateur-`rice-s` (voir Section 6.2 [Invoquer guix package], page 44). La variable `%base-packages` fournit tous les outils qu'on pourrait attendre pour les tâches de base de l'administrateur-`rice` et de l'utilisateur-`rice` — dont les GNU Core Utilities, les GNU Networking Utilities, l'éditeur de texte léger `mg`, `find`, `grep`, etc. L'exemple ci-dessus ajoute GNU Screen à ces paquets, récupéré depuis le module (`gnu packages screen`) (voir Section 9.1 [Modules de paquets], page 104). Vous pouvez utiliser la syntaxe (`list package output`) pour ajouter une sortie spécifique d'un paquet :

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

(operating-system
  ;; ...
  (packages (cons (list isc-bind "utils")
                  %base-packages)))
```

Se référer aux paquets par le nom de leur variable, comme `isc-bind` ci-dessus, a l'avantage d'être sans ambiguïté ; cela permet aussi de se rendre rapidement compte de coquilles quand on a des « variables non liées ». L'inconvénient est qu'on a besoin de savoir dans quel module est défini le paquet, et de modifier la ligne `use-package-modules` en conséquence. Pour éviter cela, on peut utiliser la procédure `specification->package` du module (`gnu packages`), qui renvoie le meilleur paquet pour un nom donné ou un nom et une version :

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

(operating-system
  ;; ...
  (packages (append (map specification->package
                      '("tcpdump" "htop" "gnupg@2.0"))
                  %base-packages)))
```

## Services systèmes

Le champ `services` liste les *services système* à rendre disponible lorsque le système démarre (voir Section 12.9 [Services], page 267). La déclaration `operating-system` au-dessus spécifie que, en plus des services de base, on veut que le démon `ssh` OpenSSH écoute sur le port 2222 (voir Section 12.9.5 [Services réseau], page 299). Sous le capot, `openssh-service-type` s'arrange pour que `sshd` soit lancé avec les bonnes options de la ligne de commande,

éventuellement en générant des fichiers de configuration (voir Section 12.18 [Définir des services], page 599).

Parfois, plutôt que d'utiliser les services de base tels-quels, on peut vouloir les personnaliser. Pour cela, utilisez `modify-services` (voir Section 12.18.3 [Référence de service], page 602) pour modifier la liste.

Par exemple, supposons que vous souhaitiez modifier `guix-daemon` et `Mingetty` (l'écran de connexion en console) dans la liste `%base-services` (voir Section 12.9.1 [Services de base], page 268). Pour cela, vous pouvez écrire ce qui suit dans votre déclaration de système d'exploitation :

```
(define %my-services
  ;; ma propre liste de services.
  (modify-services %base-services
    (guix-service-type config =>
      (guix-configuration
       (inherit config)
       ;; Récupérer les substituts depuis example.org.
       (substitute-urls
        (list "https://example.org/guix"
              "https://ci.guix.gnu.org"))))
    (mingetty-service-type config =>
      (mingetty-configuration
       (inherit config)
       ;; Connexion automatique en tant que « guest ».
       (auto-login "guest"))))

  (operating-system
   ;; ...
   (services %my-services))
```

Cela modifie la configuration — c.-à-d. les paramètres du service — de l'instance de `guix-service-type`, et de toutes les instances de `mingetty-service-type` dans la liste `%base-services` (voir Section “Connexion automatique à un TTY donné” dans *GNU Guix Cookbook*). Remarquez comment on fait cela : d'abord, on s'arrange pour que la configuration de départ soit liée à l'identifiant `config` dans `body` puis on écrit `body` pour qu'il s'évalue en la configuration désirée. En particulier, remarquez comment on utilise `inherit` pour créer une nouvelle configuration qui a les mêmes valeurs que l'ancienne configuration, avec seulement quelques modifications.

La configuration pour une utilisation de « bureau » typique, avec une partition racine chiffrée, a fichier d'échange sur la partition racine, le serveur d'affichage X11, GNOME et Xfce (les utilisateurs peuvent choisir l'environnement de bureau sur l'écran de connexion en appuyant sur *F1*), la gestion du réseau, la gestion de l'énergie, et bien plus, ressemblerait à ceci :

```
;; 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))

```

Un système graphique avec un choix de gestionnaires de fenêtres légers plutôt que des environnements de bureaux complets ressemblerait à cela :

```
;; 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 wmm
              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))

```

Cet exemple se réfère au système de fichier `/boot/efi` par son UUID, `1234-ABCD`. Remplacez cet UUID par le bon UUID de votre système, renvoyé par la commande `blkid`.

Voir Section 12.9.9 [Services de bureaux], page 347, pour la liste exacte des services fournis par `%desktop-services`. Voir Section 12.11 [Certificats X.509], page 572, pour des informations sur le paquet `nss-certs` utilisé ici.

Encore une fois, `%desktop-services` n'est qu'une liste d'objets `service`. Si vous voulez enlever des services de là, vous pouvez le faire avec des procédures pour les listes (voir Section "SRFI-1 Filtering and Partitioning" dans *GNU Guile Reference Manual*). Par exemple, l'expression suivante renvoie une liste qui contient tous les services dans `%desktop-services` sauf le service Avahi :

```

(remove (lambda (service)
         (eq? (service-kind service) avahi-service-type))
        %desktop-services)

```

Autrement, la macro `modify-services` peut être utilisée :

```

(modify-services %desktop-services
 (delete avahi-service-type))

```

## Instancier le système

En supposant que la déclaration `operating-system` est stockée dans le fichier `my-system-config.scm`, la commande `guix system reconfigure my-system-config.scm` instancie cette configuration et en fait l'entrée par défaut dans GRUB (voir Section 12.15 [Invoquer `guix system`], page 584).

**Remarque:** Nous recommandons que vous gardiez le fichier `my-system-config.scm` en lieu sûr et sous contrôle de version pour mieux tracer les changements de votre configuration.

Pour changer la configuration du système, on met normalement à jour ce fichier et on relance `guix system reconfigure`. On ne devrait jamais avoir à modifier de fichiers dans `/etc` ou à lancer des commandes qui modifient l'état du système comme `useradd` ou `grub-install`. En fait, vous devez les éviter parce que non seulement ça annulerait vos garanties,



mais ça empêcherait aussi de revenir à des versions précédents du système, si vous en avez besoin.

En parlant de revenir en arrière, à chaque fois que vous lancez `guix system reconfigure`, une nouvelle *génération* du système est créée — sans modifier ou supprimer les générations précédentes. Les anciennes générations du système ont une entrée dans le menu du chargeur d'amorçage, ce qui vous permet de démarrer dessus au cas où quelque chose se serait mal passé avec la dernière génération. C'est rassurant, non ? La commande `guix system list-generations` liste les générations du système disponibles sur le disque. Il est possible de revenir à une ancienne génération via les commandes `guix system roll-back` et `guix system switch-generation`.

Bien que la commande `guix system reconfigure` ne modifiera pas les générations précédentes, vous devez faire attention lorsque votre génération actuelle n'est pas la dernière (p. ex. après avoir invoqué `guix system roll-back`), puisque l'opération pourrait remplacer une génération suivante (voir Section 12.15 [Invoquer guix system], page 584).

## L'interface de programmation

Au niveau Scheme, la grosse déclaration `operating-system` est instanciée avec la procédure monadique suivante (voir Section 9.11 [La monade du dépôt], page 162) :

`operating-system-derivation os` [Procédure monadique]

Renvoie une dérivation qui construit `os`, un objet `operating-system` (voir Section 9.10 [Dérivations], page 159).

La sortie de la dérivation est un répertoire qui se réfère à tous les paquets et d'autres fichiers supports requis pour instancier `os`.

Cette procédure est fournie par le module (`gnu system`). Avec (`gnu services`) (voir Section 12.9 [Services], page 267), ce module contient les entrailles du système Guix. Ouvrez-le un jour !

## 12.2 Référence de operating-system

Cette section résume toutes les options disponibles dans les déclarations `operating-system` (voir Section 12.1 [Utiliser le système de configuration], page 238).

`operating-system` [Type de données]

C'est le type de données représentant une configuration d'un système d'exploitation. On veut dire par là toute la configuration globale du système, mais pas la configuration par utilisateur (voir Section 12.1 [Utiliser le système de configuration], page 238).

`kernel` (par défaut : `linux-libre`)

L'objet du paquet du système d'exploitation à utiliser<sup>1</sup>.

`hurd` (par défaut : `#f`)

L'objet du paquet du `hurd` à être lancé par le noyau. Lorsque ce champ est défini, produire un système d'exploitation GNU/Hurd. Dans ce cas,

<sup>1</sup> Actuellement, seul le noyau Linux-libre est entièrement pris en charge. L'utilisation de GNU mach avec le GNU Hurd est expérimentale et n'est disponible que lors de la construction d'une image disque de machine virtuelle.

`kernel` doit également être défini sur le paquet `gnumach` — le micro-noyau sur lequel tourne le Hurd.

**Attention:** Cette fonction est expérimentale et seulement prise en charge pour les images de disques.

`kernel-loadable-modules` (par défaut : `'()`)

Une liste d'objets (généralement des paquets) pour collecter les modules de noyau chargeables depuis - par exemple (`liste ddcci-driver-linux`).

`server-arguments` (par défaut : `%default-kernel-arguments`)

Liste de chaînes ou de gexps représentant des arguments supplémentaires à passer sur la ligne de commande du noyau — p. ex. (`"console=ttyS0"`).

`bootloader`

L'objet de configuration du chargeur d'amorçage. Voir Section 12.14 [Configuration du chargeur d'amorçage], page 578.

`label`

C'est l'étiquette (une chaîne de caractères) comme elle apparaît dans l'entrée du menu du chargeur d'amorçage. L'étiquette par défaut inclus le nom du noyau et sa version.

`keyboard-layout` (par défaut : `#f`)

Ce champ spécifie la disposition du clavier à utiliser dans la console. Il peut être soit `#f`, auquel cas la disposition par défaut est utilisée (habituellement anglais américain), ou un enregistrement `<keyboard-layout>`. Voir Section 12.7 [Disposition du clavier], page 263, pour plus d'informations.

Cette disposition du clavier est effective dès que le noyau démarre. Par exemple, c'est la disposition du clavier effective lorsque vous saisissez la phrase de passe de votre système de fichier racine sur une partition utilisant `luks-device-mapping` (voir Section 12.4 [Périphériques mappés], page 256).

**Remarque:** Cela ne spécifie *pas* la disposition clavier utilisée par le chargeur d'amorçage, ni celle utilisée par le serveur d'affichage graphique. Voir Section 12.14 [Configuration du chargeur d'amorçage], page 578, pour plus d'information sur la manière de spécifier la disposition du clavier pour le chargeur d'amorçage. Voir Section 12.9.7 [Système de fenêtrage X], page 325, pour plus d'informations sur la manière de spécifier la disposition du clavier utilisée par le système de fenêtrage X.

`initrd-modules` (par défaut : `%base-initrd-modules`)

La liste des modules du noyau linux requis dans l'image disque de RAM initiale. Voir Section 12.13 [Disque de RAM initial], page 575.

`initrd` (par défaut : `base-initrd`)

Une procédure qui renvoie un disque de RAM initial pour le noyau Linux. Ce champ est fournit pour pouvoir personnaliser son système à bas-niveau



`issue` (par défaut : `%default-issue`)

Une chaîne qui dénote le contenu du fichier `/etc/issue` qui est affiché lorsqu'un utilisateur se connecte sur la console.

`packages` (par défaut : `%base-packages`)

Une liste de paquets à installer dans le profil global, qui est accessible à partir de `/run/current-system/profile`. Chaque élément est soit une variable de paquet, soit un tuple de paquet/sortie. Voici un exemple simple des deux :

```
(cons* git ; la sortie "out" par défaut
      (list git "send-email") ; une autre sortie de git
      %base-packages) ; l'ensemble par défaut
```

L'ensemble par défaut contient les utilitaires de base et c'est une bonne pratique d'installer les utilitaires non essentiels dans les profils utilisateurs (voir Section 6.2 [Invoquer guix package], page 44).

`timezone` (par défaut : `"Etc/UTC"`)

Une chaîne identifiant un fuseau horaire — p. ex. `"Europe/Paris"`.

Vous pouvez lancer la commande `tzselect` pour trouver le fuseau horaire correspondant à votre région. Si vous choisissez un nom de fuseau horaire invalide, `guix system` échouera.

`locale` (par défaut : `"en_US.utf8"`)

Le nom du paramètre régional par défaut (voir Section “Locale Names” dans *The GNU C Library Reference Manual*). Voir Section 12.8 [Régionalisation], page 265, pour plus d'informations.

`locale-definitions` (par défaut : `%default-locale-definitions`)

La liste des définitions de locales à compiler et qui devraient être utilisées à l'exécution. Voir Section 12.8 [Régionalisation], page 265.

`locale-libcs` (par défaut : `(list glibc)`)

La liste des paquets GNU libc dont les données des paramètres linguistiques sont utilisées pour construire les définitions des paramètres linguistiques. Voir Section 12.8 [Régionalisation], page 265, pour des considérations sur la compatibilité qui justifient cette option.

`name-service-switch` (par défaut : `%default-nss`)

La configuration de NSS de la libc (name service switch) — un objet `<name-service-switch>`. Voir Section 12.12 [Name Service Switch], page 573, pour des détails.

`services` (par défaut : `%base-services`)

Une liste d'objets services qui dénotent les services du système. Voir Section 12.9 [Services], page 267.

`essential-services` (par défaut : `...`)

La liste des « services essentiels » — c.-à-d. les services comme des instance de `system-service-type` et `host-name-service-type` (voir Section 12.18.3 [Référence de service], page 602), qui sont dérivés de la

définition du système d'exploitation lui-même. En tant qu'utilisateur vous ne devriez *jamais* toucher à ce champ.

**pam-services** (par défaut : `(base-pam-services)`)

Services PAM (*pluggable authentication module*) Linux.

**setuid-programs** (par défaut : `%setuid-programs`)

Liste de `<setuid-program>`. Voir Section 12.10 [Programmes setuid], page 571, pour plus d'informations.

**sudoers-file** (par défaut : `%sudoers-specification`)

Le contenu du fichier `/etc/sudoers` comme un objet simili-fichier (voir Section 9.12 [G-Expressions], page 166).

Ce fichier spécifier quels utilisateurs peuvent utiliser la commande `sudo`, ce qu'ils ont le droit de faire, et quels privilèges ils peuvent gagner. La valeur par défaut est que seul `root` et les membres du groupe `wheel` peuvent utiliser `sudo`.

**this-operating-system** [Type de données]

Lorsqu'il est utilisée dans la *portée lexicale* de la définition d'un du système d'exploitation, cet identifiant est résolu comme étant le système d'exploitation définit.

L'exemple ci-dessous montre le référencement au système d'exploitation définit dans la définition du champ `label` :

```
(use-modules (gnu) (guix))
```

```
(operating-system
```

```
;; ...
```

```
(label (package-full-name
```

```
  (operating-system-kernel this-operating-system))))
```

C'est une erreur que de se référer à `this-operating-system` en dehors de la définition d'un système d'exploitation.

## 12.3 Systèmes de fichiers

La liste des systèmes de fichiers à monter est spécifiée dans le champ `file-systems` de la déclaration de système d'exploitation (voir Section 12.1 [Utiliser le système de configuration], page 238). Chaque système de fichier est déclaré avec la forme `file-system`, comme ceci :

```
(file-system
  (mount-point "/home")
  (device "/dev/sda3")
  (type "ext4"))
```

Comme d'habitude, certains de ces champs sont obligatoire — comme le montre l'exemple au-dessus — alors que d'autres peuvent être omis. Ils sont décrits plus bas.

**file-system** [Type de données]

Les objets de ce type représentent des systèmes de fichiers à monter. Ils contiennent les membres suivants :

**type** C'est une chaîne de caractères spécifiant le type de système de fichier — p. ex. "ext4".

**mount-point** Désigne l'emplacement où le système de fichier sera monté.

**device** Ce champ nomme le système de fichier « source ». il peut être l'une de ces trois choses : une étiquette de système de fichiers, un UUID de système de fichier ou le nom d'un nœud dans `/dev`. Les étiquettes et les UUID offrent une manière de se référer à des systèmes de fichiers sans avoir à coder en dur le nom de périphérique<sup>2</sup>.

Les étiquettes de systèmes de fichiers sont créés avec la procédure `file-system-label`, les UUID avec `uuid` et les nœuds de `/dev` sont de simples chaînes de caractères. Voici un exemple d'un système de fichiers référencé par son étiquette, donnée par la commande `e2label` :

```
(file-system
  (mount-point "/home")
  (type "ext4")
  (device (file-system-label "my-home")))
```

Les UUID sont convertis à partir de leur représentation en chaîne de caractères (montrée par la commande `tune2fs -l`) en utilisant la forme `uuid`<sup>3</sup>, comme ceci :

```
(file-system
  (mount-point "/home")
  (type "ext4")
  (device (uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")))
```

Lorsque la source d'un système de fichiers est un périphérique mappé (voir Section 12.4 [Périphériques mappés], page 256), son champ `device` doit se référer au nom du périphérique mappé — p. ex. `"/dev/mapper/root-partition"`. Cela est requis pour que le système sache que monter ce système de fichier dépend de la présence du périphérique mappé correspondant.

**flags** (par défaut : '() )

C'est une liste de symboles qui désignent des drapeaux de montage. Les drapeaux reconnus sont `read-only`, `bind-mount`, `no-dev` (interdit l'accès aux fichiers spéciaux), `no-suid` (ignore les bits `setuid` et `setgid`), `no-atime` (ne met pas à jour les heures d'accès aux fichiers), `no-diratime` (pareil mais pour les répertoires uniquement), `strict-atime` (met à jour les heures d'accès aux fichiers), `lazy-time` (ne met à jour les heures que dans la version mémoire des inœuds de fichiers), `no-exec` (interdit l'exécution

<sup>2</sup> Remarquez que, s'il est tentant d'utiliser `/dev/disk/by-uuid` et autres chemins similaires pour obtenir le même résultat, ce n'est pas recommandé : ces nœuds de périphériques spéciaux sont créés par le démon `udev` et peuvent ne pas être disponibles au moment de monter le périphérique.

<sup>3</sup> La forme `uuid` s'attend à des UUID sur 16 octets définis dans la RFC 4122 (<https://tools.ietf.org/html/rfc4122>). C'est la forme des UUID utilisées par la famille de systèmes de fichiers ext2 et d'autres, mais ce n'est pas le même type d'UUID que ceux qui se trouvent sur les systèmes de fichiers FAT par exemple

de programmes), et `shared` (partage le montage). Voir Section “Mount-Unmount-Remount” dans *The GNU C Library Reference Manual*, pour plus d’informations sur ces drapeaux.

`options` (par défaut : `#f`)

C’est soit `#f`, soit une chaîne qui dénote les options de montage passées au pilote de système de fichiers. Voir Section “Mount-Unmount-Remount” dans *le manuel de référence de la bibliothèque C de GNU* pour plus de détails.

Lancez `man 8 mount` pour voir les options des divers systèmes de fichiers, mais sachez que les « options de montage » indépendantes du système de fichiers sont en fait des drapeaux, et appartiennent au champ `flags` décrit plus haut.

Les procédures `file-system-options->alist` et `alist->file-system-options` de (`gnu system file-systems`) peuvent être utilisées pour convertir des options de systèmes de fichiers données sous forme de liste d’association en représentation de chaîne de caractères, et vice-versa.

`mount?` (par défaut : `#t`)

Cette valeur indique s’il faut monter automatiquement le système de fichier au démarrage du système. Lorsque la valeur est `#f`, le système de fichier reçoit une entrée dans `/etc/fstab` (lue par la commande `mount`) mais n’est pas monté automatiquement.

`needed-for-boot?` (par défaut : `#f`)

Cette valeur booléenne indique si le système de fichier est nécessaire au démarrage. Si c’est vrai alors le système de fichier est monté au chargement du disque de RAM initial. C’est toujours le cas par exemple du système de fichiers racine.

`check?` (par défaut : `#t`)

Cette valeur booléenne indique si le système de fichier devrait être vérifié avant de le monter. Comment et quand cela se produit peut être réglé plus en détail avec les options suivantes.

`skip-check-if-clean?` (par défaut : `#t`)

Lorsqu’il a la valeur vrai, ce booléen indique qu’une vérification de système de fichier initiée par `check?` peut terminer prématurément si le système de fichier est marqué comme « propre », ce qui signifie qu’il a préalablement été démonté correctement and ne devrait contenir aucune erreur.

Lui affecter la valeur faux imposera toujours une vérification de cohérence complète si `check?` est à vrai. Cela peut prendre très longtemps et n’est pas recommandé sur des systèmes sains—en fait, cela pourrait même réduire la fiabilité !

À l’inverse, certains systèmes de fichiers primitifs comme `fat` ne gardent pas trace des arrêts propres et exécuteront une vérification complète sans tenir compte de la valeur de cette option.

`repair` (par défaut : `'preen'`)

Quand `check?` trouve des erreurs, il peut (essayer de) les réparer pour poursuivre le démarrage. Cette option contrôle dans quel cas et de quelle manière appliquer cette stratégie.

À la valeur faux, il essaiera de pas modifier le système de fichier du tout. Vérifier certains systèmes de fichiers comme `jfs` peut quand même provoquer des écritures sur le périphérique pour rejouer le journal. Aucune réparation ne sera tentée.

Si la valeur `#t` est renseignée, il essaiera de récupérer toute erreur qu'il trouvera et supposera qu'il peut répondre « oui » à toutes les questions. Cela corrigera la plupart des erreurs, mais peut s'avérer risqué.

Si elle vaut `'preen'`, il ne réparera que les erreurs qui peuvent être corrigées sans risque en l'absence d'intervention humaine. Ce que cela signifie est laissé à l'appréciation des personnes qui développent chaque système de fichier et peut être équivalent à « aucune » ou « toutes ».

`create-mount-point?` (par défaut : `#f`)

Lorsque cette valeur est vraie, le point de montage est créé s'il n'existe pas déjà.

`mount-may-fail?` (par défaut : `#t`)

Lorsque cela est vrai, indique que le montage de ce système de fichiers peut échouer, mais cela ne doit pas être considéré comme une erreur. Ceci est utile dans des cas inhabituels ; un exemple de ceci est `efivarfs`, un système de fichiers qui ne peut être monté que sur des systèmes EFI/UEFI.

`dependencies` (par défaut : `'()`)

C'est une liste d'objets `<file-system>` ou `<mapped-device>` qui représentent les systèmes de fichiers qui doivent être montés ou les périphériques mappés qui doivent être ouverts avant (et monté ou fermés après) celui-ci.

Par exemple, considérons une hiérarchie de montage : `/sys/fs/cgroup` est une dépendance de `/sys/fs/cgroup/cpu` et `/sys/fs/cgroup/memory`.

Un autre exemple est un système de fichier qui dépend d'un périphérique mappé, par exemple pour une partition chiffrée (voir Section 12.4 [Périphériques mappés], page 256).

`file-system-label` *str* [Procédure Scheme]

Cette procédure renvoie un label de système de fichiers opaque à partir de *str*, une chaîne de caractères :

```
(file-system-label "home")
⇒ #<file-system-label "home">
```

Les étiquettes de systèmes de fichiers sont utilisées pour faire référence aux systèmes de fichiers par étiquette plutôt que par nom de dispositif. Voir ci-dessus pour des exemples.



Le module (`gnu system file-systems`) exporte les variables utiles suivantes.

`%base-file-systems` [Variable Scheme]

Ce sont les systèmes de fichiers essentiels qui sont requis sur les systèmes normaux, comme `%pseudo-terminal-file-system` et `%immutable-store` (voir plus bas). Les déclarations de systèmes d'exploitation devraient au moins les contenir.

`%pseudo-terminal-file-system` [Variable Scheme]

C'est le système de fichier monté sur `/dev/pts`. Il supporte les *pseudo-terminaux* créés via `openpty` et les fonctions similaires (voir Section "Pseudo-Terminals" dans *The GNU C Library Reference Manual*). Les pseudo-terminaux sont utilisés par les émulateurs de terminaux comme `xterm`.

`%shared-memory-file-system` [Variable Scheme]

Ce système de fichier est monté dans `/dev/shm` et est utilisé pour le partage de mémoire entre processus (voir Section "Memory-mapped I/O" dans *The GNU C Library Reference Manual*).

`%immutable-store` [Variable Scheme]

Ce système de fichiers effectue un « montage lié » en lecture-seule de `/gnu/store`, ce qui en fait un répertoire en lecture-seule pour tous les utilisateurs dont `root`. Cela évite que des logiciels qui tournent en `root` ou des administrateurs systèmes ne modifient accidentellement le dépôt.

Le démon lui-même est toujours capable d'écrire dans le dépôt : il est remonté en lecture-écriture dans son propre « espace de nom ».

`%binary-format-file-system` [Variable Scheme]

Le système de fichiers `binfmt_misc`, qui permet de gérer n'importe quel type de fichiers exécutables à déléguer en espace utilisateur. Cela demande que le module du noyau `binfmt.ko` soit chargé.

`%fuse-control-file-system` [Variable Scheme]

Le système de fichiers `fusectl`, qui permet à des utilisateurs non privilégiés de monter et de démonter des systèmes de fichiers FUSE en espace utilisateur. Cela requiert que le module du noyau `fuse.ko` soit chargé.

Le module (`gnu system uuid`) fournit les outils pour traiter les « identifiants uniques » de système de fichier (UUIDs).

`uuid type [value]` [Procédure Scheme]

Renvoie un objet UUID (identifiant unique) opaque du *type* donné (un symbole) en analysant *str* (une chaîne) :

```
(uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")
⇒ #<uuid> type: dce bv: ...>
```

```
(uuid "1234-ABCD" 'fat)
⇒ #<uuid> type: fat bv: ...>
```

*type* peut être l'un des codes suivants : `dce`, `iso9660`, `fat`, `ntfs`, ou l'un des synonymes les plus courants.

Les UUIDs sont un autre moyen de faire référence sans ambiguïté aux systèmes de fichiers dans la configuration du système d'exploitation. Voir les exemples ci-dessus.

### 12.3.1 Système de fichier Btrfs

Btrfs a des particularités, comme les sous-volumes, qui méritent d'être expliquées plus en détail. La section suivante tente de couvrir les utilisations de base ainsi que les utilisations complexes d'un système de fichiers Btrfs avec le système Guix.

Dans son usage le plus simple, un système de fichier Btrfs peut être décrit, par exemple, par :

```
(file-system
  (mount-point "/home")
  (type "btrfs")
  (device (file-system-label "my-home")))
```

L'exemple ci-dessous est plus complexe, car il utilise un sous-volume de Btrfs, nommé `rootfs`. Le système de fichiers Btrfs parent est appelé `my-btrfs-pool`, et se trouve sur un périphérique crypté (d'où la dépendance à l'égard de `mapped-devices`) :

```
(file-system
  (device (file-system-label "my-btrfs-pool"))
  (mount-point "/")
  (type "btrfs")
  (options "subvol=rootfs")
  (dependencies mapped-devices))
```

Certains chargeurs d'amorçage, par exemple GRUB, ne montent une partition Btrfs qu'à son niveau supérieur au début du démarrage, et s'appuient sur leur configuration pour se référer au chemin correct du sous-volume à l'intérieur de ce niveau supérieur. Les chargeurs d'amorçage fonctionnant de cette manière produisent généralement leur configuration sur un système en cours d'exécution où les partitions Btrfs sont déjà montées et où les informations sur les sous-volumes sont facilement disponibles. Par exemple, `grub-mkconfig`, la commande de générateur de configuration livrée avec GRUB, lit `/proc/self/mountinfo` pour déterminer le chemin de niveau supérieur d'un sous-volume.

Guix System produit une configuration de bootloader en utilisant la configuration du système d'exploitation comme seule entrée ; il est donc nécessaire d'extraire le nom du sous-volume sur lequel vit `/gnu/store` (s'il existe) de cette configuration du système d'exploitation. Pour mieux illustrer cela, considérons un sous-volume nommé "rootfs" qui contient les données du système de fichiers racine. Dans une telle situation, le chargeur de démarrage GRUB ne verrait que le niveau supérieur de la partition Btrfs de la racine, par exemple :

```
/                (niveau supérieur)
  rootfs         (répertoire des sous-volumes)
    gnu          (répertoire normal)
      store     (répertoire normal)
  [...]

```

Ainsi, le nom du sous-volume doit être précédé du chemin `/gnu/store` du noyau, des binaires `initrd` et de tout autre fichier mentionné dans la configuration GRUB qui doit être trouvé au début du démarrage.

L'exemple suivant montre une hiérarchie imbriquée de sous-volumes et de répertoires :

```

/                               (niveau supérieur)
rootfs                           (sous-volume)
  gnu                             (répertoire normal)
    store                         (sous-volume)
[...]
```

Ce scénario fonctionnerait sans monter le sous-volume « store ». Le montage de "rootfs" est suffisant, puisque le nom du sous-volume correspond à son point de montage prévu dans la hiérarchie du système de fichiers. Une autre solution consiste à faire référence au sous-volume « store » en définissant l'option `subvol` soit `/rootfs/gnu/store` soit `rootfs/gnu/store`.

Enfin, un exemple plus élaboré de sous-volumes imbriqués :

```

/                               (niveau supérieur)
root-snapshots                   (sous-volume)
  root-current                   (sous-volume)
    guix-store                   (sous-volume)
[...]
```

Ici, le sous-volume « guix-store » ne correspond pas à son point de montage prévu, il est donc nécessaire de le monter. Le sous-volume doit être entièrement spécifié, en passant son nom de fichier à l'option `subvol`. Par exemple, le sous-volume « guix-store » peut être monté sur `/gnu/store` en utilisant une déclaration de système de fichiers telle que :

```

(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 Périphériques mappés

Le noyau Linux a une notion de *mappage de périphériques* : un périphérique bloc, comme une partition sur un disque dur, peut être *mappé* sur un autre périphérique, typiquement dans `/dev/mapper`, avec des calculs supplémentaires sur les données qui naviguent entre les deux<sup>4</sup>. Un exemple typique est le mappage de périphériques chiffrés : toutes les écritures sont sur le périphérique mappé sont chiffrées, toutes les lectures déchiffrées, de manière transparente. Guix étend cette notion en considérant que tout périphérique ou ensemble de périphériques qui sont *transformés* d'une certaine manière créent un nouveau périphérique ; par exemple, les périphériques RAID sont obtenus en *assemblant* plusieurs autres périphériques, comme des disque ou des partitions, en un nouveau périphérique en tant qu'unique partition.

Les périphériques mappés sont déclarés avec la forme `mapped-device`, définie comme suit ; par exemple, voir ci-dessous.

<sup>4</sup> Remarquez que le Hurd ne fait pas de différence entre le concept de « périphérique mappé » et celle d'un système de fichiers : les deux correspondent à la *traduction* des opérations d'entrée-sortie faites sur un fichier en des opérations sur ce qui le contient. Ainsi, le Hurd implémente les périphériques mappés, comme les systèmes de fichiers, avec le mécanisme des *traducteurs* générique (voir Section « Translators » dans *The GNU Hurd Reference Manual*).

**mapped-device** [Type de données]

Les objets de ce type représentent des mappages de périphériques qui seront effectués au démarrage du système.

**source** C'est soit une chaîne qui spécifie le nom d'un périphérique bloc à projeter, comme `"/dev/sda3"`, soit une liste de plusieurs périphériques à assembler pour en créer un nouveau. Dans le cas de LVM, c'est une chaîne spécifiant le nom du groupe de volume à projeter.

**target** Cette chaîne spécifie le nom du périphérique mappé qui en résulte. Pour les mappeurs noyaux comme les périphériques chiffrés de type `luks-device-mapping`, spécifier `"ma-partition"` crée le périphérique `"/dev/mapper/ma-partition"`. Pour les périphériques RAID de type `raid-device-mapping`, il faut donner le nom complet comme `"/dev/md0"`. Les volumes logiques LVM de type `lvm-device-mapping` doivent être spécifiés comme étant `"VGNAME-LVNAME"`.

**targets** Cette liste de chaînes spécifie les noms des périphériques projetés s'il y en a plusieurs. Le format est identique à *target*.

**type** Ce doit être un objet `mapped-device-kind`, qui spécifie comment *source* est mappés sur *target*.

**luks-device-mapping** [Variable Scheme]

Cela définit les périphériques blocs chiffrés en LUKS avec `cryptsetup` du paquet du même nom. Elle s'appuie sur le module du noyau Linux `dm-crypt`.

**raid-device-mapping** [Variable Scheme]

Cela définit un périphérique RAID qui est assemblé avec la commande `mdadm` du paquet du même nom. Elle nécessite un module noyau Linux approprié pour le niveau RAID chargé, comme `raid456` pour RAID-4, RAID-5 et RAID-6 ou `raid10` pour RAID-10.

**lvm-device-mapping** [Variable Scheme]

Cela définit un ou plusieurs volumes logiques pour le Gestionnaire de Volume Logique (LVM) (<https://www.sourceware.org/lvm2/>) de Linux. Le groupe de volume est activé par la commande `vgchange` du paquet `lvm2`.

L'exemple suivant spécifie un mappage de `/dev/sda3` vers `/dev/mapper/home` avec LUKS — Linux Unified Key Setup (<https://gitlab.com/cryptsetup/cryptsetup>), un mécanisme standard pour chiffrer les disques. Le périphérique `/dev/mapper/home` peut ensuite être utilisé comme `device` d'une déclaration `file-system` (voir Section 12.3 [Systèmes de fichiers], page 250).

```
(mapped-device
 (source "/dev/sda3")
 (target "home")
 (type luks-device-mapping))
```

Autrement, pour devenir indépendant du numéro de périphérique, on peut obtenir l'UUID LUKS (*l'identifiant unique*) du périphérique source avec une commande comme :

```
cryptsetup luksUUID /dev/sda3
```

et l'utiliser ainsi :

```
(mapped-device
 (source (uuid "cb67fc72-0d54-4c88-9d4b-b225f30b0f44"))
 (target "home")
 (type luks-device-mapping))
```

Il est aussi désirable de chiffrer l'espace d'échange, puisque l'espace d'échange peut contenir des données sensibles. Une manière de faire cela est d'utiliser un fichier d'échange dans un système de fichiers sur un périphérique mappé avec un chiffrement LUKS. De cette manière, le fichier d'échange est chiffré parce que tout le périphérique est chiffré. Voir Section 12.5 [Espace d'échange], page 258, ou Voir Section 3.4 [Disk Partitioning], page 28, pour un exemple.

Un périphérique RAID formé des partitions `/dev/sda1` et `/dev/sdb1` peut être déclaré ainsi :

```
(mapped-device
 (source (list "/dev/sda1" "/dev/sdb1"))
 (target "/dev/md0")
 (type raid-device-mapping))
```

Le périphérique `/dev/md0` peut ensuite être utilisé comme `device` d'une déclaration `file-system` (voir Section 12.3 [Systèmes de fichiers], page 250). Remarquez que le niveau de RAID n'a pas besoin d'être donné ; il est choisi pendant la création initiale du périphérique RAID et est ensuite déterminé automatiquement.

Les volumes logiques LVM « alpha » et « beta » du groupe de volumes « vg0 » se déclarent de cette façon :

```
(mapped-device
 (source "vg0")
 (targets (list "vg0-alpha" "vg0-beta"))
 (type lvm-device-mapping))
```

Les périphériques `/dev/mapper/vg0-alpha` et `/dev/mapper/vg0-beta` sont maintenant utilisables comme `device` d'une déclaration `file-system` (voir Section 12.3 [Systèmes de fichiers], page 250).

## 12.5 Espace d'échange

Un espace d'échange est un espace disque spécifiquement dédié à la pagination mémoire : le processus en charge de la gestion de la mémoire (le noyau Linux ou le gestionnaire par défaut du Hurd) peut décider que certaines pages mémoire stockées en RAM qui appartiennent à un processus en cours d'exécution mais qui sont inutilisées devraient plutôt être stockées sur disque. Il les décharge de la RAM pour libérer la précieuse mémoire rapide, et les écrit sur l'espace d'échange. Si le programme essaye d'accéder à la page, le processus de gestion de la mémoire la charge à nouveau dans la mémoire pour que le programme puisse l'utiliser.

Une idée fausse répandue sur l'espace d'échange est qu'il n'est utile que lorsque seule une petite quantité de RAM est disponible sur le système. En réalité, vous devriez savoir que les noyaux utilisent souvent toute la RAM disponible pour le cache des accès disques pour rendre les opérations d'entrée-sortie plus rapides. En conséquence, déplacer des pages mémoire inutilisées augmentera la quantité de RAM disponible pour ce cache.

Pour une description plus complète de la gestion de la mémoire du point de vue d'un noyau monolithique, Voir Section "Memory Concepts" dans *le manuel de référence de la bibliothèque C de GNU*.

Le noyau Linux prend en charge les partitions d'échanges et les fichiers d'échange : le premier utilise une partition disque complète pour la pagination, alors que le second utilise un fichier sur un système de fichiers pour cela (le pilote du système de fichiers doit le prendre en charge). Pour une configuration comparable, les deux ont les mêmes performances, donc vous devriez considérer la facilité d'utilisation pour vous décider entre les deux. Les partitions sont « plus simples » et n'ont pas besoin de la prise en charge dans le système de fichiers, mais ont besoin d'être allouées au formatage du disque (en oubliant les volumes logiques), alors que les fichiers peuvent être alloués et désalloués à n'importe quel moment.

Remarquez que l'espace d'échange n'est pas remis à zéro à l'extinction donc les données sensibles (comme les mots de passe) peuvent y rester si elles y ont été déplacées. De fait, vous devriez faire résider votre espace d'échange sur un périphérique chiffré (voir Section 12.4 [Périphériques mappés], page 256).

**swap-space** [Type de données]

Les objets de ce type représentent des espaces d'échange. Ils contiennent les membres suivants :

**target** Le périphérique ou le fichier à utiliser, soit un UUID, soit un **file-system-label**, soit une chaîne, comme dans la définition d'un **file-system** (voir Section 12.3 [Systèmes de fichiers], page 250).

**dependencies** (par défaut : '()')

Une liste de d'objets **file-system** ou **mapped-device**, dont dépend la disponibilité de l'espace. Remarquez que, tout comme pour les objets **file-system**, les dépendances qui sont requises au démarrage et montées au plus tôt en espace utilisateur ne sont pas gérées par le Shepherd, et sont donc automatiquement retirées pour vous.

**priority** (par défaut : #f)

Seulement pris en charge par le noyau Linux. Soit #f pour désactive la priorité de l'espace d'échange, soit un entier entre 0 et 32767. Le noyau utilisera d'abord les espaces d'échange avec la plus grande priorité, et utilisera les espaces d'échange de même priorité à tour de rôle. Le noyau utilisera les espaces d'échange sans priorité après les espaces prioritaires, et dans l'ordre dans lequel ils sont apparus (donc pas à tour de rôle).

**discard?** (par défaut : #f)

Uniquement pris en charge par le noyau Linux. Lorsque la valeur est vraie, le noyau notifiera le contrôleur de disque des pages supprimées, par exemple avec l'opération TRIM sur les disques SSD.

Voici quelques exemples :

```
(swap-space (target (uuid "4dab5feb-d176-45de-b287-9b0a6e4c01cb")))
```

Utilise la partition de swap avec l'UUID donné. Vous pouvez voir l'UUID d'une partition de swap Linux en lançant `swaplabel device`, où `device` est le nom de fichier `/dev` de cette partition.

```
(swap-space
  (target (file-system-label "swap"))
  (dependencies mapped-device))
```

Utilise la partition avec l'étiquette `swap`, qui peut être trouvée après que tous les périphériques mappés *mapped-devices* ont été ouverts. De nouveau, la commande `swaplabel` vous permet de voir et de changer l'étiquette d'une partition swap de Linux.

Voici un exemple plus complet de la partie `file-systems` d'une déclaration `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))))))
```

utilise le fichier `/btrfs/swapfile` comme espace d'échange, ce qui dépend du système de fichiers monté sur `/btrfs`. Remarquez comment nous utilisons la fonction `filter` de Guile pour élégamment sélectionner le système de fichiers !

## 12.6 Comptes utilisateurs

Les comptes utilisateurs et les groupes sont gérés entièrement par la déclaration `operating-system`. Ils sont spécifiés avec les formes `user-account` et `user-group` :

```
(user-account
  (name "alice")
  (group "users")
  (supplementary-groups ('("wheel"      ;permet d'utiliser sudo, etc.
                          "audio"      ;carte son
                          "video"      ;périphériques réseaux comme les webcams
                          "cdrom")) ;le bon vieux CD-ROM
  (comment "Bob's sister"))
```

Voici un compte d'utilisateur `rice` qui utilise un shell différent et un répertoire d'accueil personnalisé (le répertoire par défaut serait `/home/bob`) :

```
(user-account
  (name "bob")
  (group "users")
  (comment "Alice's bro"))
```

```
(shell (file-append zsh "/bin/zsh"))
(home-directory "/home/robert"))
```

Lors du démarrage ou à la fin de `guix system reconfigure`, le système s'assure que seuls les comptes utilisateurs et les groupes spécifiés dans la déclaration `operating-system` existent, et avec les propriétés spécifiées. Ainsi, les modifications ou les créations de comptes ou de groupes effectuées directement en invoquant des commandes comme `useradd` sont perdues à la reconfiguration ou au redémarrage. Cela permet de s'assurer que le système reste exactement tel que déclaré.

**user-account** [Type de données]

Les objets de ce type représentent les comptes utilisateurs. Les membres suivants peuvent être spécifiés :

**name** Le nom du compte utilisateur.

**group** C'est le nom (une chaîne) ou un identifiant (un nombre) du groupe utilisateur auquel ce compte appartient.

**supplementary-groups** (par défaut : '() )  
Éventuellement, cela peut être définie comme une liste de noms de groupes auxquels ce compte appartient.

**uid** (par défaut : #f)  
C'est l'ID utilisateur de ce compte (un nombre) ou #f. Dans ce dernier cas, le nombre est choisi automatiquement par le système à la création du compte.

**comment** (par défaut : "")  
Un commentaire à propos du compte, comme le nom complet de l'utilisateur.  
Remarquez que, pour les compte non-systèmes, vous êtes libres de changer votre nom réel tel qu'il apparaît dans `/etc/passwd` avec la commande `chfn`. Lorsque vous faites cela, votre choix prévaut sur le choix de l'administrateur ou l'administratrice système ; la reconfiguration ne change *pas* votre nom.

**home-directory**  
C'est le nom du répertoire personnel du compte.

**create-home-directory?** (par défaut : #t)  
Indique si le répertoire personnel du compte devrait être créé s'il n'existe pas déjà.

**shell** (par défaut : Bash)  
C'est une G-expression qui désigne le nom de fichier d'un programme utilisé comme shell (voir Section 9.12 [G-Expressions], page 166). Par exemple, vous pourriez vous référer à l'exécutable Bash comme ceci :

```
(file-append bash "/bin/bash")
```

... et à l'exécutable Zsh comme ceci :

```
(file-append zsh "/bin/zsh")
```



`system?` (par défaut : `#f`)

C'est une valeur booléenne qui indique si le compte est un compte « système ». Les comptes systèmes sont parfois traités à part ; par exemple, les gestionnaires de connexion graphiques ne les liste pas.

`password` (par défaut : `#f`)

Vous laisseriez normalement ce champ à `#f` et initialiseriez les mots de passe utilisateurs en tant que `root` avec la commande `passwd`, puis laisseriez l'utilisateur le changer avec `passwd`. Les mots de passes définis avec `passwd` sont bien sûr préservés après redémarrage et reconfiguration.

Si vous voulez *vraiment* définir un mot de passe pour un compte, alors ce champ doit contenir le mot de passe chiffré, comme une chaîne de caractère. Vous pouvez utiliser la procédure `crypt` pour cela :

```
(user-account
  (name "charlie")
  (group "users")

  ;; Spécifie un mot de passe initial hashé avec sha512.
  (password (crypt "InitialPassword!" "$6$abc")))
```

**Remarque:** Le hash de ce mot de passe initial sera disponible dans un fichier dans `/gnu/store`, lisible par tous les utilisateurs, donc cette méthode est à utiliser avec soin.

Voir Section “Passphrase Storage” dans *The GNU C Library Reference Manual*, pour plus d’information sur le chiffrement des mots de passe et Section “Encryption” dans *GNU Guile Reference Manual*, pour des informations sur la procédure `crypt` de Guile.

Les déclarations de groupes sont encore plus simple :

```
(user-group (name "students"))
```

`user-group` [Type de données]

C'est le type pour, hé bien, les comptes utilisateurs. Il n'y a que quelques champs :

`name` Le nom du groupe.

`id` (par défaut : `#f`)

L'identifiant du groupe (un nombre). S'il est `#f`, un nouveau nombre est alloué automatiquement lorsque le groupe est créé.

`system?` (par défaut : `#f`)

Cette valeur booléenne indique si le groupe est un groupe « système ». les groupes systèmes ont un numéro d'ID bas.

`password` (par défaut : `#f`)

Quoi, les groupes utilisateurs peuvent avoir des mots de passe ? On dirait bien. À moins que la valeur ne soit `#f`, ce champ spécifie le mot de passe du groupe.

Par simplicité, une variable liste les groupes utilisateurs de base auxquels on pourrait s'attendre :

**%base-groups** [Variable Scheme]

C'est la liste des groupes utilisateur de base que les utilisateurs et les paquets s'attendent à trouver sur le système. Cela comprend des groupes comme « root », « wheel » et « users », ainsi que des groupes utilisés pour contrôler l'accès à certains périphériques, comme « audio », « disk » et « cdrom ».

**%base-user-accounts** [Variable Scheme]

C'est la liste des compte du système de base que les programmes peuvent s'attendre à trouver sur un système GNU/Linux, comme le compte « nobody ».

Remarquez que le compte « root » n'est pas défini ici. C'est un cas particulier et il est automatiquement ajouté qu'il soit spécifié ou non.

## 12.7 Disposition du clavier

Pour spécifier ce que fait chaque touche de votre clavier, vous devez dire au système d'exploitation quel *disposition du clavier* vous voulez utiliser. Par défaut, lorsque rien n'est spécifié, la disposition QWERTY pour l'anglais américain pour les claviers 105 touches est utilisée. Cependant, les germanophones préfèrent généralement la disposition QWERTZ, les francophones la disposition AZERTY etc. ; les hackers peuvent préférer Dvorak ou bépo, et peuvent même vouloir personnaliser plus en détails l'effet de certaines touches. Cette section explique comment faire cela.

Il y a trois composants qui devront connaître votre disposition du clavier :

- Le *chargeur d'amorçage* peut avoir besoin de connaître la disposition clavier que vous voulez utiliser (voir Section 12.14 [Configuration du chargeur d'amorçage], page 578). C'est utile si vous voulez par exemple vous assurer que vous pouvez saisir la phrase de passe de votre partition racine chiffrée avec la bonne disposition.
- Le *noyau du système d'exploitation*, Linux, en aura besoin pour configurer correctement la console (voir Section 12.2 [référence de operating-system], page 246).
- Le *serveur d'affichage graphique*, habituellement Xorg, a aussi sa propre idée sur la disposition du clavier à utiliser (voir Section 12.9.7 [Système de fenêtrage X], page 325).

Guix vous permet de configurer les trois séparément mais, heureusement, il vous permet de partager la même disposition du clavier pour chacun des trois composants.

Les dispositions de clavier sont représentées par des enregistrements créés par la procédure `keyboard-layout` de (`gnu system keyboard`). En suivant l'extension clavier de X (XKB), chaque disposition a quatre attributs : un nom (souvent un code de langue comme « fi » pour le finnois ou « jp » pour le japonais), un nom de variante facultatif, un nom de modèle de clavier facultatif et une liste éventuellement vide d'options supplémentaires. Dans la plupart des cas, vous n'aurez besoin que du nom de la disposition.

`keyboard-layout nom [variante] [#:model] [#:options '()]` [Procédure Scheme]

Renvoie une nouvelle disposition de clavier avec les *nom* et *variante* donnés.

*nom* doit être une chaîne comme "fr" ; *variante* doit être une chaîne comme "bepo" ou "nodeadkeys". Voir le paquet `xkeyboard-config` pour les options valides.

Voici quelques exemples :

```
;; La disposition QWERTZ allemande. Ici on suppose que vous utilisez un clavier
```

```
;; type « pc105 » standard.
(keyboard-layout "de")

;; La variante bépo de la disposition française.
(keyboard-layout "fr" "bepo")

;; La disposition catalane.
(keyboard-layout "es" "cat")

;; Disposition du clavier en langue arabe avec « Alt-Shift » pour passer à la mise en
(keyboard-layout "ar,us" #:options '("grp:alt_shift_toggle"))

;; La disposition espagnole américaine. En plus, la touche
;; « Verr. Maj. » est utilisée comme touche « Ctrl » supplémentaire,
;; et la touche « Menu » est utilisée comme touche « Compose » pour
;; saisir des lettres accentuées.
(keyboard-layout "latam"
  #:options '("ctrl:nocaps" "compose:menu"))

;; La disposition russe pour un clavier de ThinkPad.
(keyboard-layout "ru" #:model "thinkpad")

;; La disposition « US internationale », qui est comme la disposition US plus
;; des touches mortes pour saisir des caractères accentués. Cet exemple est pour
;; un clavier de MacBook Apple.
(keyboard-layout "us" "intl" #:model "macbook78")
```

Voir le répertoire `share/X11/xkb` du paquet `xkeyboard-config` pour une liste complète des disposition, des variantes et des modèles pris en charge.

Disons que vous voulez que votre système utilise la disposition turque sur tout le système — du chargeur d’amorçage à Xorg en passant par la console. Voici ce que votre configuration du système contiendrait :

```
;; Utiliser la disposition turque pour le chargeur d'amorçage,
;; la console et Xorg.

(operating-system
  ;; ...
  (keyboard-layout (keyboard-layout "tr")) ;pour la console
  (bootloader (bootloader-configuration
    (bootloader grub-efi-bootloader)
    (targets '("/boot/efi"))
    (keyboard-layout keyboard-layout))) ;pour GRUB
  (services (cons (set-xorg-configuration
    (xorg-configuration ;pour Xorg
      (keyboard-layout keyboard-layout)))
    %desktop-services)))
```

Dans l'exemple ci-dessus, pour GRUB et pour Xorg, nous nous référons simplement au champ `keyboard-layout` au dessus, mais on pourrait aussi bien se référer à une autre disposition. La procédure `set-xorg-configuration` communique la configuration Xorg désirée au gestionnaire de connexion, par défaut GDM.

Nous avons discuté de la manière de spécifier la disposition du clavier *par défaut* lorsque votre système démarre, mais vous pouvez aussi l'ajuster à l'exécution :

- Si vous utilisez GNOME, son panneau de configuration contient une entrée « Région & Langues » où vous pouvez choisir une ou plusieurs dispositions du clavier.
- Sous Xorg, la commande `setxkbmap` (du paquet du même nom) vous permet de changer la disposition actuelle. Par exemple, voilà comment changer la disposition pour un Dvorak américain :

```
setxkbmap us dvorak
```

- La commande `loadkey` change la disposition du clavier dans la console Linux. Cependant, remarque que `loadkeys` n'utilise *pas* la catégorisation des dispositions XKB décrite plus haut. La commande suivante charge la disposition bépo française :

```
loadkeys fr-bepo
```

## 12.8 Régionalisation

Un *paramètre linguistique* définit les conventions culturelles d'une langue et d'une région particulières (voir Section "Locales" dans *The GNU C Library Reference Manual*). Chaque paramètre linguistique a un nom de la forme *langue\_territoire.jeudef caractères* — p. ex. `fr_LU.utf8` désigne le paramètre linguistique pour le français, avec les conventions culturelles du Luxembourg, en utilisant l'encodage UTF-8.

Normalement, vous voudrez spécifier les paramètres linguistiques par défaut pour la machine en utilisant le champ `locale` de la déclaration `operating-system` (voir Section 12.2 [référence de `operating-system`], page 246).

Les paramètres régionaux choisis sont automatiquement ajoutés aux définitions des *paramètres régionaux* connues par le système au besoin, avec le jeu de caractères inféré à partir de son nom, p. ex. `bo_CN.utf8` supposera qu'il faut utiliser le jeu de caractères UTF-8. Des définitions supplémentaires peuvent être spécifiées dans le champ `locale-definitions` de `operating-system` — c'est utile par exemple si le jeu de caractères n'a pas été inféré à partir du nom. L'ensemble par défaut de définitions comprend certains paramètres linguistiques parmi les plus utilisés, mais pas toutes les variantes disponibles, pour gagner de la place.

Par exemple, pour ajouter les paramètres pour le frison septentrional en Allemagne, la valeur de ce champ serait :

```
(cons (locale-definition
      (name "fy_DE.utf8") (source "fy_DE"))
      %default-locale-definitions)
```

De me, pour gagner de la place, on peut vouloir lister dans `locale-definitions` seulement les paramètres qui sont vraiment utilisés, comme dans :

```
(list (locale-definition
      (name "ja_JP.eucjp") (source "ja_JP")))
```

```
(charset "EUC-JP")))
```

Les définitions compilées de locale sont disponibles à `/run/current-system/locale/X.Y`, où `X.Y` est la version de la `libc`, qui est l'emplacement par défaut où la GNU `libc` fournie par Guix recherche les données locales. Il est possible d'y remédier en utilisant la variable d'environnement `LOCPATH` (voir [locales-and-locpath], page 22).

La forme `locale-definition` est fournie par le module (`gnu system locale`). Des détails sont disponibles plus bas.

**locale-definition** [Type de données]

C'est le type de données d'une définition de paramètres linguistiques.

**name** Le nom du paramètre linguistique. Voir Section "Locale Names" dans *The GNU C Library Reference Manual*, pour en savoir plus sur les noms de paramètres linguistiques.

**source** Le nom de la source pour ce paramètre linguistique. C'est typiquement la partie `langue_territoire` du nom du paramètre.

**charset** (par défaut : "UTF-8")

Le « jeu de caractères » d'un paramètre linguistique, défini par l'IANA (<https://www.iana.org/assignments/character-sets>).

**%default-locale-definitions** [Variable Scheme]

Une liste des paramètres linguistiques UTF-8 couramment utilisés, utilisée comme valeur par défaut pour le champ `locale-definitions` des déclarations `operating-system`.

Ces définitions de paramètres linguistiques utilisent le *jeu de caractère normalisé* pour la partie qui suit le point dans le nom (voir Section "Using gettextized software" dans *The GNU C Library Reference Manual*). Donc par exemple il y a `uk_UA.utf8` mais *pas*, disons, `uk_UA.UTF-8`.

### 12.8.1 Considérations sur la compatibilité des données linguistiques

Les déclarations `operating-system` fournissent un champ `locale-libcs` pour spécifier les paquets GNU `libc` à utiliser pour compiler les déclarations de paramètres linguistiques (voir Section 12.2 [référence de `operating-system`], page 246). « Pourquoi je devrais m'en soucier ? », vous demandez-vous sûrement. Hé bien il se trouve que le format binaire des données linguistique est parfois incompatible d'une version de la `libc` à une autre.

Par exemple, un programme lié à la version 2.21 de la `libc` est incapable de lire les données locales produites avec la `libc` 2.22 ; pire encore, ce programme *aborts* au lieu d'ignorer simplement les données locales incompatibles<sup>5</sup>. De même, un programme lié à la `libc` 2.22 peut lire la plupart des données locales de la `libc` 2.21, mais pas toutes (en particulier, les données `LC_COLLATE` sont incompatibles) ; ainsi, les appels à `setlocale` peuvent échouer, mais les programmes ne s'arrêteront pas.

<sup>5</sup> Versions 2.23 et suivantes de GNU `libc` ignorera simplement les données locales incompatibles, ce qui est déjà une amélioration.

Le « problème » avec Guix c’est que les utilisateurs ont beaucoup de liberté : ils peuvent choisir s’ils veulent et quand ils veulent mettre à jour les logiciels de leur profil, et peuvent utiliser une version différente de la `libc` de celle que l’administrateur système utilise pour construire les données linguistiques du système global.

Heureusement, les utilisateur·rice·s non privilégié·e·s peuvent aussi installer leurs propres données linguistiques et définir `GUIX_LOCPATH` comme il le faut (voir [locales-and-locpath], page 22).

Cependant, c’est encore mieux si les données linguistiques du système dans `/run/current-system/locale` étaient construites avec les versions de la `libc` utilisées sur le système, pour que tous les programmes puissent y accéder — c’est surtout crucial sur un système multi-utilisateurs. Pour cela, l’administrateur peut spécifier plusieurs paquets de la `libc` dans le champ `locale-libcs` de `operating-system` :

```
(use-package-modules base)

(operating-system
 ;; ...
 (locale-libcs (list glibc-2.21 (canonical-package glibc))))
```

Cet exemple créera un système contenant les définitions des paramètres linguistiques pour la `libc` 2.21 et pour la version actuelle de la `libc` dans `/run/current-system/locale`.

## 12.9 Services

Une part importante de la préparation d’une déclaration `operating-system` est la liste des *services systèmes* et de leur configuration (voir Section 12.1 [Utiliser le système de configuration], page 238). Les services systèmes sont typiquement des démons lancés au démarrage ou d’autres actions requises à ce moment-là — p. ex. configurer les accès réseaux.

Guix a une définition large de « service » (voir Section 12.18.1 [Composition de services], page 599), mais beaucoup de services sont gérés par le GNU Shepherd (voir Section 12.18.4 [Services Shepherd], page 607). Sur un système lancé, la commande `herd` vous permet de lister les services disponibles, montrer leur statut, les démarrer et les arrêter, ou faire d’autres opérations spécifiques (voir Section “Jump Start” dans *The GNU Shepherd Manual*). Par exemple :

```
# herd status
```

La commande ci-dessus, lancée en `root`, liste les services actuellement définis. La commande `herd doc` montre un synopsis du service donné et ses actions associées :

```
# 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.■
```

Les sous-commandes `start`, `stop` et `restart` ont l’effet auquel on s’attend. Par exemple, les commande suivantes stoppent le service `nscd` et redémarrent le serveur d’affichage `Xorg` :

```
# herd stop nscd
Service nscd has been stopped.
```

```
# herd restart xorg-server
Service xorg-server has been stopped.
Service xorg-server has been started.
```

Pour certains services, `herd configuration` renvoie le nom du fichier de configuration du service, ce qui peut être pratique pour inspecter sa configuration :

```
# herd configuration sshd
/gnu/store/...-sshd_config
```

Les sections suivantes documentent les services disponibles, en commençant par les services de base qui peuvent être utilisés avec une déclaration `operating-system`.

### 12.9.1 Services de base

Le module (`gnu services base`) fournit des définitions de services pour les services de base qu'on peut attendre du système. Les services exportés par ce module sont listés ci-dessous.

`%base-services` [Variable Scheme]

Cette variable contient une liste de services de base (voir Section 12.18.2 [Types service et services], page 600, pour plus d'informations sur les objets service) qu'on peut attendre du système : un service de connexion (`mingetty`) sur chaque tty, `syslogd`, le démon de cache de noms de la libc (`nscd`), le gestionnaire de périphériques `udev`, et plus.

C'est la valeur par défaut du champ `services` des déclarations `operating-system`. Habituellement, lors de la personnalisation d'un système, vous voudrez ajouter des services à ceux de `%base-services`, comme ceci :

```
(append (list (service avahi-service-type)
              (service openssh-service-type))
        %base-services)
```

`special-files-service-type` [Variable Scheme]

C'est le service qui met en place des « fichiers spéciaux » comme `/bin/sh` ; une instance de ce service fait partie de `%base-services`.

La valeur associée avec les services `special-files-service-type` doit être une liste de couples dont le premier élément est le « fichier spécial » et le deuxième sa cible. Par défaut il s'agit de :

```
`(("bin/sh" ,(file-append bash "bin/sh"))
  ("usr/bin/env" ,(file-append coreutils "bin/env")))
```

Si vous voulez ajouter, disons, `/bin/bash` à votre système, vous pouvez changer cela en :

```
`(("bin/sh" ,(file-append bash "bin/sh"))
  ("usr/bin/env" ,(file-append coreutils "bin/env"))
  ("bin/bash" ,(file-append bash "bin/bash")))
```

Comme cela fait partie de `%base-services`, vous pouvez utiliser `modify-services` pour personnaliser l'ensemble des fichiers spéciaux (voir Section 12.18.3 [Référence de service], page 602). Mais la manière la plus simple d'ajouter un fichier spécial est *via* la procédure `extra-special-file` (voir ci-dessous).

**extra-special-file** *file target* [Procédure Scheme]

Utilise *target* comme « fichier spécial » *file*.

Par exemple, ajouter l'une des lignes suivantes au champ `services` de votre déclaration de système d'exploitation crée un lien symbolique `/usr/bin/env` :

```
(extra-special-file "/usr/bin/env"
  (file-append coreutils "/bin/env"))
```

**host-name-service** *name* [Procédure Scheme]

Renvoie un service qui paramètre le nom d'hôte à *name*.

**console-font-service-type** [Variable Scheme]

Installez les polices données sur les ttys spécifiés (les polices sont par console virtuelle sur le noyau Linux). La valeur de ce service est une liste de paires `tty/polices`. La police peut être le nom d'une police fournie par le paquet `kbd` ou tout argument valide de `setfont`, comme dans cet exemple :

```
`(("tty1" . "LatGrkCyr-8x16")
  ("tty2" . ,(file-append
              font-tamzen
              "/share/kbd/consolefonts/TamzenForPowerline10x20.psf")))
("tty3" . ,(file-append
              font-terminus
              "/share/consolefonts/ter-132n")) ; pour HDPI
```

**login-service** *config* [Procédure Scheme]

Renvoie un service pour lancer `login` en suivant *config*, un objet `<login-configuration>` qui spécifie le message du jour, entre autres choses.

**login-configuration** [Type de données]

Le type de données qui représente la configuration de `login`.

`motd` Un objet simili-fichier contenant le « message du jour ».

`allow-empty-passwords?` (par défaut : `#t`)

Permet les mots de passes vides par défaut pour que les utilisateurs puissent se connecter au compte « `root` » la première fois après sa création.

**mingetty-service** *config* [Procédure Scheme]

Renvoie un service qui lance `mingetty` en suivant *config*, un objet `<mingetty-configuration>`, qui spécifie le `tty` à lancer entre autres choses.

**mingetty-configuration** [Type de données]

C'est le type de données représentant la configuration de `Mingetty`, qui fournit l'implémentation par défaut de l'écran de connexion des consoles virtuelles.

`tty` Le nom de la console sur laquelle tourne ce `Mingetty`, p. ex. `"tty1"`.

`auto-login` (par défaut : `#f`)

Lorsque la valeur est vraie, ce champ doit être une chaîne de caractère dénotant le nom d'utilisateur pour lequel le système se connecte automatiquement. Lorsque la valeur est `#f`, il faut entrer un nom d'utilisateur et un mot de passe pour se connecter.



`login-program` (par défaut : `#f`)

Ce doit être soit `#f`, auquel cas le programme de connexion par défaut est utilisé (`login` de la suite d'outils Shadow), soit une gexp dénotant le nom d'un programme de connexion.

`login-pause?` (par défaut : `#f`)

Lorsque la valeur est `#t` en plus de *auto-login*, l'utilisateur devra appuyer sur une touche avant que le shell de connexion ne soit lancé.

`clear-on-logout?` (par défaut : `#t`)

Lorsque la valeur est `#t`, l'écran sera nettoyé après la déconnexion.

`mingetty` (par défaut : `mingetty`)

Le paquet Mingetty à utiliser.

`agetty-service config`

[Procédure Scheme]

Renvoie un service pour lancer agetty en suivant *config*, un objet `<agetty-configuration>`, qui spécifie le tty à lancer, entre autres choses.

`agetty-configuration`

[Type de données]

Ce type de données représente la configuration de agetty, qui implémente l'écran de connexion des consoles virtuelles et series. Voir la page de manuel de `agetty(8)` pour plus d'informations.

`tty`

Le nom de la console sur laquelle cet agetty fonctionne, sous forme de chaîne de caractères, par exemple `"ttyS0"`. Cet argument est optionnel, il sera par défaut un port série raisonnable par défaut utilisé par le noyau Linux.

Pour cela, s'il y a une valeur pour une option `agetty.tty` sur la ligne de commande du noyau, agetty extraira le nom du périphérique du port série à partir de cette option.

Sinon et s'il y a une valeur pour une option `console` avec un `tty` sur la ligne de commande du noyau Linux, agetty extraira le nom du périphérique du port série et l'utilisera.

Dans les deux cas, agetty laissera les autres paramètres du périphérique série (baud, etc.) sans y toucher — dans l'espoir que Linux leur a assigné les bonnes valeurs.

`baud-rate` (par défaut : `#f`)

Une chaîne qui contient une liste d'un ou plusieurs taux de baud séparés par des virgules, en ordre décroissant.

`term` (par défaut : `#f`)

Une chaîne de caractères contenant la valeur utilisée pour la variable d'environnement `TERM`.

`eight-bits?` (par défaut : `#f`)

Lorsque la valeur est `#t`, le `tty` est supposé être propre pour les caractères 8-bit et la détection de parité est désactivée.

- auto-login** (par défaut : **#f**)  
Lorsqu'un nom de connexion est passé comme une chaîne de caractères, l'utilisateur spécifié sera automatiquement connecté sans demande du nom d'utilisateur ni du mot de passe.
- no-reset?** (par défaut : **#f**)  
Lorsque la valeur est **#t**, ne vide pas les cflags du terminal (modes de contrôle).
- host** (par défaut : **#f**)  
Ceci accepte une chaîne de caractères contenant le "login\_host", qui sera écrite dans le fichier `/var/run/utmpx`.
- remote?** (par défaut : **#f**)  
Lorsque la valeur est **#t** en plus de *host*, cette option ajoutera une option `fakehost -r` à la ligne de commande du programme de connexion spécifié dans *login-program*.
- flow-control?** (par défaut : **#f**)  
Lorsque la valeur est **#t**, active le contrôle de flux matériel (RTS/CTS).
- no-issue?** (par défaut : **#f**)  
Lorsque la valeur est **#t**, le contenu du fichier `/etc/issue` ne sera pas affiché avant de présenter l'écran de connexion.
- init-string** (par défaut : **#f**)  
Cette option accepte une chaîne de caractères qui sera envoyée au tty ou au modem avant toute autre chose. Elle peut être utilisée pour initialiser un modem.
- no-clear?** (par défaut : **#f**)  
Lorsque la valeur est **#t**, `agetty` ne nettoiera pas l'écran avant de montrer l'écran de connexion.
- login-program** (par défaut : `(file-append shadow "/bin/login")`)  
Cette option doit être soit une `gexp` dénotant le nom d'un programme de connexion, soit non définie, auquel cas la valeur par défaut est la commande `login` de la suite d'outils `Shadow`.
- local-line** (par défaut : **#f**)  
Contrôle le drapeau `CLOCAL`. Cette option accepte l'un des trois symboles comme argument, `'auto`, `'always` ou `'never`. Si la valeur est **#f**, la valeur par défaut choisie par `agetty` est `'auto`.
- extract-baud?** (par défaut : **#f**)  
Lorsque la valeur est **#t**, dit à `agetty` d'essayer d'extraire la taux de baud depuis les messages de statut produits par certains modems.
- skip-login?** (par défaut : **#f**)  
Lorsque la valeur est **#t**, ne demande par de nom d'utilisateur. Elle peut être utilisée avec le champ *login-program* pour utiliser des systèmes de connexion non standards.

- no-newline?** (par défaut : #f)  
Lorsque la valeur est #t, n'affiche pas de retour à la ligne avant d'afficher le fichier `/etc/issue`.
- login-options** (par défaut : #f)  
Cette option accepte une chaîne de caractères contenant des options passées au programme login. Lorsqu'utilisé avec `login-program`, soyez conscient qu'un utilisateur malicieux pourrait essayer de rentrer un nom d'utilisateur contenant des options incluses qui pourraient être analysées par le programme de connexion.
- login-pause** (par défaut : #f)  
Lorsque la valeur est #t, attend qu'une touche soit appuyée avant de montrer l'écran de connexion. Cela peut être utilisé avec `auto-login` pour sauvegarder de la mémoire en lançant les shells de manière fainéante.
- chroot** (par défaut : #f)  
Change de racine dans le répertoire donné. Cette option accepte un chemin en tant que chaîne de caractères.
- hangup?** (par défaut : #f)  
Utilise l'appel système Linux `vhangup` pour raccrocher virtuellement le terminal spécifié.
- keep-baud?** (par défaut : #f)  
Lorsque la valeur est #t, essaye de garder le taux de baud existant. Les taux de baud de `baud-rate` sont utilisés lorsque `agetty` reçoit un caractère BREAK.
- timeout** (par défaut : #f)  
Lorsque la valeur est un nombre entier, termine la session si aucun nom d'utilisateur n'a pu être lu après `timeout` secondes.
- detect-case?** (par défaut : #f)  
Lorsque la valeur est #t, active le support pour la détection des terminaux en majuscule uniquement. Ce paramètre détectera qu'un nom d'utilisateur qui ne contient que des majuscules indique un terminal en majuscule et effectuera des conversion de majuscule en minuscule. Remarquez que cela ne fonctionne pas avec les caractères unicode.
- wait-cr?** (par défaut : #f)  
Lorsque la valeur est #t, attend que l'utilisateur ou le modem envoie un retour chariot ou un saut de ligne avant d'afficher `/etc/issue` ou l'écran de connexion. Cela est typiquement utilisé avec l'option `init-string`.
- no-hints?** (par défaut : #f)  
Lorsque la valeur est #t, n'affiche pas les astuces à propos des verrouillages numériques, majuscule et défilement.
- no-hostname?** (par défaut : #f)  
Par défaut, le nom d'hôte est affiché. Lorsque la valeur est #t, aucun nom d'hôte ne sera affiché.

`long-hostname?` (par défaut : `#f`)

Par défaut, le nom d'hôte n'est affiché qu'après le premier point. Lorsque la valeur est `#t`, le nom d'hôte pleinement qualifié renvoyé par `gethostname` ou `getaddrinfo` sera affiché.

`erase-characters` (par défaut : `#f`)

Cette option accepte une chaîne de caractères de caractères supplémentaires qui devraient être interprétés comme des effacements lorsque l'utilisateur les tape dans leur nom d'utilisateur.

`kill-characters` (par défaut : `#f`)

Cette option accepte une chaîne de caractères qui devrait être interprétée comme signifiant « ignore tous les caractères précédents » (aussi appelé un caractère « kill ») lorsque l'utilisateur·rice tape son nom d'utilisateur·rice.

`chdir` (par défaut : `#f`)

Cette option accepte, en tant que chaîne de caractères, un chemin vers un répertoire dans lequel se trouvera la commande avant la connexion.

`delay` (par défaut : `#f`)

Cette option accepte, en tant qu'entier, le nombre de secondes à attendre avant d'ouvrir le tty et afficher l'écran de connexion.

`nice` (par défaut : `#f`)

Cette option accepte, en tant qu'entier, la valeur « nice » avec laquelle le programme `login` tourne.

`extra-options` (par défaut : `'()`)

Cette option fournit un « mécanisme de secours » pour que l'utilisateur·rice puisse ajouter des arguments arbitraires en ligne de commande à `agetty` comme une liste de chaînes de caractères.

`rshepherd-requirement` (par défaut : `'()`)

L'option peut être utilisée pour fournir des prérequis `shepherd` supplémentaires (par exemple `'syslogd`) aux services `shepherd` `'term-*` correspondants.

`kmscon-service-type` *config*

[Procédure Scheme]

Renvoie un service qui lance `kmscon` (<https://www.freedesktop.org/wiki/Software/kmscon>) d'après *config*, un objet `<kmscon-configuration>`, qui spécifie le tty sur lequel tourner, entre autres choses.

`kmscon-configuration`

[Type de données]

C'est le type de données représentant la configuration de `Kscon`, qui implémente l'écran de chargement de la console virtuelle.

`virtual-terminal`

Le nom de la console sur laquelle `Kmscon` tourne, p. ex. `"tty1"`.

`login-program` (par défaut : `#~(string-append #shadow "/bin/login")`)

Une `gexp` qui dénote le nom d'un programme de connexion. le programme de connexion par défaut est `login` de la suite d'outils `Shadow`.

**login-arguments** (par défaut : `'("-p")`)

Une liste d'arguments à passer à `login`.

**auto-login** (par défaut : `#f`)

Lorsqu'un nom de connexion est passé comme une chaîne de caractères, l'utilisateur spécifié sera automatiquement connecté sans demande du nom d'utilisateur ni du mot de passe.

**hardware-acceleration?** (par défaut : `#f`)

S'il faut utiliser l'accélération matérielle.

**font-engine** (par défaut : `"pango"`)

Moteur de polices utilisé dans `Kmscon`.

**font-size** (par défaut : `12`)

Taille de police utilisée par `Kmscon`.

**keyboard-layout** (par défaut : `#f`)

Si la valeur est `#f`, `Kmscon` utilise la disposition du clavier par défaut — habituellement la disposition anglaise américaine (« `qwerty` ») pour un clavier de PC à 105 touches.

Sinon cela doit être un objet `keyboard-layout` spécifiant la disposition du clavier. Voir Section 12.7 [Disposition du clavier], page 263, pour plus d'informations sur la manière de spécifier la disposition du clavier.

**kmscon** (par défaut : `kmscon`)

Le paquet `Kmscon` à utiliser.

**nscd-service** [*config*] [*#:glibc glibc*] [*#:name-services '()*] [Procédure Scheme]

Renvoie un service qui lance le démon de cache de services de noms de la `libc` (`nscd`) avec la *config* donnée — un objet `<nscd-configuration>`. Voir Section 12.12 [Name Service Switch], page 573, pour un exemple.

Parce que c'est pratique, le service du Shepherd pour `nscd` fournit les actions suivantes :

**invalidate**

Cela invalide le cache donné. Par exemple, en lançant :

```
herd invalidate nscd hosts
```

on invalide le cache de noms d'hôtes de `nscd`.

**statistiques**

Lancer `herd statistics nscd` affiche des informations sur l'utilisation de `nscd` et des caches.

**%nscd-default-configuration** [Variable Scheme]

C'est la valeur par défaut de `<nscd-configuration>` (voir plus bas) utilisée par `nscd-service`. Elle utilise les caches définis par `%nscd-default-caches` ; voir plus bas.

**nscd-configuration** [Type de données]

C'est le type de données qui représente la configuration du démon de cache de services de noms (`nscd`).

- name-services** (par défaut : '() )  
Liste des paquets dénotant des *services de noms* qui doivent être visible pour `nscd`, p. ex. (`list nss-mdns`).
- glibc** (par défaut : `glibc`)  
Objet de paquet qui dénote la Bibliothèque C de GNU qui fournit la commande `nscd`.
- log-file** (par défaut : `"/var/log/nscd.log"`)  
Nom du fichier journal de `nscd`. C'est là que les sorties de débogage sont envoyée lorsque `debug-level` est strictement positif.
- debug-level** (par défaut : 0)  
Entier qui dénote le niveau de débogage. Les entiers les plus grands signifient plus de sortie de débogage.
- caches** (par défaut : `%nscd-default-caches`)  
Liste d'objets `<nscd-cache>` qui dénotent des choses à mettre en cache ; voir plus bas.

**nscd-cache** [Type de données]

Type de données représentant une base de données de cache de `nscd` et ses paramètres.

**database** C'est un symbole qui représente le nom de la base de donnée à mettre en cache. Les valeurs valide sont `passwd`, `group`, `hosts` et `services` qui désignent les bases de données NSS correspondantes (voir Section "NSS Basics" dans *The GNU C Library Reference Manual*).

**positive-time-to-live**

**negative-time-to-live** (par défaut : 20)

Un entier qui représente le nombre de secondes pendant lesquelles un résultat positif ou négatif reste en cache.

**check-files?** (par défaut : `#t`)

Indique s'il faut vérifier des mises à jours dans les fichiers correspondant à *database*.

Par exemple, lorsque *database* est `hosts`, ce drapeau indique à `nscd` de vérifier s'il y a des mises à jour de `/etc/hosts` et de les prendre en compte.

**persistent?** (par défaut : `#t`)

Indique si le cache devrait être stocké de manière persistante sur le disque.

**shared?** (par défaut : `#t`)

Indique si le cache devrait être partagé entre les utilisateurs.

**max-database-size** (par défaut : 32 MiB)

Taille maximale en octets de la base de données en cache.

**%nscd-default-caches** [Variable Scheme]

Liste d'objets `<nscd-cache>` utilisés par défaut par `nscd-configuration` (voir plus haut).

Elle active la mise en cache persistante et agressive des recherches de services et de noms d'hôtes. Ces derniers fournissent une recherche de noms d'hôtes plus performante, résiliente face à des serveurs de noms peu fiables et une protection de votre vie privée plus efficace — souvent le résultat des recherches de noms d'hôtes sont dans le cache local, donc les serveurs de nom externes n'ont même pas besoin d'être questionnés.

**syslog-configuration** [Type de données]

Ce type de données représente la configuration du démon syslog.

**syslogd** (par défaut : `#{string-append #$inetutils "/libexec/syslogd"}`)

Le démon syslog à utiliser.

**config-file** (par défaut : `%default-syslog.conf`)

Le fichier de configuration de syslog à utiliser.

**syslog-service config** [Procédure Scheme]

Renvoie un service qui lance un démon syslog en suivant *config*.

Voir Section “syslogd invocation” dans *GNU Inetutils*, pour plus d'informations sur la syntaxe du fichier de configuration.

**guix-service-type** [Variable Scheme]

C'est le type de service qui lance le démon de construction, **guix-daemon** (voir Section 2.5 [Invoquer guix-daemon], page 17). Sa valeur doit être un enregistrement **guix-configuration** décrit plus bas.

**guix-configuration** [Type de données]

Ce type de données représente la configuration du démon de construction de Guix. Voir Section 2.5 [Invoquer guix-daemon], page 17, pour plus d'informations.

**guix** (par défaut : *guix*)

Le paquet Guix à utiliser.

**build-group** (par défaut : `"guixbuild"`)

Nom du groupe des comptes utilisateurs de construction.

**build-accounts** (par défaut : 10)

Nombre de comptes utilisateurs de construction à créer.

**authorize-key?** (par défaut : `#t`)

Indique s'il faut autoriser ou non les clefs de substituts listées dans **authorize-keys** — par défaut celle de `ci.guix.gnu.org` et `bordeaux.guix.gnu.org` (voir Section 6.3 [Substituts], page 54).

Lorsque **authorize-key?** est vrai, `/etc/guix/acl` ne peut pas être changé en invoquant `guix archive --authorize`. vous devez plutôt ajuster **guix-configuration** comme vous le souhaitez et reconfigurer le système. Cela s'assure que le fichier de configuration de votre système est auto-contenu.

**Remarque:** Au démarrage ou la reconfiguration d'un système où **authorize-key?** est vrai, le fichier `/etc/guix/acl` existant est sauvegardé dans `/etc/guix/acl.bak` s'il est

déterminé qu'il a été modifié manuellement. Cela facilite la migration à partir de versions précédentes, qui permettaient des modifications en-place de `/etc/guix/acl`.

`authorized-keys` (par défaut : `%default-authorized-guix-keys`)

La liste des fichiers de clés autorisées pour les imports d'archives, en tant que liste de gexps sous forme de chaînes (voir Section 6.10 [Invoquer guix archive], page 71). Par défaut, elle contient celle de `ci.guix.gnu.org` et `bordeaux.guix.gnu.org` (voir Section 6.3 [Substituts], page 54). Voir `substitute-urls` ci-dessous pour apprendre comment la changer.

`use-substitutes?` (par défaut : `#t`)

S'il faut utiliser les substituts.

`substitute-urls` (par défaut : `%default-substitute-urls`)

La liste des URL où trouver des substituts par défaut.

Supposons que vous vouliez récupérer des substituts à partir de `guix.example.org` en plus de `ci.guix.gnu.org`. Vous devrez faire deux choses : (1) ajouter `guix.example.org` à `substitute-urls`, et (2) autoriser sa clé de signature, après avoir effectué les vérifications adéquates (voir Section 6.3.2 [Autoriser un serveur de substituts], page 55). La configuration ci-dessous fait exactement cela :

```
(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)))
```

Cet exemple suppose que le fichier `./guix.example.org-key.pub` contient la clé publique que `guix.example.org` utilise pour signer les substituts.

`generate-substitute-keys?` (par défaut : `#t`)

Indique s'il faut générer une *paire de clés de substituts* dans `/etc/guix/signing-key.pub` et `/etc/guix/signing-key.sec` si elle n'existe pas déjà.

Cette paire de clés est utilisée pour exporter les éléments du dépôt, par exemple avec `guix publish` (voir Section 10.11 [Invoquer guix publish], page 222) ou `guix archive` (voir Section 6.10 [Invoquer guix archive], page 71). Générer une paire de clés prend quelques secondes quand il y a assez d'entropie disponible et n'est fait qu'une seule fois ; vous pouvez la désactiver par exemple dans une machine virtuelle qui n'en pas besoin et où le temps de démarrage supplémentaire peut être problématique.



`max-silent-time` (par défaut : 0)

`timeout` (par défaut : 0)

Le nombre de secondes de silence et le nombre de secondes d'inactivité, respectivement, après lesquelles un processus de construction son délai d'attente. Une valeur de zéro désactive le délai d'attente.

`log-compression` (par défaut : 'gzip')

Le type de compression utilisé par les journaux de construction — parmi `gzip`, `bzip2` et `none`.

`discover?` (par défaut : #f)

Indique s'il faut découvrir les serveurs de substitut sur le réseau local avec mDNS et DNS-SD.

`extra-options` (par défaut : '()')

Liste d'options supplémentaires de la ligne de commande pour `guix-daemon`.

`log-file` (par défaut : "/var/log/guix-daemon.log")

Le fichier où les sorties standard et d'erreur de `guix-daemon` sont écrites.

`http-proxy` (par défaut : #f)

L'URL du proxy HTTP et HTTPS utilisé pour le téléchargement des dérivés et substituts à sortie fixe.

Il est également possible de changer le proxy du démon au moment de l'exécution grâce à l'action `set-http-proxy`, qui le redémarre :

```
herd set-http-proxy guix-daemon http://localhost:8118
```

Pour effacer les paramètres du proxy, exécutez :

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

`tmpdir` (par défaut : #f)

Un répertoire où `guix-daemon` effectuera ses constructions.

`guix-extension` [Type de données]

Ce type de données représente les paramètres du démon de construction de Guix qui peuvent être étendus. C'est le type d'objet qui doit être utilisé dans une extension du service `guix`. Voir Section 12.18.1 [Composition de services], page 599, pour plus d'informations.

`authorized-keys` (par défaut : '()')

Une liste d'objets simili-fichiers où chaque élément contient une clé publique.

`substitute-urls` (par défaut : '()')

Une liste de chaînes où chaque élément est l'URL d'un serveur de substituts.

`chroot-directories` (par défaut : '()')

Une liste d'objets simili-fichiers ou de chaînes pointant vers des répertoires supplémentaires que le démon de construction peut utiliser.

`udev-service` [#:*udev eudev* #:*rules* '()] [Procédure Scheme]

Lance *udev*, qui remplit le répertoire */dev* dynamiquement. Les règles *udev* peuvent être fournies comme une liste de fichier via la variable *rules*. Les procédures *udev-rule*, *udev-rules-service* et (*gnu services base*) simplifient la création de ces fichiers de règle.

The *herd rules udev* command, as root, returns the name of the directory containing all the active *udev* rules.

`udev-rule` [*file-name contents*] [Procédure Scheme]

Renvoie un fichier de règle *udev* nommé *file-name* contenant les règles définie par le littéral *contents*.

Dans l'exemple suivant, on définit une règle pour un périphérique USB qui sera stockée dans le fichier *90-usb-thing.rules*. La règle lance un script à la détection du périphérique USB avec l'identifiant de produit donné.

```
(define %example-udev-rule
  (udev-rule
    "90-usb-thing.rules"
    (string-append "ACTION==" "add", "SUBSYSTEM==" "usb", "
                  "ATTR{product}==" "Example", "
                  "RUN+=" "/path/to/script")))
```

`udev-rule-service` [*name rules*] [#:*groups groups*] [Procédure Scheme]

Renvoie un service qui étend *udev-service-type* avec *rules* et *account-service-type* avec *groups* comme groupes système. Cela fonctionne en créant un service de type *name-udev-rules*, dont le service renvoyé est une instance.

Nous montrons ici comment cela peut être utilisé pour étendre *udev-service-type* avec la règle précédemment définie *%exemple-udev-rule*.

```
(operating-system
 ;; ...
 (services
  (cons (udev-rules-service 'usb-thing %example-udev-rule)
        %desktop-services)))
```

`file->udev-rule` [*file-name file*] [Procédure Scheme]

Renvoie un fichier *udev* nommé *file-name* contenant les règles définies dans *file*, un objet simili-fichier.

L'exemple suivant montre comment utiliser un fichier de règles existant.

```
(use-modules (guix download)      ;pour url-fetch
             (guix packages)      ;pour 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/MORf30/"
                    "android-udev-rules/" version "/51-android.rules"))
(sha256
 (base32 "0lmmagpyb6xsq6zcr2w1cyx9qmjqmajkvrdbhjx32gqf1d9is003"))))
```

En plus, les définitions des paquets de Guix peuvent être inclus dans *rules* pour étendre les règles avec les définitions trouvées dans leur sous-répertoire `lib/udev/rules.d`. Au lieu de l'exemple *file->udev-rule* précédent, on aurait pu utiliser le paquet *android-udev-rules* qui existe dans le module `(gnu packages android)`.

L'exemple suivant montre comment utiliser le paquet *android-udev-rules* pour que l'outil Android `adb` puisse détecter les appareils sans privilège root. Il détaille aussi comment créer le groupe `adbusers`, requis pour le bon fonctionnement des règles définies dans le paquet *android-udev-rules*. Pour créer ce groupe, on doit le définir dans les `supplementary-groups` de la déclaration `user-account` ainsi que dans le champ `groups` de l'enregistrement *operating-system*.

```
(use-modules (gnu packages android) ;pour android-udev-rules
             (gnu system shadow) ;pour user-group
             ...)

```

```
(operating-system
 ;; ...
 (users (cons (user-account
              ;; ...
              (supplementary-groups
                ("adbusers" ;for adb
                 "wheel" "netdev" "audio" "video")))))
 ;; ...
 (services
  (cons (udev-rules-service 'android android-udev-rules
                           #:groups ("adbusers"))
        %desktop-services)))

```

**urandom-seed-service-type** [Variable Scheme]

Conserve de l'entropie dans `%random-seed-file` pour amorcer `/dev/urandom` lors du redémarrage. Il essaie également d'amorcer `/dev/urandom` à partir de `/dev/hwrng` lors du démarrage, si `/dev/hwrng` existe et est lisible.

**%random-seed-file** [Variable Scheme]

C'est le nom du fichier où des octets aléatoires sont sauvegardés par *urandom-seed-service* pour démarrer `/dev/urandom` au redémarrage. Sa valeur par défaut est `/var/lib/random-seed`.

**gpm-service-type** [Variable Scheme]

C'est le type du service qui lance GPM, le *démon de souris à but général*, qui fournit le support de la souris sur la console Linux. GPM permet aux utilisateurs d'utiliser la souris dans la console, entre autres pour sélectionner, copier et coller du texte.

La valeur pour les services de ce type doit être un `gpm-configuration` (voir plus bas). Ce service ne fait pas partie de `%base-services`.

**gpm-configuration** [Type de données]

Type de données représentant la configuration de GPM.

**options** (par défaut : `%default-gpm-options`)

Les options de la ligne de commande à passer à `gpm`. L'ensemble des options par défaut dit à `gpm` d'écouter les événements de la souris dans `/dev/input/mice`. Voir Section "Command Line" dans *gpm manual*, pour plus d'informations.

**gpm** (par défaut : `gpm`)

Le paquet GPM à utiliser.

**guix-publish-service-type** [Variable Scheme]

C'est le type de service pour `guix publish` (voir Section 10.11 [Invoquer `guix publish`], page 222). Sa valeur doit être un objet `guix-publish-configuration` décrit plus bas.

Ce service suppose que `/etc/guix` contient déjà une paire de clefs créée par `guix archive --generate-key` (voir Section 6.10 [Invoquer `guix archive`], page 71). Si ce n'est pas le cas, le service ne démarrera pas.

**guix-publish-configuration** [Type de données]

Le type de données représentant la configuration du service `guix publish`.

**guix** (par défaut : `guix`)

Le paquet Guix à utiliser.

**port** (par défaut : `80`)

Le port TCP sur lequel écouter les connexions.

**host** (par défaut : `"localhost"`)

L'hôte (et donc, l'interface réseau) sur lequel écouter. Utilisez `"0.0.0.0"` pour écouter sur toutes les interfaces réseaux.

**advertise?** (par défaut : `#f`)

Lorsque la valeur est vraie, annonce le service sur le réseau local via le protocole DNS-SD, avec Avahi.

Cela permet aux machines Guix voisines qui ont activé la découverte (voir `guix-configuration` ci-dessus) de découvrir cette instance de `guix publish` et de télécharger des substituts directement à partir d'elle.

**compression** (par défaut : `'(("gzip" 3) ("zstd" 3))`)

Il s'agit d'une liste de méthodes de compression/numéros de niveau utilisés lors de la compression des substituts. Par exemple, pour compresser tous les substituts avec *both* `lzip` au niveau 7 et `gzip` au niveau 9, écrivez :

```
'(("lzip" 7) ("gzip" 9))
```

Le niveau 9 atteint le meilleur taux de compression au détriment d'une utilisation accrue du CPU, tandis que le niveau 1 atteint une compression rapide. Voir Section 10.11 [Invoquer `guix publish`], page 222, pour

plus d'information sur les méthodes de compression disponibles et les compromis à faire entre elles.

Une liste vide désactive complètement la compression.

**nar-path** (par défaut : "nar")

Le chemin d'URL où les « nars » se trouvent. Voir Section 10.11 [Invoquer guix publish], page 222, pour des détails.

**cache** (par défaut : #f)

Lorsque la valeur est #f, désactive le cache et génère les archives à la demande. Sinon, cela devrait être le nom d'un répertoire — p. ex. "/var/cache/guix/publish" — où guix publish gère le cache des archives et des métadonnées prêtes à être envoyées. Voir Section 10.11 [Invoquer guix publish], page 222, pour plus d'informations sur les compromis impliqués.

**workers** (par défaut : #f)

Lorsque la valeur est un entier, c'est le nombre de threads de travail utilisés pour le cache ; lorsque la valeur est #f, le nombre de processeurs est utilisé. Voir Section 10.11 [Invoquer guix publish], page 222, pour plus d'informations.

**cache-bypass-threshold** (par défaut : 10 Mio)

Lorsque cache est vrai, c'est la taille maximale en octets d'un élément du dépôt pour lequel guix publish peut contourner le cache s'il n'y est pas. Voir Section 10.11 [Invoquer guix publish], page 222, pour plus d'informations.

**ttl** (par défaut : #f)

Lorsque la valeur est un entier, il dénote la *durée de vie* en secondes des archives publiées. Voir Section 10.11 [Invoquer guix publish], page 222, pour plus d'informations.

**negative-ttl** (par défaut : #f)

Lorsque la valeur est un entier, il dénote la *durée de vie* en secondes pour les réponses négatives. Voir Section 10.11 [Invoquer guix publish], page 222, pour plus d'informations.

**rngd-service** [#:rng-tools rng-tools] [#:device [Procédure Scheme]  
"/dev/hwrng"]

Renvoie un service qui lance le programme rngd de rng-tools pour ajouter device à la réserve d'entropie du noyau. Le service échouera si device n'existe pas.

**pam-limits-service** [#:limits '()] [Procédure Scheme]

Renvoie un service qui installe un fichier de configuration pour le module pam\_limits ([http://linux-pam.org/Linux-PAM-html/sag-pam\\_limits.html](http://linux-pam.org/Linux-PAM-html/sag-pam_limits.html)). La procédure prend éventuellement une liste de valeurs pam-limits-entry qui peuvent être utilisées pour spécifier les limites ulimit et les priorités nice des sessions utilisateurs. La définition de limites suivante définit deux limites matérielles et logicielles pour toutes les sessions connectées des utilisateurs du groupe realtime :

```
(pam-limits-service
```

```
(list
  (pam-limits-entry "@realtime" 'both 'rtprio 99)
  (pam-limits-entry "@realtime" 'both 'memlock 'unlimited)))
```

La première entrée augmente la priorité en temps réel maximale des processus non privilégiés ; la deuxième entrée abandonne les restrictions sur l'espace d'adressage maximal qui peut être verrouillé en mémoire. Ces paramètres sont souvent utilisés sur les systèmes audio temps-réel.

Un autre exemple utile est d'augmenter le nombre maximum de descripteurs de fichiers ouverts qui peuvent être utilisés :

```
(pam-limits-service
  (list
    (pam-limits-entry "*" 'both 'nofile 100000)))
```

Dans l'exemple au-dessus, l'astérisque signifie que la limite s'applique à tous les utilisateurs. Il est important de s'assurer que la valeur choisie n'excède pas la valeur maximale du système visible dans le fichier `/proc/sys/fs/file-max`, sinon les utilisateurs ne pourront pas se connecter. Pour plus d'information sur les limites du module d'authentification greffable (PAM), regardez la page de manuel `'pam_limits'` du paquet `linux-pam`.

**greetd-service-type** [Variable Scheme]

`greetd` (<https://git.sr.ht/~kennylevinsen/greetd>) est un démon de gestion des connexions minimaliste et flexible, qui ne fait aucune hypothèse sur ce que vous voulez lancer.

Si vous pouvez le lancer depuis votre shell dans un TTY, `greetd` peut le lancer. S'il peut apprendre à parler un protocole IPC simple basé sur JSON, alors ce peut être un écran d'accueil.

`greetd-service-type` fournit l'infrastructure nécessaire pour la connexion des utilisateurs, ce qui comprend :

- le service PAM `greetd`
- Une variante spéciale de `pam-mount` pour monter `XDG_RUNTIME_DIR`

Voici un exemple pour passer de `mingetty-service-type` à `greetd-service-type` et comment vous pouvez le spécifier pour plusieurs terminaux :

```
(append
  (modify-services %base-services
    ;; greetd-service-type fournit le service PAM « greetd »
    (delete login-service-type)
    ;; et peut être utilisé à la place de mingetty-service-type
    (delete mingetty-service-type))
  (list
    (service greetd-service-type
      (greetd-configuration
        (terminals
          (list
            ;; on peut rendre n'importe quel terminal actif par défaut
            (greetd-terminal-configuration (terminal-vt "1") (terminal-switch
```

```

;; on peut créer des environnement sans initialiser XDG_RUNTIME_D
;; et même fournir nos propres variables d'environnement
(greetd-terminal-configuration
 (terminal-vt "2")
 (default-session-command
  (greetd-agreety-session
   (extra-env '(("MY_VAR" . "1"))
   (xdg-env? #f))))
;; on peut utiliser un shell différent au lieu de bash par défaut
(greetd-terminal-configuration
 (terminal-vt "3")
 (default-session-command
  (greetd-agreety-session (command (file-append zsh "/bin/zsh"))
  ;; on peut utiliser n'importe quelle autre commande exécutable co
(greetd-terminal-configuration
 (terminal-vt "4")
 (default-session-command (program-file "my-noop-greeter" #~(exit
(greetd-terminal-configuration (terminal-vt "5"))
(greetd-terminal-configuration (terminal-vt "6"))))))
;; mingetty-service-type peut être utilisé en parallèle
;; si vous voulez faire cela, n'utilisez pas (delete login-service-type)
;; comme indiqué plus haut
#! (service mingetty-service-type (mingetty-configuration (tty "tty8"))) |#)

```

**greetd-configuration** [Type de données]

La configuration pour `greetd-service-type`.

**motd** Un objet simili-fichier contenant le « message du jour ».

**allow-empty-passwords?** (par défaut : `#t`)  
Permet les mots de passes vides par défaut pour que les utilisateurs puissent se connecter au compte « root » la première fois après sa création.

**terminal** (par défaut : `'()`)  
Liste de `greetd-terminal-configuration` par terminal pour lequel `greetd` doit démarrer.

**greeter-supplementary-groups** (par défaut : `'()`)  
Liste des groupes qui devraient être ajoutés à l'utilisateur `greeter`. Par exemple :

```
(greeter-supplementary-groups ("seat" "video"))
```

Remarquez que cet exemple échouera si le groupe `seat` n'existe pas.

**greetd-terminal-configuration** [Type de données]

Enregistrement de la configuration par terminal du service `greetd`.

**greetd** (par défaut : `greetd`)  
Le paquet `greetd` à utiliser.

**config-file-name**

Nom du fichier de configuration utilisé pour le démon `greetd`. En général, c'est une dérivation générée à partir de la valeur de `terminal-vt`.

**log-file-name**

Nom du fichier de journalisation utilisé par le démon `greetd`. En général, c'est un nom généré à partir de la valeur de `terminal-vt`.

**terminal-vt** (par défaut : `"7"`)

Le terminal virtuel sur lequel tourner. L'utilisation d'un VT spécifique en évitant les conflits est recommandé.

**terminal-switch** (par défaut : `#f`)

Rend ce terminal actif au démarrage de `greetd`.

**default-session-user** (par défaut : `"greeter"`)

L'utilisateur à utiliser pour lancer l'écran d'accueil.

**default-session-command** (par défaut : `(greetd-agreety-session)`)

Peut être soit une instance d'une configuration `greetd-agreety-session` ou un objet comme `gexp->script` à utiliser comme écran d'accueil.

**greetd-agreety-session** [Type de données]

Enregistrement de la configuration de l'écran d'accueil `agreety` de `greetd`.

**agreety** (par défaut : `greetd`)

Le paquet dans lequel se trouve la commande `/bin/agreety`.

**command** (par défaut : `(file-append bash "/bin/bash")`)

La commande à démarrer par `/bin/agreety` si la connexion réussit.

**command-args** (par défaut : `'("-1")`)

Les arguments à passer à la commande.

**extra-env** (par défaut : `'()`)

Variables d'environnement supplémentaires à définir à la connexion.

**xdg-env?** (par défaut : `#t`)

Si la valeur est vraie, `XDG_RUNTIME_DIR` et `XDG_SESSION_TYPE` seront initialisés avant de démarrer la commande. Il faut remarquer que les variables `extra-env` sont initialisées juste après les variables mentionnées, donc elles peuvent être écrasées.

**greetd-wlgreet-session** [Type de données]

Enregistrement de configuration générique de l'écran d'accueil `wlgreet` de `greetd`.

**wlgreet** (par défaut : `wlgreet`)

Le paquet dans lequel se trouve la commande `/bin/wlgreet`.

**command** (par défaut : `(file-append sway "/bin/sway")`)

La commande à démarrer par `/bin/wlgreet` si la connexion réussit.

**command-args** (par défaut : `'()`)

Les arguments à passer à la commande.



**output-mode** (par défaut : "all")  
Option à utiliser pour `outputMode` dans le fichier de configuration TOML.

**sclae** (par défaut : 1)  
Option à utiliser pour `scale` dans le fichier de configuration TOML.

**background** (par défaut : '(0 0 0 0.9))  
Liste RGBA à utiliser comme couleur de fond pour l'invite de connexion.

**headline** (par défaut : '(1 1 1 1))  
Liste RGBA à utiliser comme couleur de titre dans la popup.

**prompt** (par défaut : '(1 1 1 1))  
Liste RGBA à utiliser comme couleur d'invite dans la popup.

**prompt-error** (par défaut : '(1 1 1 1))  
Liste RGBA à utiliser comme couleur d'erreur dans la popup.

**border** (par défaut : '(1 1 1 1))  
Liste RGBA à utiliser comme couleur de bordure de la popup.

**extra-env** (par défaut : '())  
Variables d'environnement supplémentaires à définir à la connexion.

**greetd-wlgreet-sway-session** [Type de données]  
Enregistrement de la configuration spécifique à sway, de l'écran d'accueil wlgreet de greetd.

**wlgreet-session** (par défaut : (greetd-wlgreet-session))  
Un enregistrement `greetd-wlgreet-session` pour la configuration générale de wlgreet, en plus du `greetd-wlgreet-sway-session` spécifique à Sway.

**sway** (par défaut : sway)  
Le paquet qui fournit la commande `/bin/sway`.

**sway-configuration** (par défaut : #f)  
Objet simili-fichier qui fournit un fichier de configuration Sway supplémentaire à ajouter au début de la partie obligatoire de la configuration.

Voici un exemple de configuration de greetd qui utilise wlgreet et Sway :

```
(greetd-configuration
;; Nous devons donner ces permissions à l'utilisateur de l'écran d'accueil, si
;; Sway plantera au démarrage.
(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 Exécution de tâches planifiées

Le module (`gnu services mcron`) fournit une interface pour GNU mcron, un démon qui lance des tâches planifiées (voir *GNU mcron*). GNU mcron est similaire au démon Unix traditionnel `cron` ; la principale différence est qu'il est implémenté en Guile Scheme, qui fournit beaucoup de flexibilité lors de la spécification de la planification des tâches et de leurs actions.

L'exemple en dessous définit un système d'exploitation qui lance les commandes `updatedb` (voir Section "Invoking updatedb" dans *Finding Files*) et `guix gc` (voir Section 6.5 [Invoquer guix gc], page 60) tous les jours, ainsi que la commande `mkid` en tant qu'utilisateur non privilégié (voir Section "mkid invocation" dans *ID Database Utilities*). Il utilise des `gexps` pour introduire des définitions de tâches qui sont passées à mcron (voir Section 9.12 [G-Expressions], page 166).

```
(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
  ;; Lance le ramasse-miettes tous les jours à minuit cinq.
  ;; L'action de la tâche est une commande shell.
  #~(job "5 0 * * *" ;Vixie cron syntax
        "guix gc -F 1G"))
```

```
(define idutils-job
  ;; Met à jour la base de données d'index en tant que « charlie » à 12h15
  ;; et 19h15. La commande est lancée depuis le répertoire personnel de l'utilisateur
  #~(job '(next-minute-from (next-hour '(12 19)) '(15))
        (string-append #$idutils "/bin/mkid src")
        #:user "charlie"))
```

```
(operating-system
  ;; ...
```

```
  ;; %BASE-SERVICES inclus déjà une instance de
  ;; « mcron-service-type », que nous étendons avec un
  ;; travail supplémentaire avec « simple-service ».
  (services (cons (simple-service 'my-cron-jobs
                                mcron-service-type
```

```

                (list garbage-collector-job
                    updatedb-job
                    idutils-job))
        %base-services)))

```

**Astuce:** Lorsque vous fournissez l'action d'une spécification de tâche avec une procédure, vous devriez donner un nom explicite à la tâche via le troisième argument facultatif comme dans l'exemple `updatedb-job` ci-dessus. Sinon, la tâche apparaîtra comme « Lambda function » dans la sortie de `herd schedule mcron`, ce qui n'est pas très descriptif !

Pour les tâches plus complexes définies dans Scheme où vous devez contrôler le haut niveau, par exemple pour introduire un formulaire `use-modules`, vous pouvez déplacer votre code vers un programme séparé en utilisant la procédure `program-file` du module (`guix gexp`) (voir Section 9.12 [G-Expressions], page 166). L'exemple ci-dessous illustre cette possibilité.

```

(define %battery-alert-job
  ;; Beep when the battery percentage falls below %MIN-LEVEL.
  #~(job
    '(next-minute (range 0 60 1))
    #$(program-file
      "battery-alert.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-level 20)

          (setenv "LC_ALL" "C") ; assure que la sortie est en anglais
          (and-let* ((input-pipe (open-pipe*
                                   OPEN_READ
                                   #$(file-append acpi "/bin/acpi")))
                    (output (get-string-all input-pipe))
                    (m (string-match "Discharging, ([0-9]+)%" output))
                    (level (string->number (match:substring m 1)))
                    (<< level %min-level)))
            (format #t "warning: Battery level is low (~a%)~%" level)
            (invoke #$(file-append beep "/bin/beep") "-r5"))))))))

```

Voir Section “Guile Syntax” dans *GNU mcron*, pour plus d'informations sur les spécifications des tâches de mcron. Ci-dessous est la référence du service mcron.

Sur un système lancé, vous pouvez utiliser l'action `schedule` du service pour visualiser les travaux mcron qui seront exécutés ensuite :

```
# herd schedule mcron
```

Cet exemple ci-dessus montre les cinq tâches qui seront exécutés, mais vous pouvez spécifier le nombre de tâches à afficher :

```
# herd schedule mcron 10
```

**mcron-service-type** [Variable Scheme]

C'est le type du service `mcron`, dont la valeur est un objet `mcron-configuration`.

Ce type de service peut être la cible d'une extension de service qui fournit des spécifications de tâches supplémentaires (voir Section 12.18.1 [Composition de services], page 599). En d'autres termes, il est possible de définir des services qui fournissent des tâches `mcron` à lancer.

**mcron-configuration** [Type de données]

Les champs de `mcron-configuration` disponibles sont :

`mcron` (par défaut : `mcron`) (type : simili-fichier)

Le paquet `mcron` à utiliser.

`jobs` (par défaut : `()`) (type : liste-de-gexps)

C'est la liste des gexps (voir Section 9.12 [G-Expressions], page 166), où chaque gexp correspond à une spécification de tâche de `mcron` (voir Section "Syntax" dans *GNU mcron*).

`log?` (par défaut : `#t`) (type : booléen)

Journalise les messages sur la sortie standard.

`log-format` (par défaut : `"~1@*~a ~a: ~a~%"`) (type : chaîne)

Chaîne de format (`ice-9 format`) pour les messages de journaux. La valeur par défaut produit des messages de type « `'pid name: message'` » (voir Section "Invoking `mcron`" dans *GNU mcron*). Chaque message est aussi préfixé par un horodatage par le GNU Shepherd.

### 12.9.3 Rotation des journaux

Les fichiers journaux comme ceux qui se trouvent dans `/var/log` ont tendance à grandir sans fin, donc c'est une bonne idée de le *faire tourner* de temps à autres — c.-à-d. archiver leur contenu dans des fichiers séparés, potentiellement compressés. Le module (`gnu services admin`) fournit une interface pour GNU Rot[t]log, un outil de rotation de journaux (voir *GNU Rot[t]log Manual*).

Ce service fait partie de `%base-services`, et se trouve donc activé par défaut, avec les paramètres par défaut, pour les fichiers journaux les plus fréquemment rencontrés. L'exemple ci-dessous montre comment l'étendre avec une *rotation* supplémentaire, si vous devez le faire (généralement, les services qui produisent des fichiers journaux s'en chargent déjà) :

```
(use-modules (guix) (gnu))
```

```
(use-service-modules admin)
```

```
(define my-log-files
```

```
  ;; Journaux que je veux faire tourner.
```

```
  '("/var/log/something.log" "/var/log/another.log"))
```

```
(operating-system
```

```
;; ...
(services (cons (simple-service 'rotate-my-stuff
                             rottlog-service-type
                             (list (log-rotation
                                   (frequency 'daily)
                                   (files my-log-files))))
              %base-services)))
```

**rottlog-service-type** [Variable Scheme]

C'est le type du service Rotlog, dont la valeur est un objet `rottlog-configuration`.

D'autres services peuvent étendre celui-ci avec de nouveaux objets `log-rotation` (voir plus bas), en augmentant ainsi l'ensemble des fichiers à faire tourner.

Ce type de service peut définir des tâches (voir Section 12.9.2 [Exécution de tâches planifiées], page 287) pour lancer le service `rottlog`.

**rottlog-configuration** [Type de données]

Type de données représentant la configuration de `rottlog`.

**rottlog** (par défaut : `rottlog`)  
Le paquet Rotlog à utiliser.

**rc-file** (par défaut : `(file-append rottlog "/etc/rc")`)  
Le fichier de configuration Rotlog à utiliser (voir Section "Mandatory RC Variables" dans *GNU Rot[t]log Manual*).

**rotations** (par défaut : `%default-rotations`)  
Une liste d'objets `log-rotation` définis plus bas.

**jobs** C'est une liste de gexps où chaque gexp correspond à une spécification de tâche de `mcron` (voir Section 12.9.2 [Exécution de tâches planifiées], page 287).

**log-rotation** [Type de données]

Type de données représentant la rotation d'un groupe de fichiers journaux.

En reprenant un exemple du manuel de Rotlog (voir Section "Period Related File Examples" dans *GNU Rot[t]log Manual*), on peut définir la rotation d'un journal de cette manière :

```
(log-rotation
 (frequency 'daily)
 (files '("/var/log/apache/*"))
 (options '("storedir apache-archives"
           "rotate 6"
           "notifempty"
           "nocompress")))
```

La liste des champs est la suivante :

**frequency** (par défaut : `'weekly`)  
La fréquence de rotation, un symbole.

**files** La liste des fichiers ou des motifs de noms de fichiers à faire tourner.

`options` (par défaut : `%default-log-rotation-options`)

La liste des options de `rotlog` pour cette rotation (voir Section “Configuration parameters” dans *GNU Rot[t]log Manual*).

`post-rotate` (par défaut : `#f`)

Soit `#f`, soit une `gexp` à exécuter une fois la rotation terminée.

`%default-rotations` [Variable Scheme]

Spécifie la rotation hebdomadaire de `%rotated-files` et de `/var/log/guix-daemon.log`.

`%rotated-files` [Variable Scheme]

La liste des fichiers contrôlés par `syslog` à faire tourner. Par défaut il s’agit de : `'(/var/log/messages" "/var/log/secure" "/var/log/debug" \"/var/log/mailllog")`.

Certains fichiers journaux doivent simplement être supprimés périodiquement quand ils deviennent trop vieux, sans autre critère et sans étape d’archivage. C’est le cas des journaux de construction stockés par `guix-daemon` dans `/var/log/guix/drvs` (voir Section 2.5 [Invoquer `guix-daemon`], page 17). Le service `log-cleanup` gère ce cas. Par exemple, `%base-services` (voir Section 12.9.1 [Services de base], page 268) inclus ce qui suit :

```
;; Supprime régulièrement les anciens journaux de construction.
(service log-cleanup-service-type
  (log-cleanup-configuration
    (directory "/var/log/guix/drvs"))))
```

Cela s’assure que les journaux de construction ne s’accumulent pas sans fin.

`log-cleanup-service-type` [Variable Scheme]

C’est le type du service pour supprimer les anciens journaux. Sa valeur doit être un enregistrement `log-cleanup-configuration`, décrit ci-dessous.

`log-cleanup-configuration` [Type de données]

Type de données représentant la configuration de nettoyage des journaux

`directory`

Nom du répertoire contenant les fichiers journaux.

`expiry` (par défaut : `(* 6 30 24 3600)`)

Âge en secondes après lequel un fichier peut être supprimé (six moins par défaut).

`schedule` (par défaut : `"30 12 01,08,15,22 * *"`)

Chaîne ou `gexp` dénotant la date de la tâche `mcron` correspondant (voir Section 12.9.2 [Exécution de tâches planifiées], page 287).

## Service Anonip

Anonip est un filtre pour la vie privée qui supprime les adresses IP des journaux des serveurs web. Ce service crée une FIFO et filtre toutes les lignes écrites avec `anonip` avant d’écrire les journaux filtrés dans un fichier cible.

L'exemple suivant configure la FIFO `/var/run/anonip/https.access.log` et écrit le fichier journal filtré dans `/var/log/anonip/https.access.log`.

```
(service anonip-service-type
  (anonip-configuration
    (input "/var/run/anonip/https.access.log")
    (output "/var/log/anonip/https.access.log")))
```

Configurez votre serveur web pour écrire ses journaux dans la FIFO `/var/run/anonip/https.access.log` et récupérez les fichiers journaux anonymisés dans `/var/web-logs/https.access.log`.

**anonip-configuration** [Type de données]

Ce type de données représente la configuration d'anonip. Il a les paramètres suivants :

**anonip** (par défaut : **anonip**)  
Le paquet anonip à utiliser.

**input** Le nom de fichier du fichier journal d'entrée à traiter. Le service crée une FIFO de ce nom. Le serveur web doit écrire ses journaux dans cette FIFO.

**output** Le nom du fichier journal traité.

Les réglages facultatifs suivants peuvent être fournis :

**skip-private?**  
Lorsque la valeur est **#true**, ne masque pas les adresses dans les plages privées.

**column** Un numéro de colonne commençant par 1 pour la première colonne. Suppose que l'adresse IP est dans la colonne donnée (1 par défaut).

**replacement**  
Chaîne de remplacement au cas où l'analyse de l'adresse échoue, p. ex. "0.0.0.0".

**ipv4mask** Nombre de bits à masquer dans les adresses IPv4.

**ipv6mask** Nombre de bits à masquer dans les adresses IPv6.

**increment**  
Incrémente l'adresse IP par le nombre donné. La valeur par défaut est zéro.

**delimiter**  
Chaîne de délimitation des journaux.

**regex** Expression régulière pour détecter l'adresse IP. À utiliser à la place de **column**.

### 12.9.4 Configuration du réseau

Le module (**gnu services networking**) fournit des services pour configurer les interfaces réseau et paramétrer le réseau sur votre machine. Ces services fournissent différentes manières pour paramétrer votre machine : en déclarant une configuration réseau statique, en lançant

un client DHCP (Dynamic Host Configuration Protocol) ou en lançant des démon comme NetworkManager et Connman qui automatisent le processus complet, en s'adaptant automatiquement aux changements de connectivité et en fournissant une interface utilisateur de haut niveau.

Sur un portable, NetworkManager et Connman sont de loin les options les plus pratiques, ce qui explique que les services de bureau par défaut incluent NetworkManager voir Section 12.9.9 [Services de bureaux], page 347). Pour un serveur, une machine virtuelle ou un conteneur, la configuration réseau statique ou un simple client DHCP sont plus souvent mieux appropriés.

Cette section décrit les divers services de paramétrage réseau existants, en commençant par le configuration réseau statique.

**static-networking-service-type** [Variable Scheme]

C'est le type pour les interfaces réseaux configurées statiquement. Sa valeur doit être une liste d'enregistrements **static-networking**. Chacun déclare un ensemble d'adresses, de routes et de liens, comme indiqué plus bas.

Voici la plus simple des configurations, avec un seul contrôleur d'interface réseau (NIC) et une seule connectivité IPv4 :

```
;; Réseau statique pour un NIC, IPv4 uniquement.
(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")))))
```

Le bout de code ci-dessus peut être ajouté au champ **services** de votre configuration du système d'exploitation (voir Section 12.1 [Utiliser le système de configuration], page 238). Il configurera votre machine pour avoir l'adresse IP 10.0.2.15, avec un masque de sous-réseau de 24 bits pour le réseau local — cela signifie que toutes les adresse 10.0.2.x sont sur le réseau local (LAN). Le trafic à destination d'adresses extérieures au réseau local est routé *via* 10.0.2.2. Les noms d'hôte sont résolus en envoyant des requêtes au système de nom de domaine (DNS) à 10.0.2.3.

**static-networking** [Type de données]

C'est le type de données représentant la configuration d'un réseau statique.

Voici un exemple de la manière dont vous pouvez déclarer la configuration sur une machine avec un seul contrôleur d'interface réseau (NIC) disponible sous le nom **eno1** et avec une adresse IPv4 et une adresse IPv6 :

```
;; Configuration réseau pour un NIC, IPv4 + IPv6.
(static-networking
  (addresses (list (network-address
```



```

        (device "eno1")
        (value "10.0.2.15/24"))
(network-address
 (device "eno1")
 (value "2001:123:4567:101::1/64"))))
(routes (list (network-route
              (destination "default")
              (gateway "10.0.2.2"))
             (network-route
              (destination "default")
              (gateway "2020:321:4567:42::1"))))
(name-servers '("10.0.2.3"))

```

Si vous connaissez la commande `ip` du paquet `iproute2` (<https://wiki.linuxfoundation.org/networking/iproute2>) qui se trouve sur les systèmes Linux, la déclaration ci-dessus est équivalente à exécuter :

```

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

```

Exécutez `man 8 ip` pour plus d'informations. Les utilisateurs et utilisatrices vétérans de GNU/Linux sauront sans doute comment le faire avec `ifconfig` et `route`, mais nous vous épargnons cela.

Les champs disponibles pour ce type de données sont les suivants :

**adresses**

**links** (par défaut : '()')

**routes** (par défaut : '()')

La liste de enregistrements `network-address`, `network-link` et `network-route` pour ce réseau (voir plus bas).

**name-servers** (par défaut : '()')

La liste des adresses IP (chaines) des serveurs de nom de domaine. Ces adresses IP vont dans `/etc/resolv.conf`.

**provision** (par défaut : '(networking)')

Si la valeur est vraie, cela devrait être une liste de symboles pour le service Shepherd correspondant à cette configuration réseau.

**requirement** (par défaut : '()')

La liste des services Shepherd dont celui-ci dépend.

**network-address** [Type de données]

C'est le type de données représentant l'adresse IP d'une interface réseau.

**device** Le nom de l'interface réseau pour cette adresse — p. ex. "eno1".

**valeur** L'adresse IP effective et le masque de sous-réseau en notation CIDR (Classless Inter-Domain Routing) ([https://fr.wikipedia.org/wiki/Adresse\\_IP#Agr%C3%A9gation\\_des\\_adresses](https://fr.wikipedia.org/wiki/Adresse_IP#Agr%C3%A9gation_des_adresses)), en tant que chaîne de caractères.

Par exemple, "10.0.2.15/24" dénote l'adresse IPv4 10 0 2 15 sur un sous-réseau de 24 bits — toutes les adresses 10.0.2.x sont sur le même réseau local.

`ipv6?` Indique si `value` dénote une adresse IPv6. Par défaut cela est déterminé automatiquement.

`network-route` [Type de données]

C'est le type de donnée représentant une route réseau.

`destination`

La destination de la route (une chaîne), soit une adresse IP et un masque de sous-réseau, soit "default" qui dénote la route par défaut.

`source` (par défaut : #f)

Le source de la route.

`device` (par défaut : #f)

Le périphérique utilisé pour la route — p. ex. "eno2".

`ipv6?` (par défaut : auto)

Indique si c'est une route IPv6. Par défaut cela est déterminé automatiquement en fonction de `destination` ou de `gateway`.

`gateway` (par défaut : #f)

Adresse IP (une chaîne) à travers laquelle le trafic est routé.

`network-link` [Type de données]

Type de données pour un lien réseau (voir Section "Link" dans *le manuel de Guile-Netlink*).

`name` Le nom du lien — p. ex. "v0p0".

`type` Un symbole dénotant le type de lien — p. ex. 'veth.

`arguments`

La liste des arguments pour ce type de lien.

`%loopback-static-networking` [Variable Scheme]

C'est l'enregistrement `static-networking` qui représente le « périphérique de rebouclage », `lo`, pour les adresses IP 127.0.0.1 et ::1, et qui fournit le service Shepherd `loopback`.

`%qemu-static-networking` [Variable Scheme]

C'est l'enregistrement `static-networking` représentant la configuration réseau pour utiliser la pile réseau en mode utilisateur de QEMU sur `eth0` (voir Section "Using the user mode network stack" dans *la documentation de QEMU*).

`dhcp-client-service-type` [Variable Scheme]

C'est le type de services qui lance `dhcp`, un client DHCP (protocole de configuration d'hôte dynamique).

**dhcp-client-configuration** [Type de données]

Le type de données représentant la configuration du service client DHCP.

**package** (par défaut : `isc-dhcp`)

Le paquet du client DHCP à utiliser.

**interfaces** (par défaut : `'all'`)

Soit `'all'`, soit la liste des noms d'interface sur lesquelles le client DHCP devrait écouter — p. ex. `'("eno1")'`.

Lorsque la valeur est `'all'`, le client DHCP écoute sur toutes les interfaces disponibles et activables en dehors des interfaces de rebouclage. Sinon le client DHCP écoute uniquement sur les interfaces spécifiées.

**network-manager-service-type** [Variable Scheme]

C'est le type de service pour le service NetworkManager (<https://wiki.gnome.org/Projects/NetworkManager>). La valeur pour ce type de service est un enregistrement `network-manager-configuration`.

Ce service fait partie de `%desktop-services` (voir Section 12.9.9 [Services de bureaux], page 347).

**network-manager-configuration** [Type de données]

Type de données représentant la configuration de NetworkManager.

**network-manager** (par défaut : `network-manager`)

Le paquet NetworkManager à utiliser.

**dns** (par défaut : `"default"`)

Mode de gestion pour le DNS, qui affecte la manière dont NetworkManager utilise le fichier de configuration `resolv.conf`.

`'default'` NetworkManager mettra à jour `resolv.conf` pour refléter les serveurs de noms fournis par les connexions actives.

`'dnsmasq'` NetworkManager exécutera `dnsmasq` comme un serveur de noms local en cache, en utilisant une configuration *transfert conditionnel* si vous êtes connecté à un VPN, puis mettra à jour `resolv.conf` pour pointer vers le serveur de noms local. Avec ce paramètre, vous pouvez partager votre connexion réseau. Par exemple, lorsque vous souhaitez partager votre connexion réseau avec un autre ordinateur portable *via* un câble Ethernet, vous pouvez ouvrir `nm-connection-editor` et configurer la méthode de connexion câblée pour IPv4 et IPv6 de manière à ce qu'elle soit "partagée avec d'autres ordinateurs" et à ce que la connexion soit rétablie (ou redémarrée).

Vous pouvez également établir une *connexion hôte-invité* aux VM QEMU (voir Section 3.8 [Installer Guix dans une VM], page 36). Avec une connexion d'hôte à invité, vous pouvez par exemple : accéder à un serveur Web fonctionnant sur la VM (voir Section 12.9.19 [Services web], page 441) à partir

d'un navigateur Web sur votre système hôte, ou vous connecter à la VM *via* SSH (voir Section 12.9.5 [Services réseau], page 299). Pour établir une connexion entre hôtes, exécutez cette commande une fois :

```
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
```

Ensuite, chaque fois que vous lancez votre VM QEMU (voir Section 12.17 [Lancer Guix dans une VM], page 597), passez `-nic tap,ifname=tap0,script=no,downscript=no` à `qemu-system-....`

`'none'` NetworkManager ne modifiera pas `resolv.conf`.

`vpn-plugins` (par défaut : '()')

C'est la liste des greffons disponibles pour les VPN (réseaux privés virtuels). Un exemple est le paquet `network-manager-openvpn`, qui permet à NetworkManager de gérer des VPN via OpenVPN.

`connman-service-type` [Variable Scheme]

C'est le type de service pour lancer Connman (<https://01.org/connman>), un gestionnaire de connexions réseaux.

Sa valeur doit être un enregistrement `connman-configuration` comme dans cet exemple :

```
(service connman-service-type
  (connman-configuration
    (disable-vpn? #t)))
```

Voir plus bas pour des détails sur `connman-configuration`.

`connman-configuration` [Type de données]

Type de données représentant la configuration de connman.

`connman` (par défaut : `connman`)

Le paquet connman à utiliser.

`disable-vpn?` (par défaut : `#f`)

Lorsque la valeur est vraie, désactive le greffon `vpn` de connman.

`wpa-supPLICANT-service-type` [Variable Scheme]

C'est le type du service qui lance WPA supplicant ([https://w1.fi/wpa\\_supplicant/](https://w1.fi/wpa_supplicant/)), un démon d'authentification requis pour s'authentifier sur des WiFi chiffrés ou des réseaux ethernet.

`wpa-supPLICANT-configuration` [Type de données]

Type données qui représente la configuration de WPA Supplicant.

Il prend les paramètres suivants :

`wpa-supPLICANT` (par défaut : `wpa-supPLICANT`)

Le paquet WPA Supplicant à utiliser.

- requirement** (par défaut : '(user-processes loopback syslogd)  
Liste des services qui doivent être démarrés avant le début de WPA Supplicant.
- dbus?** (par défaut : #t)  
Indique s'il faut écouter les requêtes sur D-Bus.
- pid-file** (par défaut : "/var/run/wpa\_supplicant.pid")  
Où stocker votre fichier de PID.
- interface** (par défaut : #f)  
Si une valeur est indiquée, elle doit spécifier le nom d'une interface réseau que WPA supplicant contrôlera.
- config-file** (par défaut : #f)  
Fichier de configuration facultatif à utiliser.
- extra-options** (par défaut : '() )  
Liste d'arguments de la ligne de commande supplémentaires à passer au démon.

Certains périphériques réseau comme les modems ont des besoins spécifiques et c'est ce dont s'occupe les services suivants.

**modem-manager-service-type** [Variable Scheme]  
C'est le type de service pour le service ModemManager (<https://wiki.gnome.org/Projects/ModemManager>). La valeur de ce type de service est un enregistrement **modem-manager-configuration**.

Ce service fait partie de %desktop-services (voir Section 12.9.9 [Services de bureaux], page 347).

**modem-manager-configuration** [Type de données]  
Type de donnée représentant la configuration de ModemManager.

**modem-manager** (par défaut : modem-manager)  
Le paquet ModemManager à utiliser.

**usb-modeswitch-service-type** [Variable Scheme]  
C'est le type de service pour le service USB\_ModeSwitch ([https://www.draisberghof.de/usb\\_modeswitch/](https://www.draisberghof.de/usb_modeswitch/)). La valeur pour ce type de service est un enregistrement **usb-modeswitch-configuration**.

Lorsqu'ils sont branchés, certains modems USB (et autres dispositifs USB) se présentent initialement comme un support de stockage en lecture seule et non comme un modem. Ils doivent être *modeswitched* avant d'être utilisables. Le service de type USB\_ModeSwitch installe des règles udev pour commuter automatiquement les modes de ces dispositifs lorsqu'ils sont branchés.

Ce service fait partie de %desktop-services (voir Section 12.9.9 [Services de bureaux], page 347).

**openvswitch-configuration** [Type de donnée]

Type de données représentant la configuration du USB\_ModeSwitch.

**usb-modeswitch** (par défaut : `usb-modeswitch`)

Le paquet USB\_ModeSwitch fournit les binaires pour la commutation de modes.

**usb-modeswitch-data** (par défaut : `usb-modeswitch-data`)

Le paquet fournissant les données du dispositif et le fichier de règles udev utilisé par USB\_ModeSwitch.

**config-file** (par défaut : `#~(string-append #$usb-modeswitch:dispatcher "/etc/usb_modeswitch.conf")`)

Le fichier de configuration à utiliser pour le répartiteur USB\_ModeSwitch. Par défaut, le fichier de configuration livré avec USB\_ModeSwitch est utilisé, ce qui désactive la connexion au `/var/log` parmi d'autres paramètres par défaut. S'il est défini sur `#f`, aucun fichier de configuration n'est utilisé.

## 12.9.5 Services réseau

Le module (`gnu services networking`) dont on a parlé dans la section précédente fournit des services pour des configuration plus avancées : fournir un service DHCP que d'autres peuvent utiliser, filtrer des paquets avec `iptables` ou `nftables`, faire tourner un point d'accès WiFi avec `hostadp`, lancer le « superdémon » `inetd` et bien plus encore. Cette section les décrit.

**dhcpd-service-type** [Procédure Scheme]

Ce type définit un service qui lance un démon DHCP. Pour créer un service de ce type, vous devez appliquer un objet `<dhcpd-configuration>`. Par exemple :

```
(service dhcpd-service-type
  (dhcpd-configuration
    (config-file (local-file "my-dhcpd.conf"))
    (interfaces '("enp0s25"))))
```

**dhcpd-configuration** [Type de données]

**package** (par défaut : `isc-dhcp`)

Le paquet qui fournit le démon DHCP. ce paquet doit fournir le démon `sbin/dhcpd` relativement à son répertoire de sortie. Le paquet par défaut est le serveur DHCP d'ISC (<https://www.isc.org/dhcp/>).

**config-file** (par défaut : `#f`)

Le fichier de configuration à utiliser. Il est requis. Il sera passé à `dhcpd` via son option `-cf`. La valeur peut être n'importe quel objet « simili-fichier » (voir Section 9.12 [G-Expressions], page 166). Voir `man dhcpd.conf` pour des détails sur la syntaxe du fichier de configuration.

**version** (par défaut : `"4"`)

La version de DHCP à utiliser. Le serveur DHCP d'ISC supporte les valeur « 4 », « 6 » et « 4o6 ». Elles correspondent aux options `-4`, `-6` et `-4o6` du programme `dhcpd`. Voir `man dhcpd` pour plus de détails.

**run-directory** (par défaut : `"/run/dhcpd"`)  
Le répertoire d'exécution à utiliser. Au moment de l'activation du service, ce répertoire sera créé s'il n'existe pas.

**pid-file** (par défaut : `"/run/dhcpd/dhcpd.pid"`)  
Le fichier de PID à utiliser. Cela correspond à l'option `-pf` de `dhcpd`. Voir `man dhcpd` pour plus de détails.

**interfaces** (par défaut : `'()`)  
Les noms des interfaces réseaux sur lesquelles `dhcpd` écoute. Si cette liste n'est pas vide, alors ses éléments (qui doivent être des chaînes de caractères) seront ajoutés à l'invocation de `dhcpd` lors du démarrage du démon. Il n'est pas forcément nécessaire de spécifier des interfaces ici ; voir `man dhcpd` pour plus de détails.

**hostapd-service-type** [Variable Scheme]  
Il s'agit du type de service permettant d'exécuter le démon `hostapd` (<https://w1.fi/hostapd/>) pour mettre en place des points d'accès et des serveurs d'authentification WiFi (IEEE 802.11). Sa valeur associée doit être un `hostapd-configuration` comme indiqué ci-dessous :

```
;; Utiliser wlan1 pour lancer le point d'accès pour « My Network ».
(service hostapd-service-type
  (hostapd-configuration
    (interface "wlan1")
    (ssid "My Network")
    (channel 12)))
```

**hostapd-configuration** [Type de données]  
Ce type de données représente la configuration du service `hostapd`, avec les champs suivants :

**package** (par défaut : `hostapd`)  
Le paquet `hostapd` à utiliser.

**interface** (par défaut : `"wlan0"`)  
L'interface réseau pour faire fonctionner le point d'accès WiFi.

**ssid** Le SSID (*identifiant de l'ensemble de services*), une chaîne de caractères qui identifie ce réseau.

**broadcast-ssid?** (par défaut : `#t`)  
La diffusion de ce SSID.

**channel** (par défaut : `#f`)  
Le canal WiFi à utiliser.

**drivers** (par défaut : `"n180211"`)  
Le type d'interface du conducteur. `"n180211"` est utilisé avec tous les pilotes `mac80211` de Linux. Utilisez `"none"` si vous construisez `hostapd` comme un serveur RADIUS autonome qui ne # contrôle aucun pilote sans-fil/filaire.

`extra-settings` (par défaut : "")

Paramètres supplémentaires à ajouter tels quels au fichier de configuration de `hostapd`. Voir <https://w1.fi/cgit/hostap/plain/hostapd/hostapd.conf> pour la référence du fichier de configuration.

`simulated-wifi-service-type` [Variable Scheme]

C'est le type de service permettant de simuler la mise en réseau WiFi, qui peut être utile dans les machines virtuelles à des fins de test. Le service charge le noyau Linux module `mac80211_hwsim` ([https://www.kernel.org/doc/html/latest/networking/mac80211\\_hwsim/mac80211\\_hwsim.html](https://www.kernel.org/doc/html/latest/networking/mac80211_hwsim/mac80211_hwsim.html)) et lance `hostapd` pour créer un pseudo réseau WiFi qui peut être vu sur `wlan0`, par défaut.

La valeur du service est un enregistrement `hostapd-configuration`.

`iptables-service-type` [Variable Scheme]

C'est le type de service pour mettre en place une configuration `iptables`. `iptables` est un outil de filtrage de paquets pris en charge par le noyau Linux. Ce service prend en charge la configuration d'`iptables` pour IPv4 et IPv6. Un exemple de configuration simple, qui rejette les connexions entrantes sauf celles sur le port 22 est présenté ci-dessous.

```
(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` [Type de données]

Type de données représentant la configuration d'`iptables`.

`iptables` (par défaut : `iptables`)

Le paquet `iptables` qui fournit `iptables-restore` et `ip6tables-restore`.



**ipv4-rules** (par défaut : `%iptables-accept-all-rules`)

Les règles iptables à utiliser. Elles seront passées à `iptables-restore`. Cela peut être un objet « simili-fichier » (voir Section 9.12 [G-Expressions], page 166).

**ipv6-rules** (par défaut : `%iptables-accept-all-rules`)

Les règles iptables à utiliser. Elles seront passées à `ip6tables-restore`. Cela peut être un objet « simili-fichier » (voir Section 9.12 [G-Expressions], page 166).

**nftables-service-type** [Variable Scheme]

C'est le type de service permettant de mettre en place une configuration nftables. nftables est un projet de netfilter qui vise à remplacer le cadre existant iptables, ip6tables, arptables et ebtables. Il fournit un nouveau cadre de filtrage de paquets, un nouvel utilitaire d'espace utilisateur-riche `nft`, et une couche de compatibilité pour iptables. Ce service est fourni avec un jeu de règles par défaut `%default-nftables-ruleset` qui rejette toutes les connexions entrantes sauf celles vers le port ssh 22. Pour l'utiliser, il suffit d'écrire :

```
(service nftables-service-type)
```

**nftables-configuration** [Type de données]

Type de données représentant la configuration de nftables.

**package** (par défaut : `nftables`)

Le paquet nftables qui fournit `nft`.

**ruleset** (par défaut : `%default-nftables-ruleset`)

Les règles d'utilisation de nftables. Il peut s'agir de n'importe quel objet « de type fichier » (voir Section 9.12 [G-Expressions], page 166).

**ntp-service-type** [Variable Scheme]

C'est le type de service qui lance le démon Network Time Protocol (NTP) (`https://www.ntp.org`), `ntpd`. Le démon gardera l'horloge système synchronisée avec celle des serveurs NTP spécifiés.

La valeur de ce service est un objet `ntp-configuration`, décrit ci-dessous.

**ntp-configuration** [Type de données]

C'est le type de données représentant la configuration du service NTP.

**servers** (par défaut : `%ntp-servers`)

C'est la liste des serveurs (enregistrements `<ntp-server>`) avec lesquels `ntpd` sera synchronisé. Voir la définition du type de données `ntp-server` ci-dessous.

**allow-large-adjustment?** (par défaut : `#t`)

Détermine si `ntpd` peut faire un ajustement initial de plus de 1 000 secondes.

**ntp** (par défaut : `ntp`)

Le paquet NTP à utiliser.

**%ntp-servers** [Variable Scheme]  
 Liste de noms d'hôtes à utiliser comme serveurs NTP par défaut. Ce sont les serveurs du projet NTP Pool (<https://www.ntppool.org/fr/>).

**ntp-server** [Type de données]  
 Le type de données représentant la configuration d'un serveur NTP.

**type** (par défaut : 'server)  
 Le type de serveur NTP, indiqué par un symbole. L'un des types suivants : 'pool, 'server, 'peer, 'broadcast ou 'manycastclient.

**adresse** L'adresse du serveur, comme une chaîne de caractères.

**options** Les options NTPD à utiliser avec ce serveur spécifique, données sous la forme d'une liste de noms d'options et/ou de tuples de noms et de valeurs d'options. L'exemple suivant définit un serveur à utiliser avec les options `iburst` et `prefer`, ainsi que `version 3` et un temps `maxpoll` de 16 secondes.

```
(ntp-server
 (type 'server)
 (address "some.ntp.server.org")
 (options `(iburst (version 3) (maxpoll 16) prefer)))
```

**openntpd-service-type** [Procédure Scheme]  
 Lance le démon NTP `ntpd`, implémenté par OpenNTPD (<http://www.openntpd.org>). Le démon gardera l'horloge système synchronisée avec celle des serveurs donnés.

```
(service
 openntpd-service-type
 (openntpd-configuration
 (listen-on '("127.0.0.1" ":::1"))
 (sensor '("udcf0 correction 70000"))
 (constraint-from '("www.gnu.org"))
 (constraints-from '("https://www.google.com/"))))
```

**%openntpd-servers** [Variable Scheme]  
 Cette variable est une liste des adresses de serveurs définies dans `%ntp-servers`.

**openntpd-configuration** [Type de données]

**openntpd** (par défaut : (file-append openntpd "/sbin/ntpd"))  
 L'exécutable openntpd à utiliser.

**listen-on** (par défaut : '("127.0.0.1" ":::1"))  
 Une liste d'adresses IP locales ou de noms d'hôtes que devrait écouter le démon `ntpd`.

**query-from** (par défaut : '())  
 Une liste d'adresses IP que le démon devrait utiliser pour les requêtes sortantes.

- sensor** (par défaut : '() )  
Spécifie une liste de senseurs de différences de temps que `ntpd` devrait utiliser. `ntpd` écoutera chaque senseur qui existe et ignorera ceux qui n'existent pas. Voir la documentation en amont (<https://man.openbsd.org/ntpd.conf>) pour plus d'informations.
- server** (par défaut : '() )  
Spécifie une liste d'adresses IP ou de noms d'hôtes de serveurs NTP avec lesquels se synchroniser.
- servers** (par défaut : %openntp-servers)  
Spécifie une liste d'adresses IP ou de noms d'hôtes de banques de serveurs NTP avec lesquelles se synchroniser.
- constraint-from** (par défaut : '() )  
`ntpd` peut être configuré pour demander la « Date » à des serveurs HTTPS de confiance via TLS. Cette information de temps n'est pas utilisée pour sa précision mais agit comme une contrainte authentifiée, ce qui réduit l'impact d'une attaque par l'homme du milieu sur le protocole NTP non authentifié. Spécifie une liste d'URL, d'adresses IP ou de noms d'hôtes de serveurs HTTPS qui fournissent cette contrainte.
- constraints-from** (par défaut : '() )  
Comme pour `constraint-from`, spécifie une liste d'URL, d'adresses IP ou de noms d'hôtes de serveurs HTTPS qui fournissent une contrainte. Si les noms d'hôtes sont résolus en plusieurs adresses IP, `ntpd` calculera la contrainte médiane.

### `inetd-service-type` [Variable Scheme]

Ce service lance le démon `inetd` (voir Section “`inetd` invocation” dans *GNU Inetutils*). `inetd` écoute des connexions sur des sockets internet et démarre le programme spécifié uniquement lorsqu'une connexion arrive sur l'un de ces sockets.

La valeur de ce service est un objet `inetd-configuration`. L'exemple suivant configure le démon `inetd` pour qu'il fournisse le service `echo`, ainsi qu'un service `smtp` qui transfère le trafic `smtp` par `ssh` à un serveur `smtp-server` derrière une passerelle `hostname` :

```
(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" "/chemin/vers/ssh_key"
   "-W" "smtp-server:25" "user@hostname")))))))
```

Voir plus bas pour plus de détails sur `inetd`-configuration.

**inetd-configuration** [Type de données]

Type de données représentant la configuration de `inetd`.

**program** (par défaut : `(file-append inetutils "/libexec/inetd")`)

L'exécutable `inetd` à utiliser.

**entries** (par défaut : `'()`)

Une liste d'entrées de services `inetd`. Chaque entrée devrait être créée avec le constructeur `inetd-entry`.

**inetd-entry** [Type de données]

Type de données représentant une entrée dans la configuration d'`inetd`. Chaque entrée correspond à un socket sur lequel `inetd` écouterait les requêtes.

**node** (par défaut : `#f`)

Chaîne de caractères facultative, une liste d'adresses locales séparées par des virgules que `inetd` devrait utiliser pour écouter ce service. Voir Section "Configuration file" dans *GNU Inetutils* pour une description complète de toutes les options.

**name** Une chaîne de caractères dont le nom doit correspondre à une entrée de `/etc/services`.

**socket-type**

Un symbole parmi `'stream`, `'dgram`, `'raw`, `'rdm` ou `'seqpacket`.

**protocol** Une chaîne de caractères qui doit correspondre à une entrée dans `/etc/protocols`.

**wait?** (par défaut : `#t`)

Indique si `inetd` devrait attendre que le serveur ait quitté avant d'écouter de nouvelles demandes de service.

**user** Une chaîne de caractères contenant le nom d'utilisateur (et éventuellement de groupe) de l'utilisateur en tant que lequel le serveur devrait tourner. Le nom du groupe peut être spécifié comme un suffixe, séparé par un deux-points ou un point, c.-à-d. `"utilisateur"`, `"utilisateur:groupe"` ou `"utilisateur.groupe"`.

**program** (par défaut : `"internal"`)

Le programme du serveur qui servira les requêtes, ou `"internal"` si `inetd` devrait utiliser un service inclus.

**arguments** (par défaut : '() )

Une liste de chaînes de caractères ou d'objets simili-fichiers qui sont les arguments du programme du serveur, en commençant par le zéroième argument, c.-à-d. le nom du programme lui-même. Pour les services internes à `inetd`, cette entrée doit être '() ou ("internal").

Voir Section “Configuration file” dans *GNU Inetutils* pour trouver une discussion plus détaillée de chaque champ de configuration.

**opendht-service-type** [Variable Scheme]

C'est le type du service qui lance un nœud OpenDHT (<https://opendht.net>), `dhtnode`. Le démon peut être utilisé pour héberger votre propre service mandataire à la table de hash distribuée (DHT), par exemple pour y connecter Jami, entre autres applications.

**Important:** Lorsque vous utiliser le serveur mandataire OpenDHT, les adresses IP qu'il « voit » venant des clients devraient être les adresses atteignables depuis les autres pairs. En pratique cela signifie qu'une adresse accessible publiquement est préférable pour un serveur mandataire, en dehors de votre réseau privé. Par exemple, héberger le serveur mandataire sur un réseau local privé en IPv4 et l'exposer via une redirection de port pourrait fonctionner pour les pairs externes, mais les pairs locaux au mandataire verront leur adresse privée partagées avec les pairs externes, ce qui posera des problèmes de connectivité.

La valeur de ce service est un objet `opendht-configuration`, décrit ci-dessous.

**opendht-configuration** [Type de données]

Les champs de `opendht-configuration` disponibles sont :

**opendht** (par défaut : `opendht`) (type : simili-fichier)

Le paquet `opendht` à utiliser.

**peer-discovery?** (par défaut : `#f`) (type : booléen)

Indique s'il faut activer le mécanisme de découverte de pairs locaux en multidiffusion.

**enable-logging?** (par défaut : `#f`) (type : booléen)

Indique s'il faut activer la journalisation des messages vers `syslog`. C'est désactivé par défaut car cette option est plutôt verbeuse.

**debug?** (par défaut : `#f`) (type : booléen)

Indique s'il faut activer la journalisation de débogage. Cela n'a aucun effet si la journalisation n'est pas activée.

**bootstrap-host** (par défaut : `"bootstrap.jami.net:4222"`) (type : peut-être-chaîne)

Le nom d'hôte du nœud utilisé pour effectuer la première connexion au réseau. Vous pouvez fournir une valeur de port spécifique en ajoutant le suffixe `:PORT`. Par défaut, cela utilise les nœuds d'amorçage de Jami, mais n'importe quel hôte peut être spécifié ici. Il est aussi possible de désactiver l'amorçage en paramétrant explicitement ce champ à la valeur `%unset-value`.

**port** (par défaut : 4222) (type : peut-être-entier)

Le port UDP sur lequel se lier. Lorsque la valeur est laissée non spécifiée, un port disponible est automatiquement choisi.

**proxy-server-port** (type : peut-être-entier)

Crée un serveur mandataire écoutant sur le port spécifié.

**proxy-server-port-tls** (type : peut-être-entier)

Crée un serveur mandataire écoutant les connexions TLS sur le port spécifié.

**tor-service-type** [Variable Scheme]

C'est le type pour un service qui lance le démon de navigation anonyme Tor (<https://torproject.org>). Le service est configuré avec un enregistrement `<tor-configuration>`. Par défaut, le démon Tor est lancé en tant qu'utilisateur non privilégié `tor`, membre du groupe `tor`.

**tor-configuration** [Type de données]

**tor** (par défaut : `tor`)

Le paquet qui fournit le démon Tor. Ce paquet doit fournir le démon `bin/tor` relativement à son répertoire de sortie. Le paquet par défaut est l'implémentation du projet Tor (<https://www.torproject.org>).

**config-file** (par défaut : (`plain-file "empty" ""`))

Le fichier de configuration à utiliser. Il sera ajouté au fichier de configuration par défaut, et le fichier de configuration final sera passé à `tor` via son option `-f`. Cela peut être n'importe quel objet « simili-fichier » (voir Section 9.12 [G-Expressions], page 166). Voir `man tor` pour plus de détails sur la syntaxe du fichier de configuration.

**hidden-services** (par défaut : `'()`)

La liste des enregistrements `<hidden-service>` à utiliser. Pour n'importe quel service cache que vous ajoutez à cette liste, la configuration appropriée pour activer le service caché sera automatiquement ajouté au fichier de configuration par défaut. Vous pouvez aussi créer des enregistrements `<hidden-service>` avec la procédure `tor-hidden-service` décrite plus bas.

**socks-socket-type** (par défaut : `'tcp`)

Le type de socket par défaut que Tor devrait utiliser pour les socket SOCKS. Cela doit être soit `'tcp` soit `'unix`. S'il s'agit de `'tcp`, alors Tor écoutera par défaut sur le port TCP 9050 sur l'interface de boucle locale (c.-à-d. `localhost`). S'il s'agit de `'unix`, Tor écoutera sur le socket UNIX domain `/var/run/tor/socks-sock`, qui sera inscriptible pour les membres du groupe `tor`.

Si vous voulez personnaliser le socket SOCKS plus avant, laissez `socks-socket-type` à sa valeur par défaut de `'tcp` et utilisez `config-file` pour remplacer les valeurs par défaut avec votre propre option `SocksPort`.

`control-socket?` (par défaut : `#f`)

Indique s'il faut utiliser un « socket de contrôle » par lequel on peut contrôler Tor pour, par exemple, instancier des services onion dynamiquement. Si la valeur est `#t`, Tor écoutera les commandes de contrôle sur le socket UNIX domain `/var/run/tor/control-sock`, qui sera inscriptible pour les membres du groupe `tor`.

`tor-hidden-service name mapping` [Procédure Scheme]

Définit un *service caché* pour Tor nommé *name* qui implémente *mapping*. *mapping* est une liste de paires de port et d'hôte, comme dans :

```
'((22 "127.0.0.1:22")
  (80 "127.0.0.1:8080"))
```

Dans cet exemple, le port 22 du service caché est relié au port local 22 et le port 80 est relié au port local 8080.

Cela crée un répertoire `/var/lib/tor/hidden-services/name` où le fichier `hostname` contient le nom d'hôte `.onion` pour le service caché.

Voir the Tor project's documentation (<https://www.torproject.org/docs/tor-hidden-service.html.en>) pour trouver plus d'information.

Le module (`gnu services rsync`) fournit les services suivant :

Vous pourriez vouloir un démon `rsync` si vous voulez que des fichiers soient disponibles pour que n'importe qui (ou juste vous) puisse télécharger des fichiers existants ou en téléverser des nouveaux.

`rsync-service-type` [Variable Scheme]

C'est le type pour le démon `rsync` (<https://rsync.samba.org>). La valeur de ce service est un enregistrement `rsync-configuration` comme dans cet exemple :

```
;; Exporte deux répertoires par rsync. Par défaut rsync écoute sur toutes
;; les interfaces réseau.
(service rsync-service-type
  (rsync-configuration
    (modules (list (rsync-module
                    (name "music")
                    (file-name "/srv/zik")
                    (read-only? #f))
                  (rsync-module
                    (name "movies")
                    (file-name "/home/charlie/movies"))))))))
```

Voir plus pas pour trouver des détails à propos de `rsync-configuration`.

`rsync-configuration` [Type de données]

Type de données représentant la configuration de `rsync-service`.

`package` (par défaut : `rsync`)

Le paquet `rsync` à utiliser.

`address` (par défaut : `#f`)

L'adresse sur laquelle `rsync` écoute les connexions entrantes. Si la valeur n'est pas spécifiée, écoute sur toutes les adresses par défaut.

- port-number** (par défaut : 873)  
Le port TCP sur lequel `rsync` écoute les connexions entrantes. Si le port est inférieur à 1024, `rsync` doit être démarré en tant qu'utilisateur et groupe `root`.
- pid-file** (par défaut : `"/var/run/rsyncd/rsyncd.pid"`)  
Nom du fichier où `rsync` écrit son PID.
- lock-file** (par défaut : `"/var/run/rsyncd/rsyncd.lock"`)  
Nom du fichier où `rsync` écrit son fichier de verrouillage.
- log-file** (par défaut : `"/var/log/rsyncd.log"`)  
Nom du fichier où `rsync` écrit son fichier de journal.
- user** (par défaut : `"root"`)  
Propriétaire du processus `rsync`.
- group** (par défaut : `"root"`)  
Groupe du processus `rsync`.
- uid** (par défaut : `"rsyncd"`)  
Nom d'utilisateur ou ID utilisateur en tant que lequel les transferts de fichiers ont lieu si le démon a été lancé en `root`.
- gid** (par défaut : `"rsyncd"`)  
Nom du groupe ou ID du groupe qui sera utilisé lors de l'accès au module.
- modules** (par défaut : `%default-modules`)  
Liste des « modules » — c.-à-d. des répertoires exportés par `rsync`. Chaque élément doit être un enregistrement `rsync-module`, comme décrit plus bas.

**rsync-module** [Type de données]

C'est le type de données pour les « modules » `rsync`. Un module est un répertoire exporté par le protocole `rsync`. Les champs disponibles sont les suivants :

**name** Le nom du module. C'est le nom qui apparaît dans les URL. Par exemple, si le module se nomme `music`, l'URL correspondante sera `rsync://host.example.org/music`.

**file-name** Nom du répertoire exporté.

**comment** (par défaut : `""`)  
Commentaire associé au module. Les interfaces utilisateurs clientes peuvent l'afficher quand ils récupèrent la liste des modules disponibles.

**read-only?** (par défaut : `#t`)  
Indique si le client pourra envoyer des fichiers. Si la valeur est fausse, les envois seront autorisés si les permissions du côté du démon le permettent.

**chroot?** (par défaut : `#t`)  
Lorsque la valeur est vraie, le démon `rsync` change de racine pour le répertoire du module avant de commencer le transfert de fichiers avec le client. Cela améliore la sécurité, mais nécessite que `rsync` tourne en `root`.



**timeout** (par défaut : 300)

La durée en secondes de silence après laquelle le démon ferme une connexion avec le client.

Le module (**gnu services syncthing**) fournit les services suivant :

Vous pourriez vouloir un démon **syncthing** si vous avez des fichiers sur plusieurs ordinateurs et souhaitez les synchroniser en temps réel, de manière sécurisée sans espionnage possible.

**syncthing-service-type** [Variable Scheme]

C'est le type pour le démon **syncthing** (<https://syncthing.net/>). La valeur de ce service est un enregistrement **syncthing-configuration** comme dans cet exemple :

```
(service syncthing-service-type
  (syncthing-configuration (user "alice")))
```

Voir plus pas pour trouver des détails à propos de **syncthing-configuration**.

**syncthing-configuration** [Type de données]

Type de données représentant la configuration de **syncthing-service**.

**syncthing** (par défaut : *syncthing*)

Le paquet **syncthing** à utiliser.

**arguments** (par défaut : '())

Liste d'arguments de la ligne de commande à passer au binaire **syncthing**.

**logflags** (par défaut : 0)

L'ensemble des drapeaux de journalisation, voir logfiles sur la documentation de Syncthing (<https://docs.syncthing.net/users/syncthing.html#cmdoption-logflags>).

**user** (par défaut : *#f*)

L'utilisateur qui lance le service Syncthing. Cela suppose que l'utilisateur spécifié existe.

**group** (par défaut : "users")

Le groupe qui lance le service Syncthing. Cela suppose que le group spécifié existe.

**home** (par défaut : *#f*)

Configuration et répertoires de données communs. Le répertoire de configuration par défaut est **\$HOME** de l'utilisateur **user** de Syncthing.

En plus, (**gnu services ssh**) fournit les services suivant.

**lsh-service** [*#:host-key "/etc/lsh/host-key"*] [*#:daemonic?*] [Procédure Scheme] [*#: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*]

Lance le programme **lshd** de **lsh** pour écouter sur le port *port-number*. *host-key* doit désigner un fichier contenant la clef d'hôte et ne doit être lisible que par root.

Lorsque *daemonic?* est vrai, `lshd` se détachera du terminal qui le contrôle et enregistrera ses journaux avec `syslogd`, à moins que *syslog-output?* ne soit faux. Évidemment, cela rend aussi `lsh-service` dépendant de l'existence d'un service `syslogd`. Lorsque *pid-file?* est vrai, `lshd` écrit son PID dans le fichier *pid-file*.

Lorsque *initialize?* est vrai, la graine et la clef d'hôte seront créés lors de l'activation du service s'ils n'existent pas encore. Cela peut prendre du temps et demande une interaction.

Lorsque *initialize?* est faux, c'est à l'utilisateur d'initialiser le générateur d'aléatoire (voir Section "lsh-make-seed" dans *LSH Manual*) et de créer une paire de clefs dont la clef privée sera stockée dans le fichier *host-key* (voir Section "lshd basics" dans *LSH Manual*).

Lorsque *interfaces* est vide, `lshd` écoute les connexions sur toutes les interfaces réseau ; autrement, *interfaces* doit être une liste de noms d'hôtes et d'adresses.

*allow-empty-passwords?* spécifie si les connexions avec des mots de passes vides sont acceptés et *root-login?* spécifie si la connexion en root est acceptée.

Les autres options devraient être évidentes.

**openssh-service-type** [Variable Scheme]

C'est le type pour le démon `ssh` OpenSSH (<http://www.openssh.org>), `sshd`. Sa valeur doit être un enregistrement `openssh-configuration` comme dans cet exemple :

```
(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")))))
```

Voir plus bas pour trouver des détails sur `openssh-configuration`.

Ce service peut être étendu avec des clefs autorisées supplémentaires, comme dans cet exemple :

```
(service-extension openssh-service-type
  (const `(("charlie"
            ,(local-file "charlie.pub")))))
```

**openssh-configuration** [Type de données]

C'est l'enregistrement de la configuration de la commande `sshd` d'OpenSSH.

**openssh** (par défaut : *openssh*)  
Le paquet OpenSSH à utiliser.

**pid-file** (par défaut : `"/var/run/sshd.pid"`)  
Nom du fichier où `sshd` écrit son PID.

**port-number** (par défaut : 22)  
Port TCP sur lequel `sshd` écoute les connexions entrantes.

- max-connections** (par défaut : 200)  
Limite du nombre maximum de connexions clientes simultanées, appliquée par le service Shepherd dans le style d'inetd (voir Section "Service De- and Constructors" dans *le manuel de GNU Shepherd*).
- permit-root-login** (par défaut : #f)  
Ce champ détermine si et quand autoriser les connexions en root. Si la valeur est #f, les connexions en root sont désactivées ; si la valeur est #t, elles sont autorisées. S'il s'agit du symbole 'prohibit-password, alors les connexions root sont autorisées mais pas par une authentification par mot de passe.
- allow-empty-passwords?** (par défaut : #f)  
Lorsque la valeur est vraie, les utilisateurs avec un mot de passe vide peuvent se connecter. Sinon, ils ne peuvent pas.
- password-authentication?** (par défaut : #t)  
Lorsque la valeur est vraie, les utilisateurs peuvent se connecter avec leur mot de passe. Sinon, ils doivent utiliser une autre méthode d'authentification.
- public-key-authentication?** (par défaut : #t)  
Lorsque la valeur est vraie, les utilisateurs peuvent se connecter avec leur clef publique. Sinon, les utilisateurs doivent utiliser une autre méthode d'authentification.  
Les clefs publiques autorisées sont stockées dans ~/.ssh/authorized\_keys. Ce n'est utilisé que par le protocole version 2.
- x11-forwarding?** (par défaut : #f)  
Lorsque la valeur est vraie, le transfert de connexion du client graphique X11 est activé — en d'autres termes, les options -X et -Y de ssh fonctionneront.
- allow-agent-forwarding?** (par défaut : #t)  
Indique s'il faut autoriser la redirection d'agent.
- allow-tcp-forwarding?** (par défaut : #t)  
Indique s'il faut autoriser la redirection TCP.
- gateway-ports?** (par défaut : #f)  
Indique s'il faut autoriser les ports de passerelle.
- challenge-response-authentication?** (par défaut : #f)  
Spécifie si l'authentification par défi est autorisée (p. ex. via PAM).
- use-pam?** (par défaut : #t)  
Active l'interface avec le module d'authentification greffable, PAM. Si la valeur est #t, cela activera l'authentification PAM avec challenge-response-authentication? et password-authentication?, en plus des modules de compte et de session de PAM pour tous les types d'authentification.  
Comme l'authentification par défi de PAM sert généralement un rôle équivalent à l'authentification par mot de passe, vous

devriez désactiver soit `challenge-response-authentication?`, soit `password-authentication?`.

`print-last-log?` (par défaut : `#t`)

Spécifie si `sshd` devrait afficher la date et l'heure de dernière connexion des utilisateurs lorsqu'un utilisateur se connecte de manière interactive.

`subsystems` (par défaut : `'(("sftp" "internal-sftp"))`)

Configure les sous-systèmes externes (p. ex. le démon de transfert de fichiers).

C'est une liste de paires, composées chacune du nom du sous-système et d'une commande (avec éventuellement des arguments) à exécuter à la demande du sous-système.

La commande `internal-sftp` met en œuvre un serveur SFTP en cours de traitement. On peut aussi spécifier la commande `sftp-server` :

```
(service openssh-service-type
  (openssh-configuration
    (subsystems
      `(("sftp" ,(file-append openssh "/libexec/sftp-server")))))
```

`accepted-environment` (par défaut : `'()`)

Liste de chaînes de caractères qui décrivent les variables d'environnement qui peuvent être exportées.

Chaque chaîne a sa propre ligne. Voir l'option `AcceptEnv` dans `man sshd_config`.

Cet exemple permet aux clients `ssh` d'exporter la variable `COLORTERM`. Elle est initialisée par les émulateurs de terminaux qui supportent les couleurs. Vous pouvez l'utiliser dans votre fichier de ressource de votre shell pour activer les couleurs sur la ligne de commande si cette variable est initialisée.

```
(service openssh-service-type
  (openssh-configuration
    (accepted-environment '("COLORTERM"))))
```

`authorized-keys` (par défaut : `'()`)

C'est la liste des clefs autorisées. Chaque élément de la liste est un nom d'utilisateur suivi d'un ou plusieurs objets simili-fichiers qui représentent les clefs publiques SSH. Par exemple :

```
(openssh-configuration
  (authorized-keys
    `(("rekado" ,(local-file "rekado.pub"))
      ("chris" ,(local-file "chris.pub"))
      ("root" ,(local-file "rekado.pub") ,(local-file "chris.pub")))))
```

enregistre les clefs publiques spécifiées pour les comptes `rekado`, `chris` et `root`.

Des clefs autorisées supplémentaires peuvent être spécifiées via `service-extension`.

Remarquez que cela n'interfère *pas* avec l'utilisation de `~/.ssh/authorized_keys`.

`generate-host-keys?` (par défaut : `#t`)

Indique s'il faut générer des paires de clés hôtes avec `ssh-keygen -A` dans `/etc/ssh` s'il n'y en a pas.

Générer des paires de clés prend quelques secondes quand assez d'entropie est disponible et n'arrive qu'une fois. Vous pouvez vouloir désactiver cette génération dans une machine virtuelle qui n'en a pas besoin car les clés hôtes sont fournies d'une autre manière, et où le temps de démarrage supplémentaire peut être un problème.

`log-level` (par défaut : `'info`)

C'est le symbole qui spécifie le niveau de journalisation : `quiet`, `fatal`, `error`, `info`, `verbose`, `debug`, etc. Voir la page de manuel de `sshd_config` pour trouver la liste complète des noms de niveaux.

`extra-content` (par défaut : `""`)

Ce champ peut être utilisé pour ajouter un texte arbitraire au fichier de configuration. C'est particulièrement utile pour des configurations élaborées qui ne pourraient pas être exprimées autrement. Cette configuration, par exemple, désactiverait les connexions en root, mais les permettrait depuis une adresse IP spécifique :

```
(openssh-configuration
 (extra-content "\
Match Address 192.168.0.1
  PermitRootLogin yes"))
```

`dropbear-service` [`config`] [Procédure Scheme]

Lance le démon SSH Dropbear (<https://matt.ucc.asn.au/dropbear/dropbear.html>) avec la configuration `config` donnée, un objet `<dropbear-configuration>`.

Par exemple, pour spécifier un service Dropbear qui écoute sur le port 1234, ajoutez cet appel au champ `services` de votre système d'exploitation :

```
(dropbear-service (dropbear-configuration
 (port-number 1234)))
```

`dropbear-configuration` [Type de données]

Ce type de données représente la configuration d'un démon SSH Dropbear.

`dropbear` (par défaut : `dropbear`)

Le paquet Dropbear à utiliser.

`port-number` (par défaut : `22`)

Le port TCP sur lequel le démon attend des connexions entrantes.

`syslog-output?` (par défaut : `#t`)

Indique s'il faut activer la sortie vers syslog.

`pid-file` (par défaut : `"/var/run/dropbear.pid"`)

Nom du fichier de PID du démon.

`root-login?` (par défaut : `#f`)

Indique s'il faut autoriser les connexions en `root`.

`allow-empty-passwords?` (par défaut : `#f`)

Indique s'il faut autoriser les mots de passes vides.

`password-authentication?` (par défaut : `#t`)

Indique s'il faut autoriser l'authentification par mot de passe.

`autossh-service-type`

[Variable Scheme]

C'est le type du programme AutoSSH (<https://www.harding.motd.ca/autossh>) qui exécute une copie de `ssh` et la surveille, en la redémarrant si nécessaire si elle meurt ou arrête le trafic passant. AutoSSH peut être exécuté manuellement à partir de la ligne de commande en passant des arguments au binaire `autossh` du paquet `autossh`, mais peut aussi être exécuté comme un service Guix. Ce dernier cas d'utilisation est documenté ici.

AutoSSH peut être utilisé pour transférer le trafic local vers une machine distante en utilisant un tunnel SSH, et il respecte le `~/.ssh/config` de l'utilisateur sous lequel il est exécuté.

Par exemple, pour spécifier un service exécutant `autossh` comme utilisateur-riche `pino` et transférant toutes les connexions locales au port `8081` vers `remote:8081` en utilisant un tunnel SSH, ajoutez cet appel au champ `services` du système d'exploitation :

```
(service autossh-service-type
  (autossh-configuration
    (user "pino")
    (ssh-options (list "-T" "-N" "-L" "8081:localhost:8081" "remote.net"))))
```

`autossh-configuration`

[Type de données]

Ce type de données représente la configuration du service AutoSSH.

`user` (par défaut : `"autossh"`)

L'utilisateur-riche en tant que responsable du service AutoSSH. Cela suppose que l'utilisateur spécifié existe.

`poll` (par défaut : `600`)

Spécifie le temps de sondage de la connexion en secondes.

`first-poll` (par défaut : `#f`)

Indique le nombre de secondes que l'AutoSSH attend avant le premier test de connexion. Après ce premier test, le vote reprend au rythme défini dans `poll`. Lorsqu'il est défini à `#f`, le premier sondage n'est pas traité spécialement et utilisera également le sondage de connexion spécifié dans `poll`.

`gate-time` (par défaut : `30`)

Spécifie combien de secondes une connexion SSH doit être active avant qu'elle soit considérée comme réussie.

`log-level` (par défaut : `1`)

Le niveau du log, correspondant aux niveaux utilisés par syslog—donc 0 est le plus silencieux tandis que 7 est le plus bavard.

**max-start** (par défaut : **#f**)  
 Le nombre maximum de fois que SSH peut être (re)démarré avant la sortie d'AutoSSH. Lorsqu'il est défini sur **#f**, aucun maximum n'est configuré et AutoSSH peut redémarrer indéfiniment.

**message** (par défaut : "")  
 Le message à ajouter à celui de echo envoyé lors du test des connexions.

**port** (par défaut : "0")  
 Les ports utilisés pour la surveillance de la connexion. Lorsqu'il est défini sur "0", la surveillance est désactivée. Lorsqu'il est défini sur "**n**" où **n** est un entier positif, les ports **n** et **n+1** sont utilisés pour surveiller la connexion, de sorte que le port **n** est le port de surveillance de base et **n+1** est le port d'echo. Lorsqu'ils sont définis sur "**n:m**" où **n** et **m** sont des entiers positifs, les ports **n** et **m** sont utilisés pour surveiller la connexion, de sorte que le port **n** est le port de surveillance de base et **m** est le port d'echo.

**ssh-options** (par défaut : '()'  
 La liste des arguments de la ligne de commande à passer à **ssh** lorsqu'elle est exécutée. Les options **-f** et **-M** sont réservées à AutoSSH et peuvent provoquer un comportement indéfini.

**webssh-service-type** [Variable Scheme]

C'est le type du programme WebSSH (<https://webssh.huashengdun.org/>) qui exécute un client web SSH. WebSSH peut être exécuté manuellement à partir de la ligne de commande en passant des arguments au binaire **wssh** du paquet **webssh**, mais il peut aussi être exécuté comme un service Guix. Ce dernier cas d'utilisation est documenté ici.

Par exemple, pour spécifier un service exécutant WebSSH sur l'interface de bouclage sur le port 8888 avec une politique de rejet avec une liste des hôtes autorisés à se connecter, et NGINX comme reverse-proxy de ce service écoutant la connexion HTTPS, ajouter cet appel au champ **services** du système d'exploitation :

```
(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** [Type de données]

Type de données représentant la configuration pour **webssh-service**.

**package** (par défaut : *webssh*)

Le paquet **webssh** à utiliser.

**user-name** (par défaut : "*webssh*")

Le nom ou l'identifiant de l'utilisateur-riche qui transfère le fichier vers et depuis ce module doit avoir lieu.

**group-name** (par défaut : "*webssh*")

Nom du groupe ou ID du groupe qui sera utilisé lors de l'accès au module.

**address** (par défaut : *#f*)

Adresse IP sur laquelle **webssh** écoute les connexions entrantes.

**port** (par défaut : 8888)

Port TCP sur lequel **webssh** écoute les connexions entrantes.

**policy** (par défaut : *#f*)

Politique de connexion. La politique de *reject* nécessite de spécifier *known-hosts*.

**known-hosts** (par défaut : *'()*)

Liste des hôtes qui ont permis une connexion SSH à partir de **webssh**.

**log-file** (par défaut : *"/var/log/webssh.log"*)

Nom du fichier où **webssh** écrit son fichier de journal.

**log-level** (par défaut : *#f*)

Niveau d'enregistrement.

**%facebook-host-aliases** [Variable Scheme]

Cette variable contient une chaîne de caractères à utiliser dans */etc/hosts* (voir Section "Host Names" dans *The GNU C Library Reference Manual*). Chaque ligne contient une entrée qui fait correspondre les noms des serveurs connus du service en ligne Facebook — p. ex. *www.facebook.com* — à l'hôte local — *127.0.0.1* ou son équivalent en IPv6, *::1*.

Cette variable est typiquement utilisée dans le champ **hosts-file** d'une déclaration **operating-system** (voir Section 12.2 [référence de *operating-system*], page 246) :

```
(use-modules (gnu) (guix))
```

```
(operating-system
```

```
  (host-name "mamachine")
```



```
;; ...
(hosts-file
  ;; Crée un fichier /etc/hosts avec des alias pour « localhost »
  ;; et « mamachine », ainsi que pour les serveurs de Facebook.
  (plain-file "hosts"
    (string-append (local-host-aliases host-name)
      %facebook-host-aliases))))
```

Ce mécanisme peut éviter que des programmes qui tournent localement, comme des navigateurs Web, ne se connectent à Facebook.

Le module (`gnu services avahi`) fournit la définition suivante.

**avahi-service-type** [Variable Scheme]

C'est le service qui lance `avahi-daemon`, un service système qui répond aux requêtes mDNS/DNS-SD qui permet la découverte de services et la recherche de nom en « zéro configuration » (voir <https://avahi.org/>). Sa valeur doit être un enregistrement `avahi-configuration` — voir plus bas.

Ce service étend le démon de cache de services de noms (`nscd`) pour qu'il puisse résoudre les noms d'hôtes en `.local` avec `nss-mdns` (<https://0pointer.de/lennart/projects/nss-mdns/>). Voir Section 12.12 [Name Service Switch], page 573, pour plus d'informations sur la résolution des noms d'hôte.

En plus, cela ajoute le paquet `avahi` au profil du système pour que les commandes comme `avahi-browse` soient directement utilisables.

**avahi-configuration** [Type de données]

Type de données représentant la configuration d'Avahi.

`host-name` (par défaut : `#f`)

Si la valeur n'est pas `#f`, utilise cette valeur comme nom d'hôte à publier pour la machine ; sinon, utilise le vrai nom d'hôte de la machine.

`publish?` (par défaut : `#t`)

Lorsque la valeur est vraie, permet la publication sur le réseau (en diffusion) des noms d'hôtes et des services.

`publish-workstation?` (par défaut : `#t`)

Lorsque la valeur est vraie, `avahi-daemon` publie le nom d'hôte et l'adresse IP de la machine via mDNS sur le réseau local. Pour voir les noms d'hôtes publiés sur votre réseau local, vous pouvez lancer :

```
avahi-browse _workstation._tcp
```

`wide-area?` (par défaut : `#f`)

Lorsque la valeur est vraie, DNS-SD sur DNS unicast est activé.

`ipv4?` (par défaut : `#t`)

`ipv6?` (par défaut : `#t`)

Ces champs déterminent s'il faut utiliser des socket IPv4/IPv6.

`domains-to-browse` (par défaut : `'()`)

C'est la liste des domaines sur lesquels naviguer.

**openvswitch-service-type** [Variable Scheme]

C'est le type de service Open vSwitch (<https://www.openvswitch.org>), dont la valeur devrait être un objet `openvswitch-configuration`.

**openvswitch-configuration** [Type de données]

Type de données représentant la configuration de Open vSwitch, un commutateur virtuel multiniveaux conçu pour rendre possible l'automatisation massive des réseaux avec des extensions programmables.

**package** (par défaut : `openvswitch`)

Objet de paquet de Open vSwitch.

**pagekite-service-type** [Variable Scheme]

Il s'agit du type de service du service PageKite (<https://pagekite.net>), une solution de tunneling permettant de rendre les serveurs locaux visibles au public, même derrière des pare-feu restrictifs ou du NAT sans transfert de ports. La valeur pour ce type de service est un enregistrement `pagekite-configuration`.

Voici un exemple exposant les démons locaux HTTP et SSH :

```
(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** [Type de données]

Type de données représentant la configuration de PageKite.

**package** (par défaut : `pagekite`)

Objet du paquet de PageKite.

**kitename** (par défaut : `#f`)

Nom de PageKite pour l'authentification au serveur frontal.

**kitesecret** (par défaut : `#f`)

Secret partagé pour s'authentifier auprès du serveur frontal. Vous devriez probablement mettre cela dans `extra-file` à la place.

**frontend** (par défaut : `#f`)

Connectez-vous au serveur frontal nommé PageKite au lieu du service `pagekite.net`.

**kites** (par défaut : `'("http:@kitename:localhost:80:@kitesecret")`)

Liste de services kites à utiliser. Expose HTTP sur le port 80 par défaut. Le format est `proto:nomdukite:hôte:port:secret`.

**extra-file** (par défaut : `#f`)

Fichier de configuration supplémentaire à lire, que vous devez créer manuellement. Utilisez-le pour ajouter des options supplémentaires et gérer les secrets partagés hors bande.

**yggdrasil-service-type** [Variable Scheme]

Le type de service pour se connecter au réseau Yggdrasil (<https://yggdrasil-network.github.io/>), une jeune implémentation d'un réseau IPv6 entièrement chiffré de bout en bout.

Yggdrasil fournit le routage indépendant du nom avec des adresses générées de manière cryptographiques. L'adressage statique signifie que vous pouvez garder la même adresse aussi longtemps que vous le souhaitez, même si vous changez d'emplacement, et que vous pouvez générer une nouvelle adresse (en générant de nouvelles clés) quand vous le souhaitez. <https://yggdrasil-network.github.io/2018/07/28/addressing.html>

Passez-lui une valeur de type `yggdrasil-configuration` pour le connecter aux pairs publics ou aux pairs locaux.

voici un exemple qui utilise des pairs publics et une adresse statique. Les clés de signature et de chiffrement statiques sont définies dans `/etc/yggdrasil-private.conf` (la valeur par défaut de `config-file`).

```
;; extrait d'une déclaration de système d'exploitation
(service yggdrasil-service-type
  (yggdrasil-configuration
    (autoconf? #f) ;; utiliser uniquement les pairs publics
    (json-config
      ;; choisir parmi
      ;; https://github.com/yggdrasil-network/public-peers
      '((peers . #("tcp://1.2.3.4:1337")))
      ;; /etc/yggdrasil-private.conf est la valeur par défaut pour confi
    ))
  # exemple de contenu de /etc/yggdrasil-private.conf
  {
    # Votre clé publique. Vos pairs peuvent vous la demander pour la
    # mettre dans leur configuration AllowedPublicKeys.
    PublicKey: 64277...

    # Votre clé privée. NE la partagez avec PERSONNE !
    PrivateKey: 5c750...
  }
```

**yggdrasil-configuration** [Type de données]

Type de données qui représente la configuration de Yggdrasil.

`package` (par défaut : `yggdrasil`)

Objet du paquet de Yggdrasil.

`json-config` (par défaut : `'()`)

Contenu de `/etc/yggdrasil.conf`. Sera fusionné avec `/etc/yggdrasil-private.conf`. Remarquez que ces paramètres sont stockés dans le dépôt de Guix, qui est disponible en lecture pour tout le monde. **Ne stockez pas vos clés privées dedans.** Voir la sortie de `yggdrasil -genconf` pour un aperçu rapide de ce que sont des clés valides et leurs valeurs par défaut.

- autoconf?** (par défaut : `#f`)  
Indique s'il faut utiliser le mode automatique. L'activer fera utiliser une IP dynamique à Yggdrasil et l'appairer à ses voisins IPv6.
- log-level** (par défaut : `'info`)  
La quantité de détails à inclure dans les journaux. Utilisez `'debug` pour plus de détails.
- log-to** (par défaut : `'stdout`)  
L'emplacement où envoyer les journaux. Par défaut, le service enregistre la sortie standard dans `/var/log/yggdrasil.log`. L'alternative est `'syslog`, qui envoie la sortie au service `syslog`.
- config-file** (par défaut : `"/etc/yggdrasil-private.conf"`)  
Le fichier HJSON depuis lequel charger les données sensibles. C'est l'emplacement où stocker les clés privées, qui doivent être spécifiées si vous ne voulez pas une adresse aléatoire à chaque démarrage. Utilisez `#f` pour désactiver. Les options définies dans ce fichier prennent le pas sur `json-config`. Utilisez la sortie de `yggdrasil -genconf` pour commencer. Pour configurer une adresse statique, supprimez tout sauf ces options :
- `EncryptionPublicKey`
  - `EncryptionPrivateKey`
  - `SigningPublicKey`
  - `SigningPrivateKey`

**ipfs-service-type** [Variable Scheme]  
Le type de service pour se connecter au réseau IPFS (<https://ipfs.io>), un système de fichiers mondial, versionné et en pair-à-pair. Passez lui un objet `ipfs-configuration` pour changer les ports utilisés pour la passerelle et l'API.

Voici un exemple de configuration, avec des ports non standards :

```
(service ipfs-service-type
  (ipfs-configuration
    (gateway "/ip4/127.0.0.1/tcp/8880")
    (api "/ip4/127.0.0.1/tcp/8881")))
```

**ipfs-configuration** [Type de données]  
Type de données représentant la configuration d'IPFS.

**package** (par défaut : `go-ipfs`)  
Objet du paquet d'IPFS.

**gateway** (par défaut : `"/ip4/127.0.0.1/tcp/8082"`)  
Adresse de la passerelle, au format « multiaddress ».

**api** (par défaut : `"/ip4/127.0.0.1/tcp/5001"`)  
Adresse du point d'accès de l'API, au format « multiaddress ».

**keepalived-service-type** [Variable Scheme]

C'est le type pour le logiciel de routage Keepalived (<https://www.keepalived.org/>), `keepalived`. Sa valeur doit être un enregistrement `keepalived-configuration` comme dans cet exemple pour la machine maître :

```
(service keepalived-service-type
  (keepalived-configuration
    (config-file (local-file "keepalived-master.conf"))))
```

où `keepalived-master.conf` :

```
vrp_instance my-group {
  state MASTER
  interface enp9s0
  virtual_router_id 100
  priority 100
  unicast_peer { 10.0.0.2 }
  virtual_ipaddress {
    10.0.0.4/24
  }
}
```

et pour la machine de secours :

```
(service keepalived-service-type
  (keepalived-configuration
    (config-file (local-file "keepalived-backup.conf"))))
```

où `keepalived-backup.conf` :

```
vrp_instance my-group {
  state BACKUP
  interface enp9s0
  virtual_router_id 100
  priority 99
  unicast_peer { 10.0.0.3 }
  virtual_ipaddress {
    10.0.0.4/24
  }
}
```

### 12.9.6 Mises à jour non surveillées (Unattended Upgrades)

Guix fournit un service pour effectuer des *unattended upgrades* : périodiquement, le système se reconfigure automatiquement à partir du dernier Guix. Guix System possède plusieurs propriétés qui rendent les mises à jour non surveillées sûres :

- les mises à jour sont transactionnelles (soit la mise à jour réussit, soit elle échoue, mais vous ne pouvez pas vous retrouver avec un état système "intermédiaire") ;
- le journal de mise à niveau est kept— vous pouvez le consulter avec `guix system list-generations`—et vous pouvez revenir à n'importe quelle génération précédente, si le système mis à jour ne se comporte pas comme prévu ;
- le code du canal est authentifié, vous savez donc que vous ne pouvez exécuter que du code authentique (voir Chapitre 7 [Canaux], page 75) ;

- **guix system reconfigure** empêche les déclassements, ce qui le rend immunisé contre les attaques *downgrade attacks*.

Pour mettre en place des mises à jour sans surveillance, ajoutez une instance de **unattended-upgrade-service-type** comme celle ci-dessous, à la liste des services de votre système d'exploitation :

```
(service unattended-upgrade-service-type)
```

Les valeurs par défaut ci-dessus mettent en place des mises à jour hebdomadaires : tous les dimanches à minuit. Il n'est pas nécessaire de fournir le fichier de configuration du système d'exploitation : il utilise `/run/current-system/configuration.scm`, ce qui garantit qu'il utilise toujours votre dernière configuration—voir [provenance-service-type], page 606, pour plus d'informations sur ce fichier.

Plusieurs choses peuvent être configurés, notamment la périodicité et les services (démons) à redémarrer à la fin. Lorsque la mise à jour est réussie, le service se charge de supprimer les générations de systèmes plus anciennes qu'un certain seuil, conformément à **guix system delete-generations**. Voir la référence ci-dessous pour plus de détails.

Pour vous assurer que les mises à jour sont bien effectuées, vous pouvez exécuter **guix system describe**. Pour enquêter sur les échecs de mise à jour, consultez le fichier journal des mises à jour non surveillées (voir ci-dessous).

**unattended-upgrade-service-type** [Variable Scheme]

C'est le type de service pour les mises à jour sans surveillance. Il met en place un job mcron (voir Section 12.9.2 [Exécution de tâches planifiées], page 287) qui exécute **guix system reconfigure** à partir de la dernière version des canaux spécifiés.

Sa valeur doit être un enregistrement **unattended-upgrade-configuration** (voir ci-dessous).

**unattended-upgrade-configuration** [Type de données]

Ce type de données représente la configuration du service de mise à jour non surveillée. Les champs suivants sont disponibles :

**schedule** (par défaut : "30 01 \* \* 0")

Il s'agit du calendrier des mises à jour, exprimé sous la forme d'une gexp contenant un calendrier des travaux mcron (voir Section "Guile Syntax" dans *GNU mcron*).

**channels** (par défaut : `#%default-channels`)

Ce gexp spécifie les canaux à utiliser pour la mise à jour (voir Chapitre 7 [Canaux], page 75). Par défaut, l'extrémité du canal officiel **guix** est utilisée.

**operating-system-file** (par défaut :

`"/run/current-system/configuration.scm"`)

Ce champ indique le fichier de configuration du système d'exploitation à utiliser. La valeur par défaut est de réutiliser le fichier de configuration de la configuration actuelle.

Il y a cependant des cas où la référence à `/run/current-system/configuration.scm` n'est pas suffisante, par exemple parce que

ce fichier fait référence à des fichiers supplémentaires (clés publiques SSH, fichiers de configuration supplémentaires, etc.) *via* `local-file` et constructions similaires. Pour ces cas, nous recommandons quelque chose qui va dans ce sens :

```
(unattended-upgrade-configuration
  (operating-system-file
    (file-append (local-file "." "config-dir" #:recursive? #t)
      "/config.scm")))
```

L'effet ici est d'importer tout le répertoire courant dans le dépôt, et de se référer à `config.scm` dans ce répertoire. Par conséquent, l'utilisation de `local-file` dans `config.scm` fonctionnera comme prévu. Voir Section 9.12 [G-Expressions], page 166, pour plus d'informations sur `local-file` et `file-append`.

`services-to-restart` (par défaut : '(mcron))

Ce champ indique les services Shepherd à redémarrer lorsque la mise à jour est terminée.

Ces services sont redémarrés dès qu'ils sont terminés, comme avec `herd restart`, ce qui garantit que la dernière version fonctionne—souvenez-vous que par défaut `guix system reconfigure` ne redémarre que les services qui ne fonctionnent pas actuellement, ce qui est conservateur : cela minimise les perturbations mais laisse fonctionner les services obsolètes.

Utilisez `herd status` pour trouver les candidats au redémarrage. Voir Section 12.9 [Services], page 267, pour des informations générales sur les services. Les services courants à redémarrer sont `ntpd` et `ssh-daemon`.

Par défaut, le service `mcron` est redémarré. Cela garantit que la dernière version du travail de mise à niveau non surveillée sera utilisée la prochaine fois.

`system-expiration` (par défaut : (\* 3 30 24 3600))

C'est le temps d'expiration en secondes pour les générations du système. Les générations du système plus anciennes que ce délai sont supprimées avec `guix system delete-generations` lorsqu'une mise à jour se termine

.

**Remarque:** Le service de mise à jour sans surveillance ne fait pas fonctionner le ramasse miettes. Vous voudrez probablement configurer votre propre job `mcron` pour exécuter `guix gc` périodiquement.

`minimum-duration` (par défaut : 3600)

Durée maximale en secondes de la mise à jour ; passé ce délai, la mise à jour est interrompue.

Cela est surtout utile pour s'assurer que la mise à jour ne finira pas par reconstruire ou retélécharger « le monde ».

`log-file` (par défaut : "/var/log/unattended-upgrade.log")

Fichier où sont consignées les mises à jour non surveillées.

### 12.9.7 Système de fenêtrage X

La prise en charge du système d'affichage graphique X Window — en particulier Xorg — est fournie par le module (`gnu services xorg`). Remarquez qu'il n'y a pas de procédure `xorg-service`. À la place, le serveur X est démarré par le *gestionnaire de connexion*, par défaut le gestionnaire d'affichage de GNOME (GDM).

GDM permet évidemment aux utilisateurs de se connecter et d'ouvrir un gestionnaire de fenêtre ou un gestionnaire d'environnement autre que GNOME ; pour ceux qui utilisent GNOME, GDM est requis pour certaines fonctionnalités comme l'écran de verrouillage automatique.

Pour utiliser X11, vous devez installer au moins un *gestionnaire de fenêtre* — par exemple les paquets `windowmaker` ou `openbox` — de préférence en l'ajoutant au champ `packages` de votre définition de système d'exploitation (voir Section 12.2 [référence de operating-system], page 246).

GDM prend aussi en charge Wayland : il peut lui-même utiliser Wayland au lieu de X11 pour son interface utilisateur, et il peut aussi démarrer des sessions Wayland. Le premier est requis pour le second. Pour l'activer indiquez `wayland?` à `#t` dans `gdm-configuration`.

`gdm-service-type` [Variable Scheme]

C'est le type pour le GNOME Desktop Manager (<https://wiki.gnome.org/Projects/GDM/>) (GDM), un programme qui gère les serveurs d'affichage graphique et qui gère les connexions graphiques des utilisateur-*rice-s*. Sa valeur doit être un `gdm-configuration` (voir ci-dessous).

GDM cherche des *types de sessions* définies par les fichiers `.desktop` dans `/run/current-system/profile/share/xsessions` (pour les session X11) et `/run/current-system/profile/share/wayland-sessions` (pour les sessions Wayland) et permet aux utilisateurs de choisir une session depuis l'écran de connexion. Les paquets comme `gnome`, `xfce`, `i3` et `sway` fournissent des fichiers `.desktop` ; les ajouter à l'ensemble des paquets du système les rendra automatiquement disponibles sur l'écran de connexion.

En plus, les fichiers `~/.xsession` sont pris en compte. Lorsqu'il est disponible, `~/.xsession` doit être un fichier exécutable qui démarre un gestionnaire de fenêtre au un autre client X.

`gdm-configuration` [Type de données]

`auto-login?` (par défaut : `#f`)

`default-user` (par défaut : `#f`)

Lorsque `auto-login?` est faux, GDM présente un écran de connexion.

Lorsque `auto-login?` est vrai, GDM se connecte directement en tant que `default-user`.

`auto-suspend?` (par défaut : `#t`)

Lorsque la valeur est vraie, GDM passera en veille lorsque personne n'est connecté physiquement. Si votre machine peut être utilisée avec un bureau à distance ou SSH, cette option devrait être mise à faux pour éviter que GDM n'interrompe les sessions distantes ou rende la machine indisponible.



- `debug?` (par défaut : `#f`)  
Lorsqu'il est « vrai », GDM écrit des messages de débogage dans son journal.
- `gnome-shell-assets` (par défaut : `...`)  
Liste de données requises par GDM : un thème d'icônes, des polices, etc.
- `xorg-configuration` (par défaut : `(xorg-configuration)`)  
Configuration du serveur graphique Xorg.
- `x-session` (par défaut : `xinitrc`)  
Le script à lancer avant de démarrer une session X.
- `xdmcp?` (par défaut : `#f`)  
Lorsque la valeur est vraie, active le protocole de contrôle du gestionnaire d'affichage X (XDMCP). Cela ne devrait être activé que dans des environnements de confiance, car le protocole n'est pas sécurisé. Lorsque l'option est activée, GDM attend des requêtes XDMCP sur le port UDP 177.
- `dbus-daemon` (par défaut : `dbus-daemon-wrapper`)  
Nom du fichier de l'exécutable `dbus-daemon`.
- `gdm` (par défaut : `gdm`)  
Le paquet GDM à utiliser.
- `wayland?` (par défaut : `#f`)  
Lorsque la valeur est vraie, active Wayland dans GDM, ce qui est requis pour utiliser les sessions Wayland.
- `wayland-session` (par défaut : `gdm-wayland-session-wrapper`)  
L'enveloppe de session Wayland à utiliser, requise pour mettre en place l'environnement.

### `slim-service-type` [Variable Scheme]

C'est de type pour le gestionnaire de connexion graphique SLiM pour X11.

Comme GDM, SLiM recherche des types de sessions décrites par des fichiers `.desktop` et permet aux utilisateurs de choisir une session à partir de l'écran de connexion avec `F1`. Il prend aussi en compte les fichiers `~/xsession`.

Contrairement à GDM, SLiM ne démarre pas la session utilisateur sur un terminal virtuel différent à la connexion, ce qui signifie que vous ne pouvez démarrer qu'une seule session graphique. Si vous voulez pouvoir exécuter plusieurs sessions graphiques en même temps, vous devez ajouter plusieurs services SLiM à la liste des services de votre système. L'exemple suivant montre comment remplacer le service GDM par défaut par deux services SLiM sur les tty 7 et 8.

```
(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` [Type de données]

Type de données représentant la configuration de `slim-service-type`.

`allow-empty-passwords?` (par défaut : `#t`)

S'il faut autoriser les connexions avec un mot de passe vide.

`gnupg?` (par défaut : `#f`)

Si l'option est activée, `pam-gnupg` essaiera automatique de débloquer les clés GPG de l'utilisateur-riche avec le mot de passe de connexion via `gpg-agent`. Les keygrips de toutes les clés à débloquer doivent se trouver dans `~/.pam-gnupg` et vous pouvez les récupérer avec `gpg -K --with-keygrip`. Vous pouvez activer le préchargement des phrases de passes en ajoutant `allow-preset-passphrase` dans `~/.gnupg/gpg-agent.conf`.

`auto-login?` (par défaut : `#f`)

`default-user` (par défaut : `""`)

Lorsque `auto-login?` est faux, SLiM présente un écran de connexion.

Lorsque `auto-login?` est vrai, SLiM se connecte directement en tant que `default-user`.

`theme` (par défaut : `%default-slim-theme`)

`theme-name` (par défaut : `%default-slim-theme-name`)

Le thème graphique à utiliser et son nom.

`auto-login-session` (par défaut : `#f`)

Si la valeur est vraie, elle doit être le nom d'un exécutable à démarrer comme session par défaut — p. ex. `(file-append windowmaker "/bin/windowmaker")`.

Si la valeur est fautive, une session décrite par l'un des fichiers `.desktop` disponibles dans `/run/current-system/profile` et `~/.guix-profile` sera utilisée.

**Remarque:** Vous devez installer au moins un gestionnaire de fenêtres dans le profil du système ou dans votre profil utilisateur. Sinon, si `auto-login-session` est faux, vous ne serez jamais capable de vous connecter.

`xorg-configuration` (par défaut : `(xorg-configuration)`)

Configuration du serveur graphique Xorg.

`display` (par défaut : `":0"`)

La session d'affichage sur laquelle démarrer le serveur graphique Xorg.

**vt** (par défaut : `"vt7"`)  
Le terminal virtuel sur lequel démarrer le serveur d’affichage graphique Xorg.

**xauth** (par défaut : `xauth`)  
Le paquet XAuth à utiliser.

**shepherd** (par défaut : `shepherd`)  
Le paquet Shepherd à utiliser pour invoquer `halt` et `reboot`.

**sessreg** (par défaut : `sessreg`)  
Le paquet `sessreg` à utiliser pour enregistrer la session.

**slim** (par défaut : `slim`)  
Le paquet SLiM à utiliser.

**%default-theme** [Variable Scheme]

**%default-theme-name** [Variable Scheme]

Le thème SLiM par défaut et son nom.

**sddm-service-type** [Variable Scheme]

C’est le type de service qui permet d’exécuter le gestionnaire d’affichage Gestionnaire d’affichage SDDM (<https://github.com/sddm/sddm>). Sa valeur doit être un enregistrement `sddm-configuration` (voir ci-dessous).

Voici un exemple d’utilisation :

```
(service sddm-service-type
  (sddm-configuration
    (auto-login-user "alice")
    (auto-login-session "xfce.desktop")))
```

**sddm-configuration** [Type de données]

Ce type de données représente la configuration du gestionnaire de connexion SDDM. Les champs disponibles sont :

**sddm** (par défaut : `sddm`)  
Le paquet SDDM à utiliser.

**display-server** (par défaut : `"x11"`)  
Choisit le serveur d’affichage à utiliser pour l’écran d’accueil. Les valeurs valides sont `"x11"` ou `"wayland"`.

**numlock** (par défaut : `"on"`)  
Les valeurs valides sont `"on"`, `"off"` or `"none"`.

**halt-command** (par défaut : `#~(string-append #$$shepherd "/sbin/halt")`)  
La commande à lancer à l’arrêt du système.

**reboot-command** (par défaut : `#~(string-append #$$shepherd "/sbin/reboot")`)  
La commande à lancer lors du redémarrage du système.

**theme** (par défaut : `"maldives"`)  
Thème à utiliser. Les thèmes par défaut fournis par SDDM sont `"elarun"`, `"maldives"` ou `"maya"`.

**themes-directory** (par défaut :  
"/run/current-system/profile/share/sddm/themes")  
Le répertoire où se trouvent les thèmes.

**faces-directory** (par défaut : "/run/current-system/profile/share/sddm/faces")  
Répertoire où se trouvent les avatars.

**default-path** (par défaut : "/run/current-system/profile/bin")  
Le PATH par défaut à utiliser.

**minimum-uid** (par défaut : 1000)  
UID minimum affiché dans le SDDM et autorisé pour la connexion.

**maximum-uid** (par défaut : 2000)  
UID maximum pour être affiché dans SDDM.

**remember-last-user?** (par défaut : #t)  
S'il faut se rappeler le dernier utilisateur connecté.

**remember-last-session?** (par défaut : #t)  
S'il faut se rappeler la dernière session.

**hide-users** (par défaut : "")  
Les noms d'utilisateurs à cacher sur l'écran d'accueil de SDDM.

**hide-shells** (par défaut : #~(string-append #shadow "/sbin/nologin"))  
Les utilisateurs avec les shells listés seront cachés sur l'écran d'accueil de SDDM.

**session-command** (par défaut : #~(string-append #sddm  
"/share/sddm/scripts/wayland-session"))  
Le script à lancer avant de démarrer une session wayland.

**sessions-directory** (par défaut :  
"/run/current-system/profile/share/wayland-sessions")  
Le répertoire où trouver les fichiers .desktop qui démarrent des sessions wayland.

**xorg-configuration** (par défaut : (xorg-configuration))  
Configuration du serveur graphique Xorg.

**xauth-path** (par défaut : #~(string-append #xauth "/bin/xauth"))  
Chemin vers xauth.

**xephyr-path** (par défaut : #~(string-append #xorg-server "/bin/Xephyr"))  
Chemin vers Xephyr.

**xdisplay-start** (par défaut : #~(string-append #sddm  
"/share/sddm/scripts/Xsetup"))  
Le script à lancer après avoir démarré xorg-server.

**xdisplay-stop** (par défaut : #~(string-append #sddm  
"/share/sddm/scripts/Xstop"))  
Le script à lancer avant d'arrêter xorg-server.

**xsession-command** (par défaut : xinitrc)  
Le script à lancer avant de démarrer une session X.

`xsessions-directory` (par défaut : `"/run/current-system/profile/share/xsessions"`)  
Répertoire où trouver les fichiers `.desktop` pour les sessions X.

`minimum-vt` (par défaut : 7)  
VT minimal à utiliser.

`auto-login-user` (par défaut : `"`)  
Compte utilisateur qui sera automatiquement connecté. Paramétrer ceci à la chaîne vide désactive la connexion automatique.

`auto-login-session` (par défaut : `"`)  
Nom du fichier `.desktop` utilisé comme session pour la connexion automatique, ou la chaîne vide.

`relogin?` (par défaut : `#f`)  
S'il faut se reconnecter après la déconnexion.

`lightdm-service-type` [Variable Scheme]

C'est le type de service qui lance le gestionnaire d'affichage LightDM (<https://github.com/canonical/lightdm>). Sa valeur doit être un enregistrement `lightdm-configuration`, qui est documenté ci-dessous. Entre autres particularité, il possède une intégration avec TigerVNC pour accéder à votre bureau à distance ainsi que la prise en charge du protocole XDMCP, qui peut être utilisé par les clients distants pour démarrer une session à partir du gestionnaire de connexion.

Dans sa forme la plus basique, il peut s'utiliser de cette manière :

```
(service lightdm-service-type)
```

Un exemple plus élaboré qui utilise la fonctionnalité VNC et active plus de fonctionnalités et les journaux verbeux ressemblerait à ceci :

```
(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` [Type de données]

Les champs de `lightdm-configuration` disponibles sont :

`lightdm` (par défaut : `lightdm`) (type : simili-fichier)  
Le paquet `lightdm` à utiliser.

`allow-empty-passwords?` (par défaut : `#f`) (type : booléen)  
Indique si les utilisateurs et utilisatrices sans mot de passe peuvent se connecter.

`debug?` (par défaut : `#f`) (type : booléen)

Active la sortie verbeuse.

`xorg-configuration` (type : `xorg-configuration`)

La configuration par défaut du serveur Xorg à utiliser pour générer le script de démarrage du serveur Xorg. Elle peut être raffinée pour chaque siège via le champ `xserver-command` de l'enregistrement `<lightdm-seat-configuration>` si vous le souhaitez.

`greeters` (type : `liste-de-greeter-configurations`)

Les configurations de l'écran d'accueil de LightDM spécifiant les écrans d'accueil à utiliser.

`seats` (type : `liste-de-seat-configurations`)

Les configurations de sièges à utiliser. Un siège LightDM correspond à un utilisateur ou une utilisatrice.

`xdmcp?` (par défaut : `#f`) (type : booléen)

Indique si le serveur XDMCP doit écouter sur le port UDP 177.

`xdmcp-listen-address` (type : `peut-être-chaine`)

L'hôte ou l'adresse IP sur laquelle le serveur XDMCP écoute les connexions entrantes. Si la valeur n'est pas spécifiée, écoute sur tous les hôtes et toutes les adresses.

`vnc-server?` (par défaut : `#f`) (type : booléen)

Si un serveur VNC est démarré.

`vnc-server-command` (type : `simili-fichier`)

La commande Xvnc à utiliser pour le serveur VNC, il est possible de fournir des options supplémentaires qui ne sont sinon pas exposées avec la commande, par exemple pour désactiver la sécurité :

```
(vnc-server-command (file-append tigervnc-server "/bin/Xvnc"
                  "-SecurityTypes None" ))
```

Ou pour indiquer un fichier de mot de passe pour le mécanisme VncAuth classique (non sécurisé) :

```
(vnc-server-command (file-append tigervnc-server "/bin/Xvnc"
                  "-PasswordFile /var/lib/lightdm/.vnc/
```

Le fichier de mots de passe devrait être créé manuellement avec la commande `vncpasswd`. Remarquez que LightDM créera de nouvelles sessions pour les utilisateurs et les utilisatrices VNC, ce qui signifie qu'ils et elles devront s'authentifier de la même manière que les utilisateurs locaux et utilisatrices locales.

`vnc-server-listen-address` (type : `peut-être-chaine`)

L'hôte ou l'adresse IP sur laquelle le serveur VNC écoute les connexions entrantes. Si la valeur n'est pas spécifiée, écoute sur tous les hôtes et toutes les adresses.

`vnc-server-port` (par défaut : `5900`) (type : `nombre`)

Port TCP sur lequel écoute le serveur VNC.

`extra-config` (par défaut : ()) (type : liste-de-chaines)  
Valeurs de configuration supplémentaires à ajouter à la fin du fichier de configuration de LightDM.

`lightdm-gtk-greeter-configuration` [Type de données]  
Les champs de `lightdm-gtk-greeter-configuration` disponibles sont :

`lightdm-gtk-greeter` (par défaut : `lightdm-gtk-greeter`) (type : simili-fichier)  
Le paquet `lightdm-gtk-greeter` à utiliser.

`assets` (par défaut :  
(`adwaita-icon-themegnome-themes-extrahicolor-icon-theme`)) (type :  
liste-de-simili-fichiers)  
La liste des paquets qui accompagnent l'écran d'accueil, comme un paquet qui fournit un thème d'icônes.

`theme-name` (par défaut : "Adwaita") (type : chaîne)  
Le nom du thème à utiliser.

`icon-theme-name` (par défaut : "Adwaita") (type : chaîne)  
Le nom du thème d'icônes à utiliser.

`cursor-theme-name` (par défaut : "Adwaita") (type : chaîne)  
Le nom du thème de curseurs à utiliser.

`cursor-theme-size` (par défaut : 16) (type : nombre)  
La taille à utiliser pour le thème de curseurs.

`allow-debugging?` (type : peut-être-booléen)  
Indiquez `#t` pour activer le niveau de journalisation de débogage.

`background` (type : simili-fichier)  
L'image de fond à utiliser.

`at-spi-enables?` (par défaut : `#f`) (type : booléen)  
Active la prise en charge de l'accessibilité à travers l'interface de fourniture de service de technologie d'assistance (AT-SPI).

`a11y-states` (par défaut : (`contrast font keyboard reader`))  
(type : liste-d'états-d'accessibilité) Les fonctionnalités d'accessibilité à activer, données en tant que liste de symboles.

`reader` (type : peut-être-simili-fichier)  
La commande à utiliser pour lancer un lecteur d'écran.

`extra-config` (par défaut : ()) (type : liste-de-chaines)  
Valeurs de configuration supplémentaires à ajouter à la fin du fichier de configuration de l'écran d'accueil de LightDM.

`lightdm-seat-configuration` [Type de données]  
Les champs de `lightdm-seat-configuration` disponibles sont :

`name` (type : nom-de-siège)  
Le nom du siège. Vous pouvez utiliser un astérisque (\*) dans le nom pour appliquer la configuration de siège à tous les noms de sièges qui correspondent.

- user-session** (type : peut-être-chaine)  
La session à utiliser par défaut. Le nom de session doit être fourni en une chaîne en minuscules, comme "gnome", "ratpoison", etc.
- type** (par défaut : local) (type : type-de-siège)  
Le type du siège, soit le symbole `local`, soit le symbole `xremote`.
- autologin-user** (type : peut-être-chaine)  
Le nom d'utilisateur à connecter par défaut.
- greeter-session** (par défaut : `lightdm-gtk-greeter`) (type : greeter-session)  
La session d'écran d'accueil à utilisé, spécifié en tant que symbole. Actuellement, seul `lightdm-gtk-greeter` est pris en charge.
- xserver-command** (type : peut-être-simili-fichier)  
La commande du serveur Xorg à lancer.
- session-wrapper** (type : simili-fichier)  
L'enveloppe de session `xinitrc` à utiliser.
- extra-config** (par défaut : ()) (type : liste-de-chaines)  
Valeurs de configuration supplémentaires à ajouter à la fin de la section de configuration des sièges.

**xorg-configuration** [Type de données]

Ce type de données représente la configuration du serveur d'affichage graphique Xorg. Remarquez qu'il n'y a pas de service Xorg ; à la place, le serveur X est démarré par un « gestionnaire d'affichage graphique » comme GDM, SDDM, LightDM et SLiM. Ainsi, la configuration de ces gestionnaires d'affichage agrègent un enregistrement `xorg-configuration`.

- modules** (par défaut : `%default-xorg-modules`)  
C'est une liste de *paquets de module* chargés par le serveur Xorg — p. ex. `xf86-video-vesa`, `xf86-input-keyboard` etc.
- fonts** (par défaut : `%default-xorg-fonts`)  
C'est une liste de répertoires de polices à ajouter au *chemin de polices* du serveur.
- drivers** (par défaut : '()')  
Cela doit être soit la liste vide, auquel cas Xorg choisit un pilote graphique automatiquement, soit une liste de noms de pilotes qui seront essayés dans cet ordre — p. ex. ("`modestring`" "`vesa`").
- resolutions** (par défaut : '()')  
Lorsque `resolutions` est la liste vide, Xorg choisit une résolution d'écran appropriée. Sinon, il doit s'agir d'une liste de résolutions — p. ex. ((1024 768) (640 480)).
- keyboard-layout** (par défaut : `#f`)  
Si la valeur est `#f`, Xorg utilise la disposition du clavier par défaut — habituellement la disposition anglaise américaine (« `qwerty` ») pour un clavier de PC à 105 touches.



Sinon cela doit être un objet `keyboard-layout` spécifiant la disposition du clavier à utiliser lorsque Xorg tourne. Voir Section 12.7 [Disposition du clavier], page 263, pour plus d'informations sur la manière de spécifier la disposition du clavier.

`extra-config` (par défaut : ' ( ) )

C'est une liste de chaînes de caractères ou d'objets ajoutés au fichier de configuration. Elle est utile pour ajouter du texte supplémentaire directement dans le fichier de configuration.

`server` (par défaut : `xorg-server`)

C'est le paquet fournissant le serveur Xorg.

`server-arguments` (par défaut : `%default-xorg-server-arguments`)

Liste d'arguments de la ligne de commande supplémentaires à passer au serveur X. La valeur par défaut est `-nolisten tcp`.

`set-xorg-configuration config` [Procédure Scheme]  
[*login-manager-service-type*]

Dit au gestionnaire de connexion (de type *login-manager-service-type*) d'utiliser *config*, un enregistrement `<xorg-configuration>`.

Comme la configuration Xog est incluse dans la configuration du gestionnaire de connexion — p. ex. `gdm-configuration` — cette procédure fournit un raccourci pour configurer Xorg.

`xorg-start-command [config]` [Procédure Scheme]

Renvoie un script `startx` dans lequel les modules, les polices, etc., spécifiés dans *config* sont disponibles. Le résultat devrait être utilisé à la place de `startx`.

Habituellement le serveur X est démarré par un gestionnaire de connexion.

`screen-locker-service package [program]` [Procédure Scheme]

Ajoute *package*, un paquet pour un verrouiller l'écran ou un économiseur d'écran dont la commande est *program*, à l'ensemble des programmes `setuid` et lui ajoute une entrée PAM. Par exemple :

```
(screen-locker-service xlockmore "xlock")
```

rend utilisable le bon vieux XlockMore.

### 12.9.8 Services d'impression

Le module (`gnu services cups`) fournit une définition de service Guix pour le service d'impression CUPS. Pour ajouter la prise en charge d'une imprimante à un système Guix, ajoutez un `cups-service` à la définition du système d'exploitation :

`cups-service-type` [Variable Scheme]

Le type de service pour un serveur d'impression CUPS. Sa valeur devrait être une configuration CUPS valide (voir plus bas). Pour utiliser les paramètres par défaut, écrivez simplement :

```
(service cups-service-type)
```

La configuration de CUPS contrôle les paramètres de base de votre installation CUPS : sur quelles interfaces il doit écouter, que faire si un travail échoue, combien de journalisation il faut faire, etc. Pour ajouter une imprimante, vous devrez visiter l'URL `http://localhost:631` ou utiliser un outil comme les services de configuration d'imprimante de GNOME. Par défaut, la configuration du service CUPS générera un certificat auto-signé si besoin, pour les connexions sécurisée avec le serveur d'impression.

Supposons que vous souhaitiez activer l'interface Web de CUPS et ajouter le support pour les imprimantes Epson via le paquet `epson-inkjet-printer-escpr` et pour les imprimantes HP via le paquet `hplip-minimal`. Vous pouvez le faire directement, comme ceci (vous devez utiliser le module `(gnu packages cups)`) :

```
(service cups-service-type
  (cups-configuration
    (web-interface? #t)
    (extensions
      (list cups-filters epson-inkjet-printer-escpr hplip-minimal))))
```

**Remarque:** Si vous souhaitez utiliser la GUI basée sur Qt5 qui provient du paquet `hplip`, nous vous suggérons d'installer le paquet `hplip`, soit dans votre configuration d'OS, soit en tant qu'utilisateur.

Les paramètres de configuration disponibles sont les suivants. Chaque définition des paramètres est précédé par son type ; par exemple, `'string-list toto'` indique que le paramètre `toto` devrait être spécifié comme une liste de chaînes de caractères. Il y a aussi une manière de spécifier la configuration comme une chaîne de caractères, si vous avez un vieux fichier `cupsd.conf` que vous voulez porter depuis un autre système ; voir la fin pour plus de détails.

Les champs de `cups-configuration` disponibles sont :

`package cups` [paramètre de `cups-configuration`]  
Le paquet CUPS.

`package-list extensions` (*par défaut :* [paramètre de `cups-configuration`]  
(`list brlaser cups-filters epson-inkjet-printer-escpr  
foomatic-filters hplip-minimal splix`))  
Pilotes et autres extensions du paquet CUPS.

`files-configuration` [paramètre de `cups-configuration`]  
`files-configuration`  
Configuration de l'emplacement où écrire les journaux, quels répertoires utiliser pour les travaux d'impression et les paramètres de configuration privilégiés liés.

Les champs `files-configuration` disponibles sont :

`log-location access-log` [paramètre de `files-configuration`]  
Définit le fichier de journal d'accès. Spécifier un nom de fichier vide désactive la génération de journaux d'accès. La valeur `stderr` fait que les entrées du journal seront envoyés sur l'erreur standard lorsque l'ordonnanceur est lancé au premier plan ou vers le démon de journal système lorsqu'il tourne en tâche de fond. La valeur `syslog` fait que les entrées du journal sont envoyées au démon

de journalisation du système. Le nom du serveur peut être inclus dans les noms de fichiers avec la chaîne `%s`, comme dans `/var/log/cups/%s-access_log`.

La valeur par défaut est `"/var/log/cups/access_log"`.

**file-name cache-dir** [paramètre de files-configuration]

L'emplacement où CUPS devrait mettre les données en cache.

La valeur par défaut est `"/var/cache/cups"`.

**string config-file-perm** [paramètre de files-configuration]

Spécifie les permissions pour tous les fichiers de configuration que l'ordonnanceur écrit.

Remarquez que les permissions pour le fichier `printers.conf` sont actuellement masqués pour ne permettre que l'accès par l'utilisateur de l'ordonnanceur (typiquement `root`). La raison est que les URI des imprimantes contiennent des informations d'authentification sensibles qui ne devraient pas être connues sur le système. Il n'est pas possible de désactiver cette fonctionnalité de sécurité.

La valeur par défaut est `"0640"`.

**log-location error-log** [paramètre de files-configuration]

Définit le fichier de journal d'erreur. Spécifier un nom de fichier vide désactive la génération de journaux d'erreur. La valeur `stderr` fait que les entrées du journal seront envoyés sur l'erreur standard lorsque l'ordonnanceur est lancé au premier plan ou vers le démon de journalisation du système lorsqu'il tourne en tâche de fond. La valeur `syslog` fait que les entrées du journal sont envoyées au démon de journalisation du système. Le nom du serveur peut être inclus dans les noms de fichiers avec la chaîne `%s`, comme dans `/var/log/cups/%s-error_log`.

La valeur par défaut est `"/var/log/cups/error_log"`.

**string fatal-errors** [paramètre de files-configuration]

Spécifie quelles erreurs sont fatales, qui font terminer l'ordonnanceur. Les types de chaînes sont :

**none** Aucune erreur n'est fatale.

**all** Toutes les erreurs ci-dessous sont fatales.

**browse** Les erreurs d'initialisation de la navigation sont fatales, par exemple les connexion échouées au démon DNS-SD.

**config** Les erreurs de syntaxe du fichier de configuration sont fatale.

**listen** Les erreurs d'écoute ou de port sont fatales, sauf pour les erreurs d'IPv6 sur la boucle locale ou les adresses `any`.

**log** Les erreurs de création ou d'écriture des fichiers de journal sont fatales.

**permissions**

Les mauvaises permissions des fichiers de démarrage sont fatales, par exemple un certificat TLS et des fichiers de clefs avec des permissions permettant la lecture à tout le monde.

La valeur par défaut est `"all -browse"`.

- boolean file-device?** [paramètre de files-configuration]  
Spécifie si le fichier de pseudo-périphérique peut être utilisé pour de nouvelles queues d'impression. L'URI `file:///dev/null` est toujours permise.  
La valeur par défaut est `'f'`.
- string group** [paramètre de files-configuration]  
Spécifie le nom ou l'ID du groupe qui sera utilisé lors de l'exécution de programmes externes.  
La valeur par défaut est `"lp"`.
- string log-file-group** [paramètre de files-configuration]  
Spécifie le nom ou l'ID du groupe qui sera utilisé pour les fichiers de journaux.  
La valeur par défaut est `"lpadmin"`.
- string log-file-perm** [paramètre de files-configuration]  
Spécifie les permissions pour tous les fichiers de journal que l'ordonnanceur écrit.  
La valeur par défaut est `"0644"`.
- log-location page-log** [paramètre de files-configuration]  
Définit le fichier de journal de page. Spécifier un nom de fichier vide désactive la génération de journaux de pages. La valeur `stderr` fait que les entrées du journal seront envoyés sur l'erreur standard lorsque l'ordonnanceur est lancé au premier plan ou vers le démon de journal système lorsqu'il tourne en tâche de fond. La valeur `syslog` fait que les entrées du journal sont envoyées au démon de journalisation du système. Le nom du serveur peut être inclus dans les noms de fichiers avec la chaîne `%s`, comme dans `/var/log/cups/%s-page_log`.  
La valeur par défaut est `"/var/log/cups/page_log"`.
- string remote-root** [paramètre de files-configuration]  
Spécifie le nom d'utilisateur associé aux accès non authentifiés par des clients qui se disent être l'utilisateur `root`. La valeur par défaut est `remroot`.  
La valeur par défaut est `"remroot"`.
- file-name request-root** [paramètre de files-configuration]  
Spécifie le répertoire qui contient les travaux d'impression et d'autres données des requêtes HTTP.  
La valeur par défaut est `"/var/spool/cups"`.
- sandboxing sandboxing** [paramètre de files-configuration]  
Spécifie le niveau d'isolation de sécurité appliqué aux filtres d'impression, aux moteurs et aux autres processus fils de l'ordonnanceur ; soit `relaxed` soit `strict`. Cette directive n'est actuellement utilisée et supportée que sur macOS.  
La valeur par défaut est `'strict'`.

- file-name server-keychain** [paramètre de files-configuration]  
Spécifie l'emplacement des certifications TLS et des clés privées. CUPS cherchera les clés publiques et privées dans ce répertoire : un fichier `.crt` pour un certificat encodé en PEM et le fichier `.key` correspondant pour la clé privée encodée en PEM.  
La valeur par défaut est `"/etc/cups/ssl"`.
- file-name server-root** [paramètre de files-configuration]  
Spécifie le répertoire contenant les fichiers de configuration du serveur.  
La valeur par défaut est `"/etc/cups"`.
- boolean sync-on-close?** [paramètre de files-configuration]  
Spécifie si l'ordonnanceur appelle `fsync(2)` après avoir écrit la configuration ou les fichiers d'état.  
La valeur par défaut est `'#f'`.
- space-separated-string-list system-group** [paramètre de files-configuration]  
Spécifie le groupe ou les groupes à utiliser pour l'authentification du groupe `@SYSTEM`.
- file-name temp-dir** [paramètre de files-configuration]  
Spécifie le répertoire où les fichiers temporaires sont stockés.  
La valeur par défaut est `"/var/spool/cups/tmp"`.
- string user** [paramètre de files-configuration]  
Spécifie le nom d'utilisateur ou l'ID utilisé pour lancer des programmes externes.  
La valeur par défaut est `"lp"`.
- string set-env** [paramètre de files-configuration]  
Indique que la variable d'environnement spécifiée doit être passée aux processus fils.  
La valeur par défaut est `"variable value"`.
- access-log-level access-log-level** [paramètre de cups-configuration]  
Spécifie le niveau de journalisation pour le fichier `AccessLog`. Le niveau `config` enregistre les ajouts, suppressions et modifications d'imprimantes et de classes et lorsque les fichiers de configuration sont accédés ou mis à jour. Le niveau `actions` enregistre la soumission, la suspension, la libération, la modification et l'annulation des travaux et toutes les conditions de `config`. Le niveau `all` enregistre toutes les requêtes.  
La valeur par défaut est `'actions'`.
- boolean auto-purge-jobs?** [paramètre de cups-configuration]  
Spécifie s'il faut vider l'historique des travaux automatiquement lorsqu'il n'est plus nécessaire pour les quotas.  
La valeur par défaut est `'#f'`.

- comma-separated-string-list** [paramètre de cups-configuration]  
**browse-dns-sd-sub-types**  
Spécifie une liste de sous-types DNS-SD à annoncer pour chaque imprimante connectée. Par exemple, `" cups " print` dira aux clients réseaux que le partage CUPS et IPP Everywhere sont pris en charge.  
La valeur par défaut est `" cups "`.
- browse-local-protocols** [paramètre de cups-configuration]  
**browse-local-protocols**  
Spécifie les protocoles à utiliser pour partager les imprimantes sur le réseau local.  
La valeur par défaut est `dnssd`.
- boolean browse-web-if?** [paramètre de cups-configuration]  
Spécifie si l'interface web de CUPS est annoncée.  
La valeur par défaut est `#f`.
- boolean browsing?** [paramètre de cups-configuration]  
Spécifie si les imprimantes partagées sont annoncées.  
La valeur par défaut est `#f`.
- string classification** [paramètre de cups-configuration]  
Spécifie la classification de sécurité du serveur. N'importe quel nom de bannière peut être utilisé, comme `"classified"`, `"confidential"`, `"secret"`, `"topsecret"` et `"unclassified"` ou la bannière peut être omise pour désactiver les fonctions d'impression sécurisées.  
La valeur par défaut est `" "`.
- boolean classify-override?** [paramètre de cups-configuration]  
Spécifie si les utilisateurs peuvent remplacer la classification (page de couverture) des travaux d'impression individuels avec l'option `job-sheets`.  
La valeur par défaut est `#f`.
- default-auth-type default-auth-type** [paramètre de cups-configuration]  
Spécifie le type d'authentification par défaut à utiliser.  
La valeur par défaut est `Basic`.
- default-encryption default-encryption** [paramètre de cups-configuration]  
Spécifie si le chiffrement sera utilisé pour les requêtes authentifiées.  
La valeur par défaut est `Required`.
- string default-language** [paramètre de cups-configuration]  
Spécifie la langue par défaut à utiliser pour le contenu textuel et web.  
La valeur par défaut est `"en"`.
- string default-paper-size** [paramètre de cups-configuration]  
Spécifie la taille de papier par défaut pour les nouvelles queues d'impression. `"Auto"` utilise la valeur par défaut du paramètre de régionalisation, tandis que `"None"` spécifie qu'il n'y a pas de taille par défaut. Des noms de tailles spécifique sont par exemple `"Letter"` et `"A4"`.  
La valeur par défaut est `"Auto"`.

- string default-policy** [paramètre de cups-configuration]  
Spécifie la politique d'accès par défaut à utiliser.  
La valeur par défaut est "default".
- boolean default-shared?** [paramètre de cups-configuration]  
Spécifie si les imprimantes locales sont partagées par défaut.  
La valeur par défaut est '#t'.
- entier-non-négatif dirty-clean-interval** [paramètre de cups-configuration]  
Spécifie le délai pour mettre à jour les fichiers de configuration et d'état. Une valeur de 0 fait que la mise à jour arrive aussi vite que possible, typiquement en quelques millisecondes.  
La valeur par défaut est '30'.
- error-policy error-policy** [paramètre de cups-configuration]  
Spécifie ce qu'il faut faire si une erreur a lieu. Les valeurs possibles sont **abort-job**, qui supprimera les travaux d'impression en échec ; **retry-job**, qui tentera de nouveau l'impression plus tard ; **retry-current-job**, qui retentera l'impression immédiatement ; et **stop-printer** qui arrête l'imprimante.  
La valeur par défaut est 'stop-printer'.
- entier-non-négatif filter-limit** [paramètre de cups-configuration]  
Spécifie le coût maximum des filtres qui sont lancés en même temps, pour minimiser les problèmes de ressources de disque, de mémoire et de CPU. Une limite de 0 désactive la limite de filtrage. Une impression standard vers une imprimante non-PostScript requiert une limite de filtre d'environ 200. Une imprimante PostScript requiert environ la moitié (100). Mettre en place la limite en dessous de ces valeurs limitera l'ordonnanceur à un seul travail d'impression à la fois.  
La valeur par défaut est '0'.
- entier-non-négatif filter-nice** [paramètre de cups-configuration]  
Spécifie la priorité des filtres de l'ordonnanceur qui sont lancés pour imprimer un travail. La valeur va de 0, la plus grande priorité, à 19, la plus basse priorité.  
La valeur par défaut est '0'.
- host-name-lookups host-name-lookups** [paramètre de cups-configuration]  
Spécifie s'il faut faire des résolutions inverses sur les clients qui se connectent. Le paramètre **double** fait que **cupsd** vérifie que le nom d'hôte résolu depuis l'adresse correspond à l'une des adresses renvoyées par ce nom d'hôte. Les résolutions doubles évitent aussi que des clients avec des adresses non enregistrées ne s'adressent à votre serveur. N'initialisez cette valeur qu'à **#t** ou **double** que si c'est absolument nécessaire.  
La valeur par défaut est '#f'.
- entier-non-négatif job-kill-delay** [paramètre de cups-configuration]  
Spécifie le nombre de secondes à attendre avant de tuer les filtres et les moteurs associés avec un travail annulé ou suspendu.  
La valeur par défaut est '30'.

**entier-non-négatif job-retry-interval** [paramètre de cups-configuration]  
Spécifie l'intervalle des nouvelles tentatives en secondes. C'est typiquement utilisé pour les queues de fax mais peut aussi être utilisé avec des queues d'impressions normales dont la politique d'erreur est `retry-job` ou `retry-current-job`.

La valeur par défaut est '30'.

**entier-non-négatif job-retry-limit** [paramètre de cups-configuration]  
Spécifie le nombre de nouvelles tentatives pour les travaux. C'est typiquement utilisé pour les queues de fax mais peut aussi être utilisé pour les queues d'impressions dont la politique d'erreur est `retry-job` ou `retry-current-job`.

La valeur par défaut est '5'.

**boolean keep-alive?** [paramètre de cups-configuration]  
Spécifie s'il faut supporter les connexion HTTP keep-alive.

La valeur par défaut est '#t'.

**entier-non-négatif limit-request-body** [paramètre de cups-configuration]  
Spécifie la taille maximale des fichiers à imprimer, des requêtes IPP et des données de formulaires HTML. Une limite de 0 désactive la vérification de la limite.

La valeur par défaut est '0'.

**multiline-string-list listen** [paramètre de cups-configuration]  
Écoute sur les interfaces spécifiées. Les valeurs valides sont de la forme *adresse:port*, où *adresse* est soit une adresse IPv6 dans des crochets, soit une adresse IPv4, soit \* pour indiquer toutes les adresses. Les valeurs peuvent aussi être des noms de fichiers de socket UNIX domain. La directive Listen est similaire à la directive Port mais vous permet de restreindre l'accès à des interfaces ou des réseaux spécifiques.

**entier-non-négatif listen-back-log** [paramètre de cups-configuration]  
Spécifie le nombre de connexions en attente qui seront permises. Ça n'affecte normalement que les serveurs très actifs qui ont atteint la limite MaxClients, mais peut aussi être déclenché par un grand nombre de connexions simultanées. Lorsque la limite est atteinte, le système d'exploitation refusera les connexions supplémentaires jusqu'à ce que l'ordonnanceur accepte les connexions en attente.

La valeur par défaut est '128'.

**location-access-control-list** [paramètre de cups-configuration]  
**location-access-controls**  
Spécifie un ensemble de contrôles d'accès supplémentaires.

Les champs de `location-access-controls` disponibles sont :

**file-name path** [paramètre de location-access-controls]  
Spécifie le chemin d'URI auquel les contrôles d'accès s'appliquent.

**access-control-list** [paramètre de location-access-controls]  
**access-controls**

Les contrôles d'accès pour tous les accès à ce chemin, dans le même format que le champ `access-controls` de `operation-access-control`.

La valeur par défaut est '()'.



**method-access-control-list** [paramètre de `location-access-controls`]  
**method-access-controls**

Contrôles d'accès pour les accès spécifiques à la méthode à ce chemin.

La valeur par défaut est '()'.  
 Les champs de `method-access-controls` disponibles sont :

**boolean reverse?** [paramètre de `method-access-controls`]

Si la valeur est `#t`, applique les contrôles d'accès à toutes les méthodes sauf les méthodes listées. Sinon, applique le contrôle uniquement aux méthodes listées.

La valeur par défaut est `#f`.

**method-list methods** [paramètre de `method-access-controls`]

Les méthodes auxquelles ce contrôle d'accès s'applique.

La valeur par défaut est '()'.

**access-control-list** [paramètre de `method-access-controls`]

**access-controls**

Directives de contrôle d'accès, en tant que liste de chaînes de caractères. Chaque chaîne devrait être une directive, comme `"Order allow,deny"`.

La valeur par défaut est '()'.

**entier-non-négatif log-debug-history** [paramètre de `cups-configuration`]

Spécifie le nombre de messages de débogage qui sont retenus pour la journalisation si une erreur arrive dans un travail d'impression. Les messages de débogage sont journalisés indépendamment du paramètre `LogLevel`.

La valeur par défaut est '100'.

**log-level log-level** [paramètre de `cups-configuration`]

Spécifie le niveau de journalisation du fichier `ErrorLog`. La valeur `none` arrête toute journalisation alors que `debug2` enregistre tout.

La valeur par défaut est 'info'.

**log-time-format log-time-format** [paramètre de `cups-configuration`]

Spécifie le format de la date et de l'heure dans les fichiers de journaux. La valeur `standard` enregistre les secondes entières alors que `usecs` enregistre les microsecondes.

La valeur par défaut est 'standard'.

**entier-non-négatif max-clients** [paramètre de `cups-configuration`]

Spécifie le nombre maximum de clients simultanés qui sont autorisés par l'ordonnanceur.

La valeur par défaut est '100'.

**entier-non-négatif** [paramètre de `cups-configuration`]

**max-clients-per-host**

Spécifie le nombre maximum de clients simultanés permis depuis une même adresse.

La valeur par défaut est '100'.

**entier-non-négatif max-copies** [paramètre de cups-configuration]  
Spécifie le nombre maximum de copies qu'un utilisateur peut imprimer pour chaque travail.

La valeur par défaut est '9999'.

**entier-non-négatif max-hold-time** [paramètre de cups-configuration]  
Spécifie la durée maximum qu'un travail peut rester dans l'état de suspension *indefinite* avant qu'il ne soit annulé. La valeur 0 désactive l'annulation des travaux suspendus.

La valeur par défaut est '0'.

**entier-non-négatif max-jobs** [paramètre de cups-configuration]  
Spécifie le nombre maximum de travaux simultanés autorisés. La valeur 0 permet un nombre illimité de travaux.

La valeur par défaut est '500'.

**entier-non-négatif max-jobs-per-printer** [paramètre de cups-configuration]  
Spécifie le nombre maximum de travaux simultanés autorisés par imprimante. La valeur 0 permet au plus MaxJobs travaux par imprimante.

La valeur par défaut est '0'.

**entier-non-négatif max-jobs-per-user** [paramètre de cups-configuration]  
Spécifie le nombre maximum de travaux simultanés permis par utilisateur. La valeur 0 permet au plus MaxJobs travaux par utilisateur.

La valeur par défaut est '0'.

**entier-non-négatif max-job-time** [paramètre de cups-configuration]  
Spécifie la durée maximum qu'un travail peut prendre avant qu'il ne soit annulé, en secondes. Indiquez 0 pour désactiver l'annulation des travaux « coincés ».

La valeur par défaut est '10800'.

**entier-non-négatif max-log-size** [paramètre de cups-configuration]  
Spécifie la taille maximale des fichiers de journaux avant qu'on ne les fasse tourner, en octets. La valeur 0 désactive la rotation.

La valeur par défaut est '1048576'.

**entier-non-négatif multiple-operation-timeout** [paramètre de cups-configuration]  
Spécifie la durée maximale à permettre entre les fichiers d'un travail en contenant plusieurs, en secondes.

La valeur par défaut est '900'.

**string page-log-format** [paramètre de cups-configuration]  
Spécifie le format des lignes PageLog. Les séquences qui commencent par un pourcent ('%') sont remplacées par l'information correspondante, tandis que les autres caractères sont copiés littéralement. Les séquences pourcent suivantes sont reconnues :

'%%' insère un seul caractères pourcent

‘%{name}’ insère la valeur de l’attribut IPP spécifié  
 ‘%C’ insère le nombre de copies pour la page actuelle  
 ‘%P’ insère le numéro de page actuelle  
 ‘%T’ insère la date et l’heure actuelle dans un format de journal commun  
 ‘%j’ insère l’ID du travail  
 ‘%p’ insère le nom de l’imprimante  
 ‘%u’ insère le nom d’utilisateur

Si la valeur est la chaîne vide, le PageLog est désactivée. La chaîne %p %u %j %T %P %C %{job-billing} %{job-originating-host-name} %{job-name} %{media} %{sides} crée un PageLog avec les entrées standards.

La valeur par défaut est ‘’.

**environment-variables** [paramètre de cups-configuration]

**environment-variables**

Passes les variables d’environnement spécifiées aux processus fils ; une liste de chaînes de caractères.

La valeur par défaut est ‘()’.

**policy-configuration-list policies** [paramètre de cups-configuration]

Spécifie des politiques de contrôle d’accès nommées.

Les champs de **policy-configuration** disponibles sont :

**string name** [paramètre de policy-configuration]

Nom de la politique.

**string job-private-access** [paramètre de policy-configuration]

Spécifie une liste d’accès pour les valeurs privées du travail. @ACL correspond aux valeurs `requesting-user-name-allowed` ou `requesting-user-name-denied` de l’imprimante. @OWNER correspond au propriétaire du travail. @SYSTEM correspond aux groupes listés dans le champ `system-group` de la configuration `files-configuration`, qui est réifié dans le fichier `cups-files.conf(5)`. Les autres éléments possibles de la liste d’accès sont des noms d’utilisateurs spécifiques et @group pour indiquer les membres d’un groupe spécifique. La liste d’accès peut aussi être simplement `all` ou `default`.

La valeur par défaut est ‘"@OWNER @SYSTEM"’.

**string job-private-values** [paramètre de policy-configuration]

Spécifie la liste des valeurs de travaux à rendre privée, ou `all`, `default`, ou `none`.

La valeur par défaut est ‘"job-name job-originating-host-name job-originating-user-name phone"’.

**string** [paramètre de `policy-configuration`]  
**subscription-private-access**

Spécifie une liste d'accès pour les valeurs privées de la souscription. `@ACL` correspond aux valeurs `requesting-user-name-allowed` ou `requesting-user-name-denied` de l'imprimante. `@OWNER` correspond au propriétaire du travail. `@SYSTEM` correspond aux groupes listés dans le champ `system-group` de la configuration `files-configuration`, qui est réifié dans le fichier `cups-files.conf(5)`. Les autres éléments possibles de la liste d'accès sont des noms d'utilisateurs spécifiques et `@group` pour indiquer les membres d'un groupe spécifique. La liste d'accès peut aussi être simplement `all` ou `default`.

La valeur par défaut est `"@OWNER @SYSTEM"`.

**string** [paramètre de `policy-configuration`]  
**subscription-private-values**

Spécifie la liste des valeurs de travaux à rendre privée, ou `all`, `default`, ou `none`.

La valeur par défaut est `"notify-events notify-pull-method notify-recipient-uri notify-subscriber-user-name notify-user-data"`.

**operation-access-control-list** [paramètre de `policy-configuration`]  
**access-controls**

Contrôle d'accès par les actions IPP.

La valeur par défaut est `('')`.

**booléen-ou-entier-non-négatif** [paramètre de `cups-configuration`]  
**preserve-job-files**

Spécifie si les fichiers de travaux (les documents) sont préservés après qu'un travail est imprimé. Si une valeur numérique est spécifiée, les fichiers de travaux sont préservés pour le nombre de secondes indiquées après l'impression. Sinon, une valeur booléenne s'applique indéfiniment.

La valeur par défaut est `'86400'`.

**booléen-ou-entier-non-négatif** [paramètre de `cups-configuration`]  
**preserve-job-history**

Spécifie si l'historique des travaux est préservé après qu'un travail est imprimé. Si une valeur numérique est spécifiée, l'historique des travaux est préservé pour le nombre de secondes indiquées après l'impression. Si la valeur est `#t`, l'historique des travaux est préservé jusqu'à atteindre la limite `MaxJobs`.

La valeur par défaut est `'#t'`.

**entier-non-négatif** `reload-timeout` [paramètre de `cups-configuration`]

Spécifie la durée d'attente pour la fin des travaux avant de redémarrer l'ordonnanceur.

La valeur par défaut est `'30'`.

**string** `rip-cache` [paramètre de `cups-configuration`]

Spécifie la quantité de mémoire maximale à utiliser pour convertir des documents en bitmaps pour l'imprimante.

La valeur par défaut est `"128m"`.

- string server-admin** [paramètre de cups-configuration]  
Spécifie l'adresse de courriel de l'administrateur système.  
La valeur par défaut est `"root@localhost.localdomain"`.
- host-name-list-or-\* server-alias** [paramètre de cups-configuration]  
La directive `ServerAlias` est utilisée pour la validation des en-tête HTTP Host lorsque les clients se connectent à l'ordonnanceur depuis des interfaces externes. Utiliser le nom spécial `*` peut exposer votre système à des attaques connues de recombinaison DNS dans le navigateur, même lorsque vous accédez au site à travers un pare-feu. Si la découverte automatique des autres noms ne fonctionne pas, nous vous recommandons de lister chaque nom alternatif avec une directive `SeverAlias` plutôt que d'utiliser `*`.  
La valeur par défaut est `'*'`.
- string server-name** [paramètre de cups-configuration]  
Spécifie le nom d'hôte pleinement qualifié du serveur.  
La valeur par défaut est `"localhost"`.
- server-tokens server-tokens** [paramètre de cups-configuration]  
Spécifie les informations incluses dans les en-têtes Server des réponses HTTP. `None` désactive l'en-tête Server. `ProductOnly` rapporte CUPS. `Major` rapporte CUPS 2. `Minor` rapporte CUPS 2.0. `Minimal` rapporte CUPS 2.0.0. `OS` rapporte CUPS 2.0.0 (`uname`) où `uname` est la sortie de la commande `uname`. `Full` rapporte CUPS 2.0.0 (`uname`) IPP/2.0.  
La valeur par défaut est `'Minimal'`.
- multiline-string-list ssl-listen** [paramètre de cups-configuration]  
Écoute des connexions chiffrées sur les interfaces spécifiées. Les valeurs valides sont de la forme `adresse:port`, où `adresse` est soit une adresse IPv6 dans des crochets, soit une adresse IPv4, soit `*` pour indiquer toutes les interfaces.  
La valeur par défaut est `'()''`.
- ssl-options ssl-options** [paramètre de cups-configuration]  
Indique les options de chiffrement. Par défaut, CUPS ne supporte que le chiffrement avec TLS v1.0 ou plus avec des suites de chiffrement connues pour être sûres. La sécurité peut être réduite en utilisant les options `Allow`, et améliorée en utilisant les options `Deny`. L'option `AllowRC4` active les suites de chiffrement 128-bits RC4, qui sont requises pour certains vieux. L'option `AllowSSL3` active SSL v3.0, qui est requis par certains vieux clients qui ne supportent pas TLS v1.0. L'option `DenyCBC` désactive toutes les suites CBC. L'option `DenyTLS1.0` désactive la prise en charge de TLS v1.0 — la version minimale du protocole est donc TLS v1.1.  
La valeur par défaut est `'()''`.
- boolean strict-conformance?** [paramètre de cups-configuration]  
Spécifie si l'ordonnanceur demande aux clients d'adhérer aux spécifications IPP.  
La valeur par défaut est `'#f'`.
- entier-non-négatif timeout** [paramètre de cups-configuration]  
Spécifie le délai d'attente des requêtes HTTP, en secondes.  
La valeur par défaut est `'900'`.

**boolean web-interface?** [paramètre de `cups-configuration`]  
 Spécifie si l'interface web est activée.  
 La valeur par défaut est `'#f'`.

Maintenant, vous vous dites peut-être « oh là là, cher manuel de Guix, je t'aime bien mais arrête maintenant avec ces options de configuration »<sup>6</sup>. En effet, cependant, encore un point supplémentaire : vous pouvez avoir un fichier `cupsd.conf` existant que vous pourriez vouloir utiliser. Dans ce cas, vous pouvez passer un `opaque-cups-configuration` en configuration d'un `cups-service-type`.

Les champs de `opaque-cups-configuration` disponibles sont :

**package cups** [paramètre de `opaque-cups-configuration`]  
 Le paquet CUPS.

**string cupsd.conf** [paramètre de `opaque-cups-configuration`]  
 Le contenu de `cupsd.conf`, en tant que chaîne de caractères.

**string cups-files.conf** [paramètre de `opaque-cups-configuration`]  
 Le contenu du fichier `cups-files.conf`, en tant que chaîne de caractères.

Par exemple, si vos fichiers `cupsd.conf` et `cups-files.conf` sont dans des chaînes du même nom, pouvez instancier un service CUPS de cette manière :

```
(service cups-service-type
  (opaque-cups-configuration
    (cupsd.conf cupsd.conf)
    (cups-files.conf cups-files.conf)))
```

### 12.9.9 Services de bureaux

Le module (`gnu services desktop`) fournit des services qui sont habituellement utiles dans le contexte d'une installation « de bureau » — c'est-à-dire sur une machine qui fait tourner un service d'affichage graphique, éventuellement avec des interfaces utilisateurs graphiques, etc. Il définit aussi des services qui fournissent des environnements de bureau spécifiques comme GNOME, Xfce et MATE.

Pour simplifier les choses, le module définit une variable contenant l'ensemble des services que les utilisateurs s'attendent en général à avoir sur une machine avec un environnement graphique et le réseau :

**%desktop-services** [Variable Scheme]

C'est la liste des services qui étend `%base-services` en ajoutant ou en ajustant des services pour une configuration « de bureau » typique.

En particulier, elle ajoute un gestionnaire de connexion graphique (voir Section 12.9.7 [Système de fenêtrage X], page 325), des verrouilleurs d'écran, un outil de gestion réseau (voir Section 12.9.5 [Services réseau], page 299) avec la prise en charge des modems (voir Section 12.9.5 [Services réseau], page 299), des services de gestion de l'énergie et des couleurs, le gestionnaire de connexion et de siège `elogind`, le service de

<sup>6</sup> NdT : je vous rassure, c'est aussi mon sentiment au moment de traduire ces lignes. Et pour moi, c'est encore loin d'être fini.

privilèges Polkit, le service de géolocalisation GeoClue, le démon AccountsService qui permet aux utilisateurs autorisés de changer les mots de passe du système, un client NTP (voir Section 12.9.5 [Services réseau], page 299), le démon Avahi, et le service name service switch est configuré pour pouvoir utiliser `nss-mdns` (voir Section 12.12 [Name Service Switch], page 573).

La variable `%desktop-services` peut être utilisée comme champ `services` d'une déclaration `operating-system` (voir Section 12.2 [référence de operating-system], page 246).

En plus, les procédures `gnome-desktop-service-type`, `xfce-desktop-service`, `mate-desktop-service-type`, `lxqt-desktop-service-type` et `enlightenment-desktop-service-type` peuvent ajouter GNOME, Xfce, MATE ou Enlightenment à un système. « Ajouter GNOME » signifie que les services du système comme les utilitaires d'ajustement de la luminosité et de gestion de l'énergie sont ajoutés au système, en étendant `polkit` et `dbus` de la bonne manière, ce qui permet à GNOME d'opérer avec des privilèges plus élevés sur un nombre limité d'interfaces systèmes spécialisées. En plus, ajouter un service construit par `gnome-desktop-service-type` ajoute le métapaquet GNOME au profil du système. De même, ajouter le service Xfce ajoute non seulement le métapaquet `xfce` au profil système, mais il permet aussi au gestionnaire de fichiers Thunar d'ouvrir une fenêtre de gestion des fichier « en mode root », si l'utilisateur s'authentifie avec le mot de passe administrateur via l'interface graphique `polkit` standard. « Ajouter MATE » signifie que `polkit` et `dbus` sont étendue de la bonne manière, ce qui permet à MATE d'opérer avec des privilèges plus élevés sur un nombre limité d'interface systèmes spécialisées. En plus, ajouter un service de type `mate-desktop-service-type` ajoute le métapaquet MATE au profil du système. « Ajouter Enlightenment » signifie que `dbus` est étendu comme il faut et que plusieurs binaires d'Enlightenment récupèrent le bit `setuid`, ce qui permet au verrouilleur d'écran d'Enlightenment et à d'autres fonctionnalités de fonctionner correctement.

Les environnement de bureau dans Guix utilisent le service d'affichage Xorg par défaut. Si vous voulez utiliser le protocole de serveur d'affichage plus récent Wayland, vous devez activer la prise en charge de Wayland dans GDM (voir [wayland-gdm], page 325). Une autre solution consiste à utiliser `sddm-service` à la place de GDM comme gestionnaire de connexion graphique. Vous devriez ensuite sélectionner la session « GNOME (Wayland) » dans SDDM. Autrement, vous pouvez essayer de démarrer GNOME sur Wayland manuellement depuis un TTY avec la commande `XDG_SESSION_TYPE=wayland exec dbus-run-session gnome-session`. Actuellement seul GNOME support Wayland.

`gnome-desktop-service-type` [Variable Scheme]

C'est le type de service qui ajoute l'environnement de bureau GNOME (<https://www.gnome.org>). Sa valeur est un objet `gnome-desktop-configuration` (voir plus bas).

Ce service ajoute le paquet `gnome` au profil du système et étend `polkit` avec les actions de `gnome-settings-daemon`.

`gnome-desktop-configuration` [Type de données]

Enregistrement de la configuration de l'environnement de bureau GNOME.

`gnome` (par défaut : `gnome`)  
Le paquet GNOME à utiliser.

`xfce-desktop-service-type` [Variable Scheme]  
C'est le type de service qui lance l'environnement de bureau <https://xfce.org/> (`Xfce`).  
Sa valeur est un objet `xfce-desktop-configuration` (voir plus bas).

Ce service ajoute le paquet `xfce` au profil du système et étend polkit avec la possibilité pour `thunar` de manipuler le système de fichier en root depuis une session utilisateur, après que l'utilisateur s'authentifie avec le mot de passe administrateur.

Remarquez que `xfce4-panel` et ses paquets de greffons devraient être installés dans le même profil pour assurer la compatibilité. Lorsque vous utilisez ce service, vous devriez ajouter les greffons supplémentaires (`xfce4-whiskermenu-plugin`, `xfce4-weather-plugin`, etc) au champ `packages` de votre `operating-system`.

`xfce-desktop-configuration` [Type de données]  
Enregistrement de la configuration de l'environnement de bureau Xfce.

`xfce` (par défaut : `xfce`)  
Le paquet Xfce à utiliser.

`mate-desktop-service-type` [Variable Scheme]  
C'est le type de service qui lance l'environnement de bureau MATE (<https://mate-desktop.org/>). Sa valeur est un objet `mate-desktop-configuration` (voir plus bas).

Ce service ajoute le paquet `mate` au profil du système, et étend polkit avec les actions de `mate-settings-daemon`.

`mate-desktop-configuration` [Type de données]  
Enregistrement de configuration pour l'environnement de bureau MATE.

`mate` (par défaut : `mate`)  
Le paquet MATE à utiliser.

`lxqt-desktop-service-type` [Variable Scheme]  
C'est le type de service qui lance l'environnement de bureau LXQt ([https://lxqt-project.org](https://lxqt-project.org/)). Sa valeur est un objet `lxqt-desktop-configuration` (voir plus bas).

Ce service ajoute le paquet `lxqt` au profil du système.

`lxqt-desktop-configuration` [Type de données]  
Enregistrement de la configuration de l'environnement de bureau LXQt.

`lxqt` (par défaut : `lxqt`)  
Le paquet LXQT à utiliser.

`enlightenment-desktop-service-type` [Variable Scheme]  
Renvoie un service qui ajoute le paquet `enlightenment` et étend dbus avec les actions de `efl`.



**enlightenment-desktop-service-configuration** [Type de données]

**enlightenment** (par défaut : **enlightenment**)

Le paquet enlightenment à utiliser.

Comme les services de bureau GNOME, Xfce et MATE récupèrent tant de paquet, la variable `%desktop-services` par défaut n'inclut aucun d'entre eux. Pour ajouter GNOME, Xfce ou MATE, utilisez `cons` pour les ajouter à `%desktop-services` dans le champ `services` de votre `operating-system` :

```
(use-modules (gnu))
(use-service-modules desktop)
(operating-system
  ...
  ;; cons* ajoute des éléments à la liste donnée en dernier argument.
  (services (cons* (service gnome-desktop-service-type)
                  (service xfce-desktop-service)
                  %desktop-services))
  ...)
```

Ces environnements de bureau seront alors disponibles comme une option dans la fenêtre de connexion graphique.

Les définitions de service qui sont vraiment incluses dans `%desktop-services` et fournies par `(gnu services dbus)` et `(gnu services desktop)` sont décrites plus bas.

**dbus-service** [`#:dbus dbus`] [`#:services '()`] [`#:verbose?`] [Procédure Scheme]

Renvoie un service qui lance le « bus système », avec `dbus`, et la prise en charge des `services`. Lorsque `verbose?` est vrai, cela initialise la variable d'environnement `'DBUS_VERBOSE'` à `'1'` ; un paquet D-Bus verbeux comme `dbus-verbose` devrait être fournit en tant que `dbus` dans ce cas. La sortie verbeuse est enregistrée dans `/var/log/dbus-daemon.log`.

D-Bus (<https://dbus.freedesktop.org/>) est un utilitaire de communication inter-processus. Son bus système est utilisé pour permettre à des services systèmes de communiquer et d'être notifiés d'événements systèmes.

`services` doit être une liste de paquets qui fournissent un répertoire `etc/dbus-1/system.d` contenant de la configuration D-Bus supplémentaire et des fichiers de politiques. Par exemple, pour permettre à `avahi-daemon` d'utiliser le bus système, `services` doit être égal à `(list avahi)`.

**elogind-service** [`#:config config`] [Procédure Scheme]

Renvoie un service qui lance le démon de gestion de connexion et de siège `elogind`. Elogind (<https://github.com/elogind/elogind>) expose une interface D-Bus qui peut être utilisée pour connaître quels utilisateurs sont connectés, le type de session qu'ils sont ouverte, suspendre le système, désactiver la veille système, redémarrer le système et d'autre tâches.

Elogind gère la plupart des événements liés à l'énergie du système, par exemple mettre en veille le système quand l'écran est rabattu ou en l'éteignant quand le bouton de démarrage est appuyé.

L'argument *config* spécifie la configuration d'elogind et devrait être le résultat d'une invocation de (`elogind-configuration (parameter value)...`). Les paramètres disponibles et leur valeur par défaut sont :

```
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` [Procédure Scheme]  
`accountsservice`]

Renvoie un service qui lance AccountsService, un service système qui peut lister les comptes disponibles, changer leur mot de passe, etc. AccountsService s'intègre à Polkit pour permettre aux utilisateurs non privilégiés de pouvoir modifier la configuration de leur système. le site de accountsservice (<https://www.freedesktop.org/wiki/Software/AccountsService/>) pour trouver plus d'informations.

L'argument `accountsservice` est le paquet `accountsservice` à exposer comme un service.

`polkit-service` [`#:polkit polkit`] [Procédure Scheme]

Renvoie un service qui lance le service de gestion des privilèges Polkit (<https://www.freedesktop.org/wiki/Software/polkit/>), qui permet aux administrateurs systèmes de permettre l'accès à des opération privilégiées d'une manière structurée. En demandant au service Polkit, un composant système privilégié peut savoir lorsqu'il peut donner des privilèges supplémentaires à des utilisateurs normaux. Par exemple, un utilisateur normal peut obtenir le droit de mettre le système en veille si l'utilisateur est connecté localement.

`polkit-wheel-service` [Variable Scheme]

Service qui ajoute le groupe `wheel` comme administrateur au service Polkit. Cela fait en sorte qu'on demande le mot de passe des utilisateurs du groupe `wheel` eux-mêmes pour effectuer des tâches d'administration au lieu du mot de passe `root`, comme le fait `sudo`.

`upower-service-type` [Variable Scheme]

Service qui lance `upowerd` (<https://upower.freedesktop.org/>), un moniteur système de consommation d'énergie et de niveau de batterie, avec les paramètres de configuration donnés.

Il implémente l'interface D-Bus `org.freedesktop.UPower` et est notamment utilisé par GNOME.

**upower-configuration** [Type de données]

Type de données représentant la configuration de UPower.

**upower** (par défaut : *upower*)

Paquet à utiliser pour **upower**.

**watts-up-pro?** (par défaut : **#f**)

Active le périphérique Watts Up Pro.

**poll-batteries?** (par défaut : **#t**)

Active les requêtes au noyau pour les changements de niveau de batterie.

**ignore-lid?** (par défaut : **#f**)

Ignore l'état de l'écran, ce qui peut être utile s'il est incorrect sur un appareil.

**use-percentage-for-policy?** (par défaut : **#t**)

Indique s'il faut utiliser une politique basée sur le pourcentage de batterie plutôt que l'estimation du temps restant. Une politique basée sur le pourcentage de batterie est souvent plus fiable.

**percentage-low** (par défaut : 20)

Lorsque **use-percentage-for-policy?** est **#t**, cela indique à quel niveau la batterie est considérée comme faible.

**percentage-critical** (par défaut : 5)

Lorsque **use-percentage-for-policy?** est **#t**, cela indique à quel niveau la batterie est considérée comme critique.

**percentage-action** (par défaut : 2)

Lorsque **use-percentage-for-policy?** est **#t**, cela indique à quel niveau l'action sera prise.

**time-low** (par défaut : 1200)

Lorsque **use-percentage-for-policy?** est **#f**, cela indique à quelle durée restante en secondes la batterie est considérée comme faible.

**time-critical** (par défaut : 300)

Lorsque **use-percentage-for-policy?** est **#f**, cela indique à quelle durée restante en secondes la batterie est considérée comme critique.

**time-action** (par défaut : 120)

Lorsque **use-percentage-for-policy?** est **#f**, cela indique à quelle durée restante en secondes l'action sera prise.

**critical-power-action** (par défaut : `'hybrid-sleep'`)

L'action à prendre lorsque **percentage-action** ou **time-action** est atteint (en fonction de la configuration de **use-percentage-for-policy?**).

Les valeurs possibles sont :

- `'power-off'`

- 'hibernate
- 'hybrid-sleep.

**udisks-service** [#:udisks udisks] [Procédure Scheme]

Renvoie un service pour UDisks (<https://udisks.freedesktop.org/docs/latest/>), un démon de *gestion de disques* qui fournit des notifications et la capacité de monter et démonter des disques à des interfaces utilisateurs. Les programmes qui parlent à UDisks sont par exemple la commande `udisksctl`, qui fait partie de UDisks, et GNOME Disks. Remarquez que Udisks s'appuie sur la commande `mount`, donc il ne sera capable d'utiliser que les utilitaires de systèmes de fichiers installés dans le profil du système. Par exemple si vous voulez monter des systèmes de fichiers NTFS en lecture-écriture, vous devrez ajouter `ntfs-3g` aux paquets installés sur le système entier.

**colord-service-type** [Variable Scheme]

C'est le type de service qui lance `colord`, un service système avec une interface D-Bus pour gérer les profils de couleur des périphériques d'entrées et de sorties comme les écrans et les scanners. Il est notamment utilisé par l'outil graphique GNOME Color Manager. Voir le site web de `colord` (<https://www.freedesktop.org/software/colord/>) pour plus d'informations.

**sane-service-type** [Variable Scheme]

Ce service fournit un accès aux scanner via SANE (<http://www.sane-project.org>) en installant les règles udev nécessaires. Il est inclus dans `%desktop-services` (voir Section 12.9.9 [Services de bureaux], page 347) et s'appuie sur le paquet `sane-backends-minimal` par défaut (voir plus bas) pour la prise en charge matérielle.

**sane-backends-minimal** [Variable Scheme]

Le paquet par défaut installé par le `sane-service-type`. Il prend en charge beaucoup de scanners récents.

**sane-backends** [Variable Scheme]

Ce paquet inclus la prise en charge de tous les scanners pris en charge par `sane-backends-minimal`, plus les scanners Hewlett-Packard plus anciens pris en charges par le paquet `hplip`. Pour l'utiliser sur un système qui utiliser les `%desktop-services`, vous pouvez utiliser `modify-services` (voir Section 12.18.3 [Référence de service], page 602) comme montré ci-dessous :

```
(use-modules (gnu))
(use-service-modules
  ...
  desktop)
(use-package-modules
  ...
  scanner)
```

```
(define %my-desktop-services
;; Liste des services de bureau qui prennent en charge un plus grand nombre de
(modify-services %desktop-services
```

```
(sane-service-type _ => sane-backends)))
```

```
(operating-system
;; ...
(services %my-desktop-services))
```

**geoclue-application** *name* [#:*allowed?* #*t*] [#:*system?* #*f*] [#:*users* '()] [Procédure Scheme]

Renvoie une configuration qui permet d'accéder aux données de localisation de GeoClue. *name* est l'ID Desktop de l'application, sans la partie en *.desktop*. Si *allowed?* est vraie, l'application aura droit d'accéder aux informations de localisation par défaut. Le booléen *system?* indique si une application est un composant système ou non. Enfin *users* est la liste des UID des utilisateurs pour lesquels cette application a le droit d'accéder aux informations de géolocalisation. Une liste d'utilisateurs vide indique que tous les utilisateurs sont autorisés.

**%standard-geoclue-applications** [Variable Scheme]

La liste standard de configuration des application GeoClue connues, qui permet à l'utilitaire *date-and-time* de GNOME de demander l'emplacement actuel pour initialiser le fuseau horaire et aux navigateurs web IceCat et Epiphany de demander les informations de localisation. IceCat et Epiphany demandent tous deux à l'utilisateur avant de permettre à une page web de connaître l'emplacement de l'utilisateur.

**geoclue-service** [#:*colord* *colord*] [#:*whitelist* '()] [Procédure Scheme]  
[#:*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]

Renvoie un service qui lance le service de géolocalisation GeoClue. Ce service fournit une interface D-Bus pour permettre aux applications de demande l'accès à la position de l'utilisateur et éventuellement d'ajouter des informations à des bases de données de géolocalisation en ligne. Voir le site web de GeoClue (<https://wiki.freedesktop.org/www/Software/GeoClue/>) pour plus d'informations.

**bluetooth-service** [#:*bluez* *bluez*] [#:*auto-enable?* #*f*] [Procédure Scheme]

Renvoie un service qui lance le démon *bluetoothd* qui gère tous les appareils Bluetooth et fournit un certain nombre d'interfaces D-Bus. Lorsque *auto-enable?* est vraie, le contrôleur bluetooth est automatiquement alimenté au démarrage, ce qui peut être utile lorsque vous utilisez un clavier ou une souris bluetooth.

Les utilisateurs doivent être dans le groupe *lp* pour accéder au service D-Bus.

**bluetooth-service-type** [Variable Scheme]

C'est le type pour le système Linux Bluetooth Protocol Stack (<https://bluez.org/>) (BlueZ), qui génère le fichier de configuration */etc/bluetooth/main.conf*. La valeur de ce type est un enregistrement *bluetooth-configuration* comme dans cet exemple :

```
(service bluetooth-service-type)
```

Voir plus bas pour des détails sur *bluetooth-configuration*.

- bluetooth-configuration** [Type de données]  
Type de données représentant la configuration pour **bluetooth-service**.
- bluez** (par défaut : **bluez**)  
Le paquet **bluez** à utiliser.
- name** (par défaut : "BlueZ")  
Nom de l'adaptateur par défaut.
- class** (par défaut : **#x000000**)  
Classe de périphérique par défaut. Seuls les bits de classe de périphérique majeurs et mineurs sont pris en compte.
- discoverable-timeout** (par défaut : **180**)  
Indique le temps pendant lequel rester en mode découvrable avant de repasser en mode non-découvrable. La valeur est en secondes.
- always-pairable?** (par défaut : **#f**)  
Permet de toujours s'appairer même s'il n'y a aucun agent enregistré.
- pairable-timeout** (par défaut : **0**)  
Indique combien de temps rester en mode d'appairage avant de repasser en mode non-découvrable. La valeur est en secondes.
- device-id** (par défaut : **#f**)  
Utilise les informations de source de l'identifiant de fabricant (assignateur), de fabricant, de produit et de version pour la prise en charge du profil DID. Les valeurs sont séparées par « : » et *assignateur*, *fabricant*, *produit* et *version*.  
Les valeurs possibles sont :
- **#f** pour le désactiver,
  - "assignateur:1234:5678:abcd", où *assignateur* est soit **usb** (par défaut) soit **bluetooth**.
- reverse-service-discovery?** (par défaut : **#t**)  
Utilise la découverte de service inversée pour les périphériques précédemment inconnus qui se connectent. Pour BR/EDR cette option n'est vraiment requise que pour la qualification cate le testeur BITE n'aime pas la découverte inversée dans certains cas de tests, pour LE cela désactive la fonctionnalité cliente GATT donc elle peut être utilisée pour les systèmes qui n'opèrent que comme un périphérique.
- name-resolving?** (par défaut : **#t**)  
Active la résolution de nom après la requête. Indiquez **#f** si vous n'avez pas besoin des noms des périphériques distants et souhaitez avoir un cycle de découverte plus court.
- debug-keys?** (par défaut : **#f**)  
Active la persistance des clés de lien de débogage à l'exécution. La valeur par défaut est fausse et rend les clés de lien de débogage valides uniquement pour la durée de la connexion pour laquelle elles ont été créées.

`controller-mode` (par défaut : `'dual`)

Restreint tous les contrôleurs à leur transport spécifié. `'dual` signifie que BR/EDR et LE sont tous deux activés (si cela est pris en charge par le matériel).

Les valeurs possibles sont :

- `'dual`
- `'breedr`
- `'le`

`multi-profile` (par défaut : `'off`)

Active la prise en charge de la spécification multi-profil. Cela permet de spécifier si le système ne prend en charge que la configuration multi-profil mono-appareil (MPSD) ou à la fois la configuration multi-profil mono-appareil (MPSD) et multi-profil multi-appareil (MPMD).

Les valeurs possibles sont :

- `'off`
- `'single`
- `'multiple`

`fast-connectable?` (par défaut : `#f`)

Active le paramètre Fast Connectable de manière permanente pour les adaptateurs qui le prennent en charge. Lorsqu'il est activé les autres appareils peuvent se connecter plus rapidement, cependant le compromis est l'augmentation de la consommation électrique. Ce paramètre ne fonctionnera correctement que sur les versions du noyau 4.1 et supérieur.

`privacy` (par défaut : `'off`)

Paramètres de vie privée par défaut.

- `'off` : désactive la vie privée locale
- `'network/on` : un appareil n'acceptera pas de paquets d'annonce d'appareils appairés qui contiennent des adresses privées. Cela peut ne pas être compatible avec certains anciens appareils car ils nécessitent l'utilisation continue de RPA
- `'device` : un appareil dans le mode de vie privée pour l'appareil ne se soucie que de la vie privée de l'appareil et acceptera les paquets d'annonces d'appareils appairés qui contiennent leur adresse d'identité ainsi que ceux qui contiennent une adresse privée, même si l'appareil a distribué son IRK par le passé

et en plus, si `controller-mode` vaut `'dual` :

- `'limited-network` : applique le mode de découverte limité aux annonces, qui suivent la même politique que BR/EDR qui publie l'adresse d'identité en mode découvrable, et le mode de vie privée réseau pour scanner
- `'limited-device` : applique le mode de découverte limitée aux annonces, qui suit la même politique que BR/EDR qui publie d'adresse



d'identité en mode découvrable, et le mode de vie privé de l'appareil pour le scan.

`just-works-repairing` (par défaut : `'never'`)

Spécifie la politique de ré-appairage JUST-WORKS du pair.

Les valeurs possibles sont :

- `'never'`
- `'confirm'`
- `'always'`

`temporary-timeout` (par défaut : `30`)

Le temps pendant lequel garder les appareils temporaires. La valeur est en secondes. `0` désactive complètement l'horloge.

`refresh-discovery?` (par défaut : `#t`)

Permet au périphérique d'envoyer une requête SDP pour mettre à jour les services connus quand le profil est connecté.

`experimental?` (par défaut : `#f`)

Active les fonctionnalités et les interfaces expérimentales, ou bien une liste d'UUID.

Les valeurs possibles sont :

- `#t`
- `#f`
- `(list (uuid <uuid-1>) (uuid <uuid-2>) ...)`.

La liste des UUID possibles :

- `d4992530-b9ec-469f-ab01-6c481c47da1c` : le débogage expérimental de BlueZ,
- `671b10b5-42c0-4696-9227-eb28d1b049d6` : simultanément central et périphérique expérimental de BlueZ,
- `"15c0a148-c273-11ea-b3de-0242ac130004` : vie privée LL expérimentale de BlueZ,
- `330859bc-7506-492d-9370-9a6f0614037f` : rapport de qualité Bluetooth expérimental de BlueZ,
- `a6695ace-ee7f-4fb9-881a-5fac66c629af` : codecs de déchargement expérimentaux de BlueZ.

`remote-name-request-retry-delay` (par défaut : `300`)

La durée pour éviter de réessayer de résoudre le nom d'un pair, si l'essai précédent a échoué.

`page-scan-type` (par défaut : `#f`)

type d'activité de scan de page BR/EDR.

`page-scan-interval` (par défaut : `#f`)

Intervalle d'activité de scan de page BR/EDR.

`page-scan-window` (par défaut : `#f`)

Fenêtre d'activité de scan de page BR/EDR.

- `inquiry-scan-type` (par défaut : `#f`)  
Type d'activité de scan de requête BR/EDR.
- `inquiry-scan-interval` (par défaut : `#f`)  
Intervalle d'activité de scan de requête BR/EDR.
- `inquiry-scan-window` (par défaut : `#f`)  
Fenêtre d'activité de scan de requête BR/EDR.
- `link-supervision-timeout` (par défaut : `#f`)  
Délai d'attente de supervision du lien BR/EDR.
- `page-timeout` (par défaut : `#f`)  
Délai d'attente de page BR/EDR.
- `min-sniff-interval` (par défaut : `#f`)  
Intervalle de sniff minimum BR/EDR.
- `max-sniff-interval` (par défaut : `#f`)  
Intervalle de sniff maximum BR/EDR.
- `min-advertisement-interval` (par défaut : `#f`)  
Intervalle d'annonce minimum LE (utilisé pour les anciens types d'annonces uniquement).
- `max-advertisement-interval` (par défaut : `#f`)  
Intervalle d'annonce maximum LE (utilisé pour les anciens types d'annonce uniquement).
- `multi-advertisement-rotation-interval` (par défaut : `#f`)  
Intervalle de rotation des multi-annonces LE.
- `scan-interval-auto-connect` (par défaut : `#f`)  
Intervalle de scan LE utilisé pour le scan passif avec connexion automatique.
- `scan-window-auto-connect` (par défaut : `#f`)  
Fenêtre de scan LE utilisé pour le scan passive avec prise en charge de la connexion automatique.
- `scan-interval-suspend` (par défaut : `#f`)  
Intervalle de scan LE utilisé pour le scan actif avec prise en charge du réveil.
- `scan-window-suspend` (par défaut : `#f`)  
Fenêtre de scan LE utilisée pour le scan actif avec prise en charge du réveil.
- `scan-interval-discovery` (par défaut : `#f`)  
Intervalle de scan LE utilisé pour le scan actif avec prise en charge de la découverte.
- `scan-window-discovery` (par défaut : `#f`)  
Fenêtre de scan LE utilisée pour le scan actif avec prise en charge de la découverte.

- scan-interval-adv-monitor** (par défaut : #f)  
Intervalle de scan LE utilisé pour le scan passif avec prise en charge des API de surveillance des annonces.
- scan-window-adv-monitor** (par défaut : #f)  
Fenêtre de scan LE utilisée pour le scan passif avec prise en charge des API de surveillance des annonces.
- scan-interval-connect** (par défaut : #f)  
Intervalle de scan LE utilisé pour l'établissement de connexion.
- scan-window-connect** (par défaut : #f)  
Fenêtre de scan LE utilisée pour l'établissement de connexion.
- min-connection-interval** (par défaut : #f)  
Intervalle minimal de connexion LE par défaut. Cette valeur est remplacée par toute valeur spécifique de l'interface de chargement des paramètres de connexion.
- max-connection-interval** (par défaut : #f)  
Intervalle maximal de connexion LE par défaut. Cette valeur est remplacée par toute valeur spécifique de l'interface de chargement des paramètres de connexion.
- connection-latency** (par défaut : #f)  
Latence de connexion LE. Cette valeur est remplacée par toute valeur spécifique de l'interface de chargement des paramètres de connexion.
- connection-supervision-timeout** (par défaut : #f)  
Délai d'attente de supervision de la connexion LE par défaut. Cette valeur est remplacée par toute valeur spécifique de l'interface de chargement des paramètres de connexion.
- autoconnect-timeout** (par défaut : #f)  
Délai d'attente de connexion automatique LE par défaut. Cette valeur est remplacée par toute valeur spécifique de l'interface de chargement des paramètres de connexion.
- adv-mon-allowlist-scan-duration** (par défaut : 300)  
Durée du scan Allowlist pendant le scan entrelacé. Utilisé uniquement pour scanner des moniteurs ADV. L'unité est en msec.
- adv-mon-no-filter-scan-duration** (par défaut : 5)  
Durée du scan sans filtre pendant le scan entrelacé. Utilisé uniquement pour scanner des moniteurs ADV. L'unité est en msec.
- enable-adv-mon-interleave-scan?** (par défaut : #t)  
Active ou désactive le scan entrelacé des moniteurs d'annonce pour la batterie.
- cache** (par défaut : 'always)  
Cache des attributs GATT.

Les valeurs possibles sont :

- `'always` : toujours mettre en cache les attributs même pour les périphériques non appairés, c'est recommandé car l'option est la plus adaptée pour l'interopérabilité, avec des temps de reconnections plus constants, et cela active le suivi correct des notifications de tous les appareils
- `'yes` : ne met en cache que les attributs des périphériques appairés
- `'no` : ne met aucun attribut en cache.

`key-size` (par défaut : 0)

Minimum requis pour la taille de clé de chiffrement pour accéder aux caractéristiques sécurisées.

Les valeurs possibles sont :

- 0 : on s'en fiche
- $7 \leq N \leq 16$

`exchange-mtu` (par défaut : 517)

Taille de MTU pour les échanges. Les valeurs possibles sont :

- $23 \leq N \leq 517$

`att-channels` (par défaut : 3)

Nombre de canaux ATT. Les valeurs possibles sont :

- 1 : désactive EATT
- $2 \leq N \leq 5$

`session-mode` (par défaut : `'basic`)

Mode de canal AVDTP L2CAP.

Les valeurs possibles sont :

- `'basic` : utiliser le mode de base L2CAP
- `'ertm` : utiliser le mode de retransmission amélioré L2CAP.

`stream-mode` (par défaut : `'basic`)

Mode de canal de transport AVDTP L2CAP.

Les valeurs possibles sont :

- `'basic` : utiliser le mode de base L2CAP
- `'streaming` : utiliser le mode de flux L2CAP.

`reconnect-uuids` (par défaut : `'()`)

ReconnectUUIDs défini l'ensemble des services distants auxquels on devrait essayer de se reconnecter en cas de perte de lien (délai d'attente dépassés sur la supervision du lien). Le greffon de politique devrait contenir un ensemble de valeurs par défaut correct, mais cette liste peut être modifiée ici. En indiquant la liste vide, la fonctionnalité de reconnexion est désactivée.

Les valeurs possibles sont :

- `'()`

- (list (uuid <uuid-1>) (uuid <uuid-2>) ...).

**reconnect-attempts** (par défaut : 7)

Définit le nombre de tentatives de reconnexion après la perte d'un lien. La valeur 0 désactive la fonctionnalité de reconnexion.

**reconnect-intervals** (par défaut : '(1 2 4 8 16 32 64))

Définit une liste d'intervalles en secondes à utiliser entre les tentatives. Si le nombre de tentatives défini dans *reconnect-attempts* est plus grand que la liste d'intervalles, le dernier intervalle est répété jusqu'à la dernière tentative.

**auto-enable?** (par défaut : #f)

Définit l'option pour activer tous les contrôleurs quand ils sont trouvés. Cela comprend les adaptateurs présents au démarrage ainsi que les adaptateurs branchés plus tard.

**resume-delay** (par défaut : 2)

Les périphériques audio qui ont été déconnectés à cause d'une hibernation seront reconnectés au redémarrage. *resume-delay* détermine le délai entre le retour du contrôleur de veille et la tentative de connexion. Un délai plus long est préférable pour une meilleure coexistence avec le Wi-Fi. La valeur est en secondes.

**rss-sampling-period** (par défaut : #xFF)

Période d'échantillonnage RSSI par défaut. C'est utilisé quand un client enregistre un moniteur d'annonce et laisse `RSSISamplingPeriod` sans valeur.

Les valeurs possibles sont :

- #x0 : rapporte toutes les annonces
- N = #xXX : rapporte les annonces toutes les  $N \times 100$  msec (de #x01 à #xFE)
- #xFF : rapporte seulement une annonce par périphérique pendant la période de monitoring.

**gnome-keyring-service-type** [Variable Scheme]

C'est le type de service qui ajoute GNOME Keyring (<https://wiki.gnome.org/Projects/GnomeKeyring>). Sa valeur est un objet `gnome-keyring-configuration` (voir plus bas).

Ce service ajoute le paquet `gnome-keyring` au profil du système et étend PAM avec des entrées avec `pam_gnome_keyring.so`, permettant de déverrouiller un portecleé utilisateur à la connexion ou de spécifier son mot de passe avec `passwd`.

**gnome-keyring-configuration** [Type de données]

Enregistrement de la configuration du service GNOME Keyring.

**keyring** (par défaut : `gnome-keyring`)

Le paquet GNOME keyring à utiliser.

**pam-services**

Une liste de paires (*service . type*) dénotant des services PAM à étendre, où *service* est le nom d'un service existant à étendre et *type* est `login` ou `passwd`.

S'il s'agit de `login`, cela ajoute un `pam_gnome_keyring.so` facultatif au bloc d'authentification sans argument et au bloc de session avec `auto_start`. S'il s'agit de `passwd`, cela ajoute un `pam_gnome_keyring.so` facultatif au bloc de mot de passe sans argument.

Par défaut, ce champ contient « `gdm-password` » avec la valeur `login` et « `passwd` » avec la valeur `passwd`.

**seatd-service-type** [Variable Scheme]

`seatd` (<https://sr.ht/~kennylevinsen/seatd/>) est un démon de gestion de siège.

La gestion des sièges prend en charge la médiation de l'accès aux périphériques partagés (d'affichage, de saisie), sans nécessiter que les applications n'aient besoin d'être `root`.

```
(append
  (list
    ;; s'assure que seatd est lancé
    (service seatd-service-type))

    ;; normalement vous avez besoin de %base-services
    %base-services)
```

`seatd` agit sur un socket domaine UNIX, et `libseat` fournit le côté client du protocole. Les applications qui ont accès aux ressources partagées par `seatd` (p. ex. `sway`) doivent pouvoir parler sur ce socket. Vous pouvez y arriver en ajoutant l'utilisateur sous lequel ils tournent au groupe auquel appartient le socket de `seatd` (habituellement « `seat` »), comme ceci :

```
(user-account
  (name "alice")
  (group "users")
  (supplementary-groups ("wheel" ; permet d'utiliser sudo, etc.
                        "seat" ; gestion des sièges
                        "audio" ; carte son
                        "video" ; périphériques réseaux comme les webcams
                        "cdrom")) ; le bon vieux CD-ROM

  (comment "Bob's sister"))
```

En fonction de votre configuration, vous devrez non seulement ajouter des utilisateur normaux, mais aussi des utilisateurs systèmes à ce groupe. Par exemple, certains écrans d'accueil `greetd` nécessitent les graphismes et doivent donc négocier avec `seatd`.

**seatd-configuration** [Type de données]

Enregistrement de la configuration du service du démon `seatd`.

`seatd` (par défaut : `seatd`)

Le paquet `seatd` à utiliser.

**group** (par défaut : `"seat"`)  
Groupe qui possède le socket seatd.

**socket** (par défaut : `"/run/seatd.sock"`)  
Indique où créer le socket seatd.

**logfile** (par défaut : `"/var/log/seatd.log"`)  
Fichier journal où écrire.

**loglevel** (par défaut : `"error"`)  
Niveau de journalisation pour l’affichage des journaux. Les valeurs possibles sont : `"silent"`, `"error"`, `"info"` et `"debug"`.

### 12.9.10 Services de son

Le module (`gnu services sound`) fournit un service pour configurer le système ALSA (architecture son linux avancée), qui fait de PulseAudio le pilote de sortie préféré d’ALSA.

**alsa-service-type** [Variable Scheme]  
C’est le type pour le système Advanced Linux Sound Architecture (<https://alsa-project.org/>) (ALSA), qui génère le fichier de configuration `/etc/asound.conf`. La valeur de ce type est un enregistrement `alsa-configuration` comme dans cet exemple :

```
(service alsa-service-type)
```

Voir plus bas pour des détails sur `alsa-configuration`.

**alsa-configuration** [Type de données]  
Type de données représentant la configuration pour `alsa-service`.

**alsa-plugins** (par défaut : `alsa-plugins`)  
Le paquet `alsa-plugins` à utiliser.

**pulseaudio?** (par défaut : `#t`)  
Indique si les applications ALSA devraient utiliser le serveur de son PulseAudio (<https://www.pulseaudio.org/>) de manière transparente pour elles.

Utiliser PulseAudio vous permet dans lancer plusieurs applications qui produisent du son en même temps et de les contrôler individuellement via `pavucontrol` entre autres choses.

**extra-options** (par défaut : `"`)  
Chaîne à ajouter au fichier `/etc/asound.conf`.

Les utilisateurs individuels qui veulent modifier la configuration système d’ALSA peuvent le faire avec le fichier `~/.asoundrc` :

```
# Dans guix, il faut spécifier le chemin absolu des greffons.
pcm_type.jack {
  lib "/home/alice/.guix-profile/lib/alsa-lib/libasound_module_pcm_jack.so"
}
```

```
# Faire passer ALSA par Jack :
```

```
# <http://jackaudio.org/faq/routing_alsa.html>.
pcm.rawjack {
    type jack
    playback_ports {
        0 system:playback_1
        1 system:playback_2
    }

    capture_ports {
        0 system:capture_1
        1 system:capture_2
    }
}

pcm.!default {
    type plug
    slave {
        pcm "rawjack"
    }
}
```

Voir <https://www.alsa-project.org/main/index.php/Asoundrc> pour les détails.

**pulseaudio-service-type** [Variable Scheme]

C'est le type de service pour le serveur de son PulseAudio (<https://www.pulseaudio.org/>). Sa valeur est un **pulseaudio-configuration** (voir plus bas).

**Attention:** Ce service prend le pas sur les fichiers de configuration utilisateurs. Si vous voulez que PulseAudio prenne en compte les fichiers de configuration de `~/.config/pulse` vous devez réinitialiser les variables d'environnement `PULSE_CONFIG` et `PULSE_CLIENTCONFIG` dans votre `~/.bash_profile`.

**Attention:** Ce service seul ne s'assure pas que le paquet `pulseaudio` existe sur votre machine. Il ajoute seulement des fichiers de configuration, comme décrit plus bas. Dans le cas (vraiment improbable) où vous n'auriez pas de paquet `pulseaudio`, vous pouvez l'activer à travers le service `alsa-service-type` plus haut.

**pulseaudio-configuration** [Type de données]

Type de données représentant la configuration pour `pulseaudio-service`.

**client-conf** (par défaut : '() )

Liste les paramètres à indiquer dans `client.conf`. Accepte une liste de chaînes ou de paires symboles-valeurs. Une chaîne sera insérée telle quelle avec un saut de ligne. Une paire sera formatée en « clé = valeur » avec un saut de ligne.

**daemon-conf** (par défaut : '((flat-volumes . no)))

Liste les paramètres à indiquer dans `daemon.conf`, formaté comme `client-conf`.



```
script-file (par défaut : (file-append pulseaudio
"/etc/pulse/default.pa"))
```

Le fichier de script à utiliser en tant que `default.pa`. Si le champ `extra-script-files` ci-dessous est utilisé, une directive `.include` pointant vers `/etc/pulse/default.pa.d` est ajoutée au script fourni.

```
extra-script-files (par défaut : '()')
```

Une liste d'objet simili-fichiers qui définissent des scripts PulseAudio supplémentaires à lancer à l'initialisation du démon `pulseaudio`, après le `script-file` principal. Les scripts sont déployés dans le répertoire `/etc/pulse/default.pa.d`. Ils devraient avoir l'extension de nom de fichier `.pa`. Pour une référence sur les commandes disponibles, consultez `man pulse-cli-syntax`.

```
system-script-file (par défaut : (file-append pulseaudio
"/etc/pulse/system.pa"))
```

Fichier de script à utiliser comme `system.pa`.

L'exemple ci-dessous initialise le profil de carte PulseAudio par défaut, le drain par défaut et la source par défaut à utiliser pour une ancienne carte son SoundBlaster Audigy :

```
(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")))))
```

Remarquez que `pulseaudio-service-type` fait partie de `%desktop-services`. Si votre déclaration de système d'exploitation dérive d'un des modèles pour ordinateur de bureau, vous devrez ajuster l'exemple ci-dessus pour modifier le service `pulseaudio-service-type` existant avec `modify-services` (voir Section 12.18.3 [Référence de service], page 602), au lieu de le définir à nouveau.

**ladspa-service-type** [Variable Scheme]

Ce service initialise la variable `LADSPA_PATH` pour que les programmes qui la respectent, p. ex. PulseAudio, puissent charger les greffons LADSPA.

L'exemple suivant met en place le service pour activer les modules du paquet `swh-plugins` :

```
(service ladspa-service-type
 (ladspa-configuration (plugins (list swh-plugins))))
```

Voir <http://plugin.org.uk/ladspa-swh/docs/ladspa-swh.html> pour les détails.

### 12.9.11 Services de bases de données

Le module (`gnu services databases`) fournit les services suivants.

## PostgreSQL

L'exemple suivant décrit un service PostgreSQL avec la configuration par défaut.

```
(service postgresql-service-type
  (postgresql-configuration
    (postgresql postgresql-10)))
```

Si les services ne démarrent pas, cela peut être dû à une grappe incompatible déjà présente dans *data-directory*. Ajustez-la (ou, si vous n'avez plus besoin de la grappe, supprimez *data-directory*) puis redémarrez le service.

L'authentification des pairs est utilisé par défaut et le compte utilisateur `postgres` n'a pas de shell, ce qui évite l'exécution directe de `psql` en tant que cet utilisateur. Pour utiliser `psql`, vous pouvez vous authentifier temporairement en tant que `postgres` avec un shell, créer un superutilisateur PostgreSQL avec le même nom qu'un des utilisateurs du système et créer la base de données associée.

```
sudo -u postgres -s /bin/sh
createuser --interactive
createdb $MY_USER_LOGIN      # Remplacez comme vous voulez.
```

`postgresql-configuration` [Type de données]  
Type de données représentant la configuration de `postgresql-service-type`.

`postgresql`

Le paquet PostgreSQL à utiliser pour ce service.

`port` (par défaut : 5432)

Port sur lequel PostgreSQL écoutera.

`locale` (par défaut : "en\_US.utf8")

Paramètre linguistique à utiliser par défaut lors de la création de la grappe de bases de données.

`config-file` (par défaut : (postgresql-config-file))

Le fichier de configuration à utiliser pour lancer PostgreSQL. Le comportement par défaut est d'utiliser l'enregistrement `postgresql-config-file` avec les valeurs par défaut pour les champs.

`log-directory` (default: "/var/log/postgresql")

Le répertoire où la sortie de `pg_ctl` sera écrite, dans un fichier nommé "`pg_ctl.log`". Ce fichier peut être utile pour déboguer les erreurs de configuration de PostgreSQL par exemple.

`data-directory` (par défaut : "/var/lib/postgresql/data")

Répertoire dans lequel stocker les données.

`extension-packages` (par défaut : '()')

Des extensions supplémentaires peuvent être chargées à partir de paquets listés dans *extension-packages*. Les extensions sont disponibles à l'exécution. Par exemple, pour créer une base de données géographique avec l'extension `postgis`, on peut configurer `postgresql-service` de cette manière :

```
(use-package-modules databases geo)
```

```

(operating-system
  ...
  ;; postgresql est requis pour lancer `psql' mais postgis n'est pas re
  ;; bon fonctionnement.
  (packages (cons* postgresql %base-packages))
  (services
    (cons*
      (service postgresql-service-type
        (postgresql-configuration
          (postgresql postgresql-10)
          (extension-packages (list postgis))))
      %base-services)))

```

Ensuite l'extension devient visible et vous pouvez initialiser une base de données géographique de cette manière :

```

psql -U postgres
> create database postgistest;
> \connect postgistest;
> create extension postgis;
> create extension postgis_topology;

```

Vous n'avez pas besoin d'ajouter ce champ pour les extensions « contrib » comme hstore ou dblink comme elles sont déjà exploitables par postgresql. Ce champ n'est requis que pour ajouter des extensions fournies par d'autres paquets.

**postgresql-config-file** [Type de données]

Type de données représentant le fichier de configuration de PostgreSQL. Comme montré dans l'exemple suivant, on peut l'utiliser pour personnaliser la configuration de PostgreSQL. Remarquez que vous pouvez utiliser n'importe quelle G-expression ou nom de fichier à la place de cet enregistrement, si par exemple vous avez déjà un fichier de configuration que vous voulez utiliser.

```

(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` (par défaut : "syslog")

La méthode de journalisation de PostgreSQL. Plusieurs valeurs sont possibles, séparées par des virgules.

`hba-file` (par défaut : %default-postgres-hba)

Nom de fichier ou G-expression pour la configuration de l'authentification par hôte.

`ident-file` (par défaut : %default-postgres-ident)

Nom de fichier ou G-expression pour la configuration de la correspondance de noms d'utilisateurs.

`socket-directory` (par défaut : "/var/run/postgresql")

Spécifie le répertoire des sockets Unix-domain sur lesquels PostgreSQL écoutera les connexion d'applications clientes. Si la valeur est "", PostgreSQL n'écoute sur aucun socket Unix-domain, auquel cas seuls les socket TCP/IP permettent de se connecter au serveur.

Par défaut, la valeur #false signifie que la valeur par défaut de PostgreSQL sera utilisée, actuellement '/tmp'.

`extra-config` (par défaut : '()')

Liste de clés et valeurs supplémentaires à inclure dans le fichier de configuration de PostgreSQL. Chaque entrée dans la liste devrait être une liste où le premier élément est la clé, et les autres éléments sont les valeurs.

Les valeurs peuvent être des nombres, des booléens ou des chaînes et seront transformées en types de paramètres de PostgreSQL Boolean, String, Numeric, Numeric with Unit et Enumerated décrits ici (<https://www.postgresql.org/docs/current/config-setting.html>).

`postgresql-role-service-type` [Variable Scheme]

Ce service permet de créer des rôles PostgreSQL et des bases de données après le démarrage du service PostgreSQL. Voici un exemple d'utilisation.

```

(service postgresql-role-service-type
  (postgresql-role-configuration
    (roles
      (list (postgresql-role
            (name "test")
            (create-database? #t))))))

```

Ce service peut être étendu avec des rôles supplémentaires, comme dans cet exemple :

```

(service-extension postgresql-role-service-type
  (const (postgresql-role
        (name "alice")
        (create-database? #t))))

```

**postgresql-role** [Type de données]

PostgreSQL gère les permissions d'accès à la base de données en utilisant le concept de rôle. Un rôle est soit comme un utilisateur de la base de données, soit un groupe d'utilisateurs, en fonction du paramétrage du rôle. Les rôles peuvent posséder des objets de la base (par exemple des tables) et peuvent assigner des privilèges sur ces objets à d'autres rôles pour contrôler qui a accès à quels objets.

**name** Le nom du rôle.

**permissions** (par défaut : '(createdb login))

La liste des permissions du rôle. Les permissions prises en charge sont **bypassrls**, **createdb**, **createrole**, **login**, **replication** et **superuser**.

**create-database?** (par défaut : #f)

Indique s'il faut créer une base de données avec le même nom que le rôle.

**postgresql-role-configuration** [Type de données]

Type de données représentant la configuration de *postgresql-role-service-type*.

**host** (par défaut : "/var/run/postgresql")

L'hôte PostgreSQL sur lequel se connecter.

**log** (par défaut : "/var/log/postgresql\_roles.log")

Nom du fichier du fichier de journal.

**roles** (par défaut : '() )

Les rôles PostgreSQL initiaux à créer.

## MariaDB/MySQL

**mysql-service-type** [Variable Scheme]

C'est le type de service pour une base de donnée MySQL ou MariaDB. Sa valeur est un objet **mysql-configuration**, qui spécifie le paquet à utiliser ainsi que divers paramètres pour le démon **mysqld**.

**mysql-configuration** [Type de données]

Type de données représentant la configuration de *mysql-service-type*.

**mysql** (par défaut : *mariadb*)

Objet paquet du serveur de base de données MySQL, qui peut être soit *mariadb*, soit *mysql*.

Pour MySQL, un mot de passe root temporaire sera affiché à l'activation.

Pour MariaDB, le mot de passe root est vide.

**bind-address** (par défaut : "127.0.0.1")

L'IP sur laquelle écouter les connexions réseau. Utilisez "0.0.0.0" pour écouter sur toutes les interfaces réseaux.

**port** (par défaut : 3306)

Port TCP sur lequel le serveur de base de données écoute les connexions entrantes.

**socket** (par défaut : "/run/mysqld/mysqld.sock")

Fichier de socket à utiliser pour les connexions locales (pas via le réseau).

- extra-content** (par défaut : "")  
Paramètres supplémentaires pour le fichier de configuration `my.cnf`.
- extra-environment** (par défaut : #~'())  
Liste de variables d'environnement passées au processus `mysqld`.
- auto-upgrade?** (par défaut : #t)  
Indique s'il faut automatiquement lancer `mysql_upgrade` après avoir démarré le service. C'est nécessaire pour mettre à jour le *schéma système* après une mise à jour « majeure » (comme de passer de MariaDB 10.4 à 10.5), mais peut être désactivé si vous voulez plutôt le faire manuellement.

## Memcached

- memcached-service-type** [Variable Scheme]  
C'est le type de service pour le service Memcached (<https://memcached.org/>) qui fournit un cache en mémoire distribué. La valeur pour le type de service est un objet `memcached-configuration`.  
(`service memcached-service-type`)
- memcached-configuration** [Type de données]  
Type de données représentant la configuration de memcached.
- memcached** (par défaut : `memcached`)  
Le paquet Memcached à utiliser.
- interfaces** (par défaut : `'("0.0.0.0")`)  
Les interfaces réseaux sur lesquelles écouter.
- tcp-port** (par défaut : `11211`)  
Port sur lequel accepter les connexions.
- udp-port** (par défaut : `11211`)  
Port sur lequel accepter les connexions UDP, une valeur de 0 désactive l'écoute en UDP.
- additional-options** (par défaut : `'()`)  
Options de la ligne de commande supplémentaires à passer à `memcached`.

## Redis

- redis-service-type** [Variable Scheme]  
C'est le type de service pour la base clef-valeur Redis (<https://redis.io/>) dont la valeur est un objet `redis-configuration`.
- redis-configuration** [Type de données]  
Type de données représentant la configuration de redis.
- redis** (par défaut : `redis`)  
Le paquet Redis à utiliser.
- bind** (par défaut : `"127.0.0.1"`)  
Interface réseau sur laquelle écouter.

**port** (par défaut : 6379)  
 Port sur lequel accepter les connexions, une valeur de 0 désactive l'écoute sur un socket TCP.

**working-directory** (par défaut : `"/var/lib/redis"`)  
 Répertoire dans lequel stocker la base de données et les fichiers liés.

### 12.9.12 Services de courriels

Le module (`gnu services mail`) fournit des définitions de services Guix pour les services de courriel : des serveurs IMAP, POP3 et LMTP ainsi que des MTA (Mail Transport Agent). Que d'acronymes ! Ces services sont détaillés dans les sous-sections ci-dessous.

#### Service Dovecot

**dovecot-service** [`#:config (dovecot-configuration)`] [Procédure Scheme]  
 Renvoie un service qui lance le serveur de courriel IMAP/POP3/LMTP Dovecot.

Par défaut, Dovecot n'a pas besoin de beaucoup de configuration ; l'objet de configuration par défaut créé par (`dovecot-configuration`) suffira si votre courriel est livré dans `~/Maildir`. Un certificat auto-signé sera généré pour les connexions TLS, bien que Dovecot écoutera aussi sur les ports non chiffrés par défaut. Il y a quelques options cependant, que les administrateurs peuvent avoir besoin de changer et comme c'est le cas avec d'autres services, Guix permet aux administrateurs systèmes de spécifier ces paramètres via une interface Scheme unifiée.

Par exemple, pour spécifier que les courriels se trouvent dans `maildir~/mail`, on peut instancier Dovecot de cette manière :

```
(dovecot-service #:config
  (dovecot-configuration
    (mail-location "maildir:~/mail")))
```

Les paramètres de configuration disponibles sont les suivants. Chaque définition des paramètres est précédé par son type ; par exemple, `'string-list toto'` indique que le paramètre `toto` devrait être spécifié comme une liste de chaînes de caractères. Il y a aussi une manière de spécifier la configuration comme une chaîne de caractères, si vous avez un vieux fichier `dovecot.conf` que vous voulez porter depuis un autre système ; voir la fin pour plus de détails.

Les champs de `dovecot-configuration` disponibles sont :

**package dovecot** [paramètre de `dovecot-configuration`]  
 Le paquet dovecot.

**comma-separated-string-list listen** [paramètre de `dovecot-configuration`]  
 Une liste d'IP ou d'hôtes à écouter pour les connexions. `*` écoute sur toutes les interfaces IPv4, `:::` écoute sur toutes les interfaces IPv6. Si vous voulez spécifier des ports différents de la valeur par défaut ou quelque chose de plus complexe, complétez les champs d'adresse et de port de `'inet-listener'` des services spécifiques qui vous intéressent.

**protocol-configuration-list** [paramètre de **dovecot-configuration**]  
**protocols**

Liste des protocoles que vous voulez servir. Les protocoles disponibles comprennent 'imap', 'pop3' et 'lmtp'.

Les champs **protocol-configuration** disponibles sont :

**string name** [paramètre de **protocol-configuration**]  
 Le nom du protocole.

**string auth-socket-path** [paramètre de **protocol-configuration**]  
 Le chemin d'un socket UNIX vers le serveur d'authentification maître pour trouver les utilisateurs. C'est utilisé par imap (pour les utilisateurs partagés) et lda. Sa valeur par défaut est `"/var/run/dovecot/auth-userdb"`.

**boolean imap-metadata?** [paramètre de **protocol-configuration**]  
 Indique s'il faut activer l'extension IMAP METADATA définie dans RFC 5464 (<https://tools.ietf.org/html/rfc5464>), qui permet aux clients d'indiquer et récupérer des métadonnées et des annotations pour chaque boîte aux lettres et chaque utilisateur par IMAP.

Si la valeur est '#t', vous devez spécifier un dictionnaire via le paramètre **mail-attribute-dict**.

La valeur par défaut est '#f'.

**space-separated-string-list** [paramètre de **protocol-configuration**]  
**managesieve-notify-capabilities**

Les capacités NOTIFY à rapporter aux clients qui se connectent pour la première fois au service ManageSieve, avant l'authentification. Elles peuvent être différentes des capacités offertes aux utilisateurs authentifiés. Si ce champ est vide, rapporte ce que l'interpréteur Sieve prend en charge par défaut.

La valeur par défaut est '()'.

**space-separated-string-list** [paramètre de **protocol-configuration**]  
**managesieve-sieve-capability**

Les capacités SIEVE à rapporter aux clients qui se connectent pour la première fois au service ManageSieve, avant l'authentification. Elles peuvent être différentes des capacités offertes aux utilisateurs authentifiés. Si ce champ est vide, rapporte ce que l'interpréteur Sieve prend en charge par défaut.

La valeur par défaut est '()'.

**space-separated-string-list** [paramètre de **protocol-configuration**]  
**mail-plugins**

Liste de greffons à charger séparés par des espaces.

**entier-non-négatif** [paramètre de **protocol-configuration**]  
**mail-max-userip-connections**

Nombre maximum de connexions IMAP permises pour un utilisateur depuis chaque adresse IP. Remarque : la comparaison du nom d'utilisateur est sensible à la casse. Par défaut '10'.



**service-configuration-list services** [paramètre de `dovecot-configuration`]  
 Liste des services à activer. Les services disponibles comprennent `imap`, `imap-login`, `pop3`, `pop3-login`, `auth` et `lmtp`.

Les champs de `service-configuration` disponibles sont :

**string kind** [paramètre de `service-configuration`]  
 Le type de service. Les valeurs valides comprennent `director`, `imap-login`, `pop3-login`, `lmtp`, `imap`, `pop3`, `auth`, `auth-worker`, `dict`, `tcpwrap`, `quota-warning` ou n'importe quoi d'autre.

**listener-configuration-list listeners** [paramètre de `service-configuration`]

Les auditeurs du service. Un auditeur est soit un `unix-listener-configuration`, soit un `fifo-listener-configuration`, soit un `inet-listener-configuration`. La valeur par défaut est `()`.

Les champs de `unix-listener-configuration` disponibles sont :

**string path** [paramètre de `unix-listener-configuration`]  
 Chemin vers le fichier, relativement au champ `base-dir`. C'est aussi utilisé comme nom de section.

**string mode** [paramètre de `unix-listener-configuration`]  
 Le mode d'accès pour le socket. La valeur par défaut est `"0600"`.

**string user** [paramètre de `unix-listener-configuration`]  
 L'utilisateur à qui appartient le socket. La valeur par défaut est `""`.

**string group** [paramètre de `unix-listener-configuration`]  
 Le groupe auquel appartient le socket. La valeur par défaut est `""`.

Les champs de `fifo-listener-configuration` disponibles sont :

**string path** [paramètre de `fifo-listener-configuration`]  
 Chemin vers le fichier, relativement au champ `base-dir`. C'est aussi utilisé comme nom de section.

**string mode** [paramètre de `fifo-listener-configuration`]  
 Le mode d'accès pour le socket. La valeur par défaut est `"0600"`.

**string user** [paramètre de `fifo-listener-configuration`]  
 L'utilisateur à qui appartient le socket. La valeur par défaut est `""`.

**string group** [paramètre de `fifo-listener-configuration`]  
 Le groupe auquel appartient le socket. La valeur par défaut est `""`.

Les champs de `inet-listener-configuration` disponibles sont :

**string protocol** [paramètre de `inet-listener-configuration`]  
 Le protocole à écouter.

- string address** [paramètre de `inet-listener-configuration`]  
L'adresse sur laquelle écouter, ou la chaîne vide pour toutes les adresses.  
La valeur par défaut est `''`.
- entier-non-négatif port** [paramètre de `inet-listener-configuration`]  
Le port sur lequel écouter.
- boolean ssl?** [paramètre de `inet-listener-configuration`]  
S'il faut utiliser SSL pour ce service ; `'yes'`, `'no'` ou `'required'`. La valeur par défaut est `'#t'`.
- entier-non-négatif client-limit** [paramètre de `service-configuration`]  
Connexions de clients simultanées maximum par processus. Une fois ce nombre de connexions atteint, la connexion suivante fera en sorte que Dovecot démarre un autre processus. Si la valeur est 0, `default-client-limit` est utilisé à la place.  
La valeur par défaut est `'0'`.
- entier-non-négatif service-count** [paramètre de `service-configuration`]  
Nombre de connexions à gérer avant de démarrer un nouveau processus. Typiquement les valeurs utiles sont 0 (sans limite) ou 1. 1 est plus sûr, mais 0 est plus rapide. [<doc/wiki/LoginProcess.txt>](#). La valeur par défaut est `'1'`.
- entier-non-négatif process-limit** [paramètre de `service-configuration`]  
Nombre de processus maximum qui peut exister pour ce service. Si la valeur est 0, `default-process-limit` est utilisé à la place.  
La valeur par défaut est `'0'`.
- entier-non-négatif process-min-avail** [paramètre de `service-configuration`]  
Nombre de processus à toujours garder en attente de connexions. La valeur par défaut est `'0'`.
- entier-non-négatif vsz-limit** [paramètre de `service-configuration`]  
Si vous mettez `'service-count 0'`, vous avez sans doute besoin d'augmenter ce paramètre. La valeur par défaut est `'256000000'`.
- dict-configuration dict** [paramètre de `dovecot-configuration`]  
Configuration du dictionnaire, créé par le constructeur `dict-configuration`.  
Les champs de `dict-configuration` disponibles sont :
- free-form-fields entries** [paramètre de `dict-configuration`]  
Une liste de paires de clefs-valeurs que ce dictionnaire contient. La valeur par défaut est `'()`.

**passdb-configuration-list** **passdbs** [paramètre de **dovecot-configuration**]  
 Une liste de configurations **passdb**, chacune créée par le constructeur **passdb-configuration**.

Les champs de **passdb-configuration** disponibles sont :

**string driver** [paramètre de **passdb-configuration**]  
 Le pilote à utiliser par **passdb**. Les valeur valides comprennent ‘**pam**’, ‘**passwd**’, ‘**shadow**’, ‘**bsdauth**’ et ‘**static**’. La valeur par défaut est “**pam**”.

**space-separated-string-list** [paramètre de **passdb-configuration**]  
**args**  
 Liste d’arguments pour le pilote **passdb** séparés par des espaces. La valeur par défaut est “”.

**userdb-configuration-list** **userdbs** [paramètre de **dovecot-configuration**]  
 Liste des configurations **userdb**, chacune créée par le constructeur **userdb-configuration**.

Les champs de **userdb-configuration** disponibles sont :

**string driver** [paramètre de **userdb-configuration**]  
 Le pilote que **userdb** devrait utiliser. Les valeurs valides comprennent ‘**passwd**’ et ‘**static**’. La valeur par défaut est “**passwd**”.

**space-separated-string-list** [paramètre de **userdb-configuration**]  
**args**  
 Liste des arguments du pilote **userdb** séparés par des espaces. La valeur par défaut est “”.

**free-form-args override-fields** [paramètre de **userdb-configuration**]  
 Remplace des champs de **passwd**. La valeur par défaut est ‘()’.

**plugin-configuration** [paramètre de **dovecot-configuration**]  
**plugin-configuration**  
 Configuration du greffon, créé par le constructeur **plugin-configuration**.

**list-of-namespace-configuration** [paramètre de **dovecot-configuration**]  
**namespaces**  
 Liste d’espaces de noms. Chaque élément de la liste est créé par le constructeur **namespace-configuration**.

Les champs de **namespace-configuration** disponibles sont :

**string name** [paramètre de **namespace-configuration**]  
 Nom de cet espace de nom.

**string type** [paramètre de **namespace-configuration**]  
 Type d’espace de nom : ‘**private**’, ‘**shared**’ ou ‘**public**’. La valeur par défaut est “**private**”.

- string separator** [paramètre de namespace-configuration]  
Séparateur de hiérarchie à utiliser. Vous devriez utiliser le même séparateur pour tous les espaces de noms ou certains clients seront confus. '/' est généralement une bonne valeur. La valeur par défaut dépend cependant du format de stockage sous-jacent. La valeur par défaut est "".
- string prefix** [paramètre de namespace-configuration]  
Préfixe requis pour accéder à cet espace de nom. Ce paramètre doit être différent pour tous les espaces de noms. Par exemple 'Public/'. La valeur par défaut est "".
- string location** [paramètre de namespace-configuration]  
Emplacement physique de la boîte aux lettres. C'est le même format que mail\_location, qui est aussi la valeur par défaut. La valeur par défaut est "".
- boolean inbox?** [paramètre de namespace-configuration]  
Il ne peut y avoir qu'un INBOX, et ce paramètre définit l'espace de nom qui le possède. La valeur par défaut est '#f'.
- boolean hidden?** [paramètre de namespace-configuration]  
Si l'espace de nom est caché, il n'est pas annoncé auprès des clients par l'extension NAMESPACE. Vous voudrez aussi sans doute indiquer 'list? #f'. C'est surtout utile lors de la conversion depuis un autre serveur avec des espaces de noms différents que vous voulez rendre obsolètes sans les casser. Par exemple vous pouvez cacher les espaces de noms avec les préfixes '~/' et '~%u/mail/'. La valeur par défaut est '#f'.
- boolean list?** [paramètre de namespace-configuration]  
Montre les boîtes aux lettres sous cet espace de nom avec la commande LIST. Cela rend l'espace de nom visible pour les clients qui ne supportent pas l'extension NAMESPACE. La valeur spéciale children liste les boîtes aux lettres filles mais cache le préfixe de l'espace de nom. La valeur par défaut est '#t'.
- boolean subscriptions?** [paramètre de namespace-configuration]  
Les espaces de noms gèrent leur propre souscription. Si la valeur est #f, l'espace de nom parent s'en charge. Le préfixe vide devrait toujours avoir cette valeur à #t. La valeur par défaut est '#t'.
- mailbox-configuration-list** [paramètre de namespace-configuration]  
**mailboxes**  
Liste des boîtes aux lettres prédéfinies dans cet espace de nom. La valeur par défaut est '()'.  
Les champs de mailbox-configuration disponibles sont :
- string name** [paramètre de mailbox-configuration]  
Nom de cette boîte aux lettres.
- string auto** [paramètre de mailbox-configuration]  
'create' créera automatiquement cette boîte aux lettres. 'subscribe' créera et souscrira à la boîte aux lettres. La valeur par défaut est "no".

**space-separated-string-list** [paramètre de mailbox-configuration]  
**special-use**

Liste des attributs SPECIAL-USE IMAP spécifiés par la RFC 6154. Les valeurs valides sont `\All`, `\Archive`, `\Drafts`, `\Flagged`, `\Junk`, `\Sent` et `\Trash`. La valeur par défaut est `'()`.

**file-name base-dir** [paramètre de dovecot-configuration]

Répertoire de base où stocker les données d'exécution. La valeur par défaut est `"/var/run/dovecot/"`.

**string login-greeting** [paramètre de dovecot-configuration]

Message d'accueil pour les clients. La valeur par défaut est `"Dovecot ready."`.

**space-separated-string-list** [paramètre de dovecot-configuration]

**login-trusted-networks**

Liste des groupes d'adresses de confiance. Les connexions depuis ces IP sont autorisées à modifier leurs adresses IP et leurs ports (pour la connexion et la vérification d'authentification). `'disable-plaintext-auth'` est aussi ignoré pour ces réseaux. Typiquement vous voudrez spécifier votre mandataire IMAP ici. La valeur par défaut est `'()`.

**space-separated-string-list** [paramètre de dovecot-configuration]

**login-access-sockets**

Liste des sockets de vérification d'accès de connexion (p. ex. tcpwrap). La valeur par défaut est `'()`.

**boolean verbose-proctitle?** [paramètre de dovecot-configuration]

Montre des titres de processus plus verbeux (dans ps). Actuellement, montre le nom d'utilisateur et l'adresse IP. Utile pour voir qui utilise en réalité les processus IMAP (p. ex. des boîtes aux lettres partagées ou si le même uid est utilisé pour plusieurs comptes). La valeur par défaut est `'#f'`.

**boolean shutdown-clients?** [paramètre de dovecot-configuration]

Indique si les processus devraient toujours être tués lorsque le processus maître de Dovecot est éteint. La valeur `#f` signifie que Dovecot peut être mis à jour sans forcer les connexions clientes existantes à se fermer (bien que cela puisse être un problème si la mise à jour est un correctif de sécurité par exemple). La valeur par défaut est `'#t'`.

**entier-non-négatif** [paramètre de dovecot-configuration]

**doveadm-worker-count**

Si la valeur n'est pas zéro, lance les commandes de courriel via ce nombre de connexions au serveur dovecadm au lieu de les lancer dans le même processus. La valeur par défaut est `'0'`.

**string dovecadm-socket-path** [paramètre de dovecot-configuration]

Socket UNIX ou hôte:port utilisé pour se connecter au serveur dovecadm. La valeur par défaut est `"doveadm-server"`.

**space-separated-string-list** [paramètre de dovecot-configuration]  
**import-environment**

Liste des variables d'environnement qui sont préservées au démarrage de Dovecot et passées à tous ses processus fils. Vous pouvez aussi donner des paires clef=valeur pour toujours spécifier ce paramètre.

**boolean disable-plaintext-auth?** [paramètre de dovecot-configuration]

Désactive la commande LOGIN et toutes les autres authentifications en texte clair à moins que SSL/TLS ne soit utilisé (capacité LOGINDISABLED). Remarquez que si l'IP distante correspond à l'IP locale (c.-à-d. que vous vous connectez depuis le même ordinateur), la connexion est considérée comme sécurisée et l'authentification en texte clair est permise. Voir aussi le paramètre ssl=required. La valeur par défaut est '#t'.

**entier-non-négatif auth-cache-size** [paramètre de dovecot-configuration]

Taille du cache d'authentification (p. ex. '#e10e6'). 0 signifie qu'il est désactivé. Remarquez que bsdauth, PAM et vpopmail ont besoin que 'cache-key' soit indiqué pour que le cache soit utilisé. La valeur par défaut est '0'.

**string auth-cache-ttl** [paramètre de dovecot-configuration]

Durée de vie des données en cache. Après l'expiration du TTL l'enregistrement en cache n'est plus utilisé \*sauf\* si la requête à la base de données principale revoie une erreur interne. Nous essayons aussi de gérer les changements de mot de passe automatiquement : si l'authentification précédente de l'utilisateur était réussie mais pas celle-ci, le cache n'est pas utilisé. Pour l'instant cela fonctionne avec l'authentification en texte clair uniquement. La valeur par défaut est '"1 hour"'.

**string auth-cache-negative-ttl** [paramètre de dovecot-configuration]

TTL pour les résultats négatifs (l'utilisateur n'est pas trouvé ou le mot de passe ne correspond pas). 0 désactive la mise en cache complètement. La valeur par défaut est '"1 hour"'.

**space-separated-string-list** [paramètre de dovecot-configuration]  
**auth-realms**

Liste des domaines pour les mécanismes d'authentification SASL qui en ont besoin. Vous pouvez laisser ce paramètre vide si vous ne voulez pas utiliser plusieurs domaines. Beaucoup de clients utilisent le premier domaine listé ici, donc gardez celui par défaut en premier. La valeur par défaut est '()'.

**string auth-default-realm** [paramètre de dovecot-configuration]

Domaine par défaut à utiliser si aucun n'est spécifié. C'est utilisé pour les domaines SASL et pour ajouter @domaine au nom d'utilisateur dans les authentification en texte clair. La valeur par défaut est '""'.

**string auth-username-chars** [paramètre de dovecot-configuration]

Liste des caractères autorisés dans les noms d'utilisateur. Si le nom d'utilisateur donné par l'utilisateur contient un caractère qui n'est pas listé ici, la connexion échoue automatiquement. C'est juste une vérification supplémentaire pour s'assurer que l'utilisateur ne puisse pas exploiter des vulnérabilités potentielles d'échappement

de guillemets avec les bases de données SQL/LDAP. Si vous voulez autoriser tous les caractères, indiquez la liste vide.

**string auth-username-translation** [paramètre de dovecot-configuration]  
Traduction de caractères dans les noms d'utilisateur avant qu'ils ne soient recherchés en base. La valeur contient une série de caractère de -> à. Par exemple '#@/@' signifie que '#' et '/' sont traduits en '@'. La valeur par défaut est "".

**string auth-username-format** [paramètre de dovecot-configuration]  
Format des noms d'utilisateur avant qu'ils ne soient recherchés en base. Vous pouvez utiliser les variables standard ici, p. ex. %Lu est le nom d'utilisateur en minuscule, %n enlève le domaine s'il est donné ou '%n-AT-%d' changerait le '@' en '-AT-'. Cette traduction est faite après les changements de 'auth-username-translation'. La valeur par défaut est "%Lu".

**string auth-master-user-separator** [paramètre de dovecot-configuration]  
Si vous voulez permettre aux utilisateurs maîtres de se connecter en spécifiant le nom d'utilisateur maître dans la chaîne de nom d'utilisateur normal (c.-à-d. sans utiliser le support du mécanisme SASL pour cela), vous pouvez spécifier le caractère de séparation ici. Le format est ensuite <nom d'utilisateur><séparateur><nom d'utilisateur maître>. UW-IMAP utilise '\*' comme séparateur, donc ça pourrait être un bon choix. La valeur par défaut est "".

**string auth-anonymous-username** [paramètre de dovecot-configuration]  
Nom d'utilisateur à utiliser pour les utilisateurs qui se connectent avec le mécanisme SASL ANONYMOUS. La valeur par défaut est "anonymous".

**entier-non-négatif auth-worker-max-count** [paramètre de dovecot-configuration]  
Nombre maximum de processus de travail dovecot-auth. Ils sont utilisés pour exécuter des requêtes passdb et userdb bloquantes (p. ex. MySQL et PAM). Ils sont créés automatiquement et détruits au besoin. La valeur par défaut est '30'.

**string auth-gssapi-hostname** [paramètre de dovecot-configuration]  
Nom d'hôte à utiliser dans les noms GSSAPI principaux. La valeur par défaut est d'utiliser le nom renvoyé par gethostname(). Utilisez '\$ALL' (avec des guillemets) pour permettre toutes les entrées keytab. La valeur par défaut est "".

**string auth-krb5-keytab** [paramètre de dovecot-configuration]  
Keytab Kerberos à utiliser pour le mécanisme GSSAPI. Utilisera la valeur par défaut du système (typiquement /etc/krb5.keytab) s'il n'est pas spécifié. Vous pourriez avoir besoin de faire en sorte que le service d'authentification tourne en root pour pouvoir lire ce fichier. La valeur par défaut est "".

**boolean auth-use-winbind?** [paramètre de dovecot-configuration]  
Effectue l'authentification NTLM et GSS-SPNEGO avec le démon winbind de Samba et l'utilitaire 'ntlm-auth'. <doc/wiki/Authentication/Mechanisms/Winbind.txt>. La valeur par défaut est '#f'.

- file-name auth-winbind-helper-path** [paramètre de dovecot-configuration]  
Chemin du binaire 'ntlm-auth' de samba. La valeur par défaut est "/usr/bin/ntlm\_auth".
- string auth-failure-delay** [paramètre de dovecot-configuration]  
Durée d'attente avant de répondre à des authentifications échouées. La valeur par défaut est "2 secs".
- boolean auth-ssl-require-client-cert?** [paramètre de dovecot-configuration]  
Requiert un certification client SSL valide ou l'authentification échoue. La valeur par défaut est '#f'.
- boolean auth-ssl-username-from-cert?** [paramètre de dovecot-configuration]  
Prend le nom d'utilisateur du certificat SSL client, avec X509\_NAME\_get\_text\_by\_NID() qui renvoie le CommonName du DN du sujet. La valeur par défaut est '#f'.
- space-separated-string-list auth-mechanisms** [paramètre de dovecot-configuration]  
Liste des mécanismes d'authentification souhaités. Les mécanismes supportés sont : 'plain', 'login', 'digest-md5', 'cram-md5', 'ntlm', 'rpa', 'apop', 'anonymous', 'gssapi', 'otp', 'skey' et 'gss-spnego'. Remarquez : Voir aussi le paramètre 'disable-plaintext-auth'.
- space-separated-string-list director-servers** [paramètre de dovecot-configuration]  
Liste des IP ou des noms d'hôtes des serveurs directeurs, dont soi-même. Les ports peuvent être spécifiés avec ip:port. Le port par défaut est le même que le 'inet-listener' du service directeur. La valeur par défaut est '()'.
- space-separated-string-list director-mail-servers** [paramètre de dovecot-configuration]  
Liste des IP ou des noms d'hôtes de tous les serveurs de courriel de la grappe. Les intervalles sont aussi permis, comme 10.0.0.10-10.0.0.30. La valeur par défaut est '()'.
- string director-user-expire** [paramètre de dovecot-configuration]  
Combien de temps avant de rediriger les utilisateurs à un serveur spécifique après qu'il n'y a plus de connexion. La valeur par défaut est "15 min".
- string director-username-hash** [paramètre de dovecot-configuration]  
La manière de traduire le nom d'utilisateur avant de le hasher. Les valeurs utiles comprennent %Ln si l'utilisateur peut se connecter avec ou sans @domain, %Ld si les boîtes aux lettres sont partagées dans le domaine. La valeur par défaut est "%Lu".
- string log-path** [paramètre de dovecot-configuration]  
Fichier de journal à utiliser pour les messages d'erreur. 'syslog' journalise vers syslog, '/dev/stderr' vers la sortie d'erreur. La valeur par défaut est "syslog".



- string info-log-path** [paramètre de dovecot-configuration]  
Fichier de journal à utiliser pour les messages d'information. La valeur par défaut est 'log-path'. La valeur par défaut est "".
- string debug-log-path** [paramètre de dovecot-configuration]  
Fichier de journal à utiliser pour les messages de débogage. La valeur par défaut est 'info-log-path'. La valeur par défaut est "".
- string syslog-facility** [paramètre de dovecot-configuration]  
Dispositif syslog à utiliser si vous journalisez avec syslog. Normalement si vous ne voulez pas utiliser 'mail', vous voudrez utiliser local0..local7. D'autres dispositifs standard sont supportés. La valeur par défaut est "mail".
- boolean auth-verbose?** [paramètre de dovecot-configuration]  
Indique s'il faut enregistrer les tentatives de connexion échouées et la raison de leur échec. La valeur par défaut est '#f'.
- string auth-verbose-passwords** [paramètre de dovecot-configuration]  
Dans le cas où le mot de passe n'était pas correct, indique s'il faut enregistrer le mauvais mot de passe. Les valeurs valides sont « no », « plain » et « sha1 ». Il peut être utile d'indiquer « sha1 » pour discriminer des attaques par force brute d'utilisateurs qui réessaient encore et encore le même mot de passe. Vous pouvez aussi tronquer la valeur à n caractères en ajoutant « :n » (p. ex. « sha1:6 »). La valeur par défaut est "no".
- boolean auth-debug?** [paramètre de dovecot-configuration]  
Journaux encore plus verbeux pour le débogage. Cela montre par exemple les requêtes SQL effectuées. La valeur par défaut est '#f'.
- boolean auth-debug-passwords?** [paramètre de dovecot-configuration]  
Dans le cas où le mot de passe était incorrect, indique s'il faut enregistrer les mots de passe et les schémas utilisés pour que le problème puisse être débogué. Activer cette option active aussi 'auth-debug'. La valeur par défaut est '#f'.
- boolean mail-debug?** [paramètre de dovecot-configuration]  
Indique s'il faut activer le débogage du traitement des courriels. Cela peut vous aider à comprendre pourquoi Dovecot ne trouve pas vos courriels. La valeur par défaut est '#f'.
- boolean verbose-ssl?** [paramètre de dovecot-configuration]  
Indique s'il faut montrer les erreurs au niveau SSL. La valeur par défaut est '#f'.
- string log-timestamp** [paramètre de dovecot-configuration]  
Préfixe à utiliser devant chaque ligne écrite dans le fichier journal. Les codes % sont au format strftime(3). La valeur par défaut est "%b %d %H:%M:%S %".
- space-separated-string-list** [paramètre de dovecot-configuration]  
**login-log-format-elements**  
Liste des éléments qu'il faut enregistrer. Les éléments qui ont une variable non vide sont agrégés pour former une chaîne de mots séparés par des virgules.

**string login-log-format** [paramètre de dovecot-configuration]  
 Format du journal de connexion. %s contient la chaîne ‘login-log-format-elements’, %s contient la donnée à enregistrer. La valeur par défaut est ‘“%\$: %s”’.

**string mail-log-prefix** [paramètre de dovecot-configuration]  
 Préfixe à utiliser devant chaque ligne du fichier de journal pour les processus traitant les courriels. Voir doc/wiki/Variables.txt pour trouver la liste des variables que vous pouvez utiliser. La valeur par défaut est ‘“\”%s(%u)<{%pid}><{%session}>: \””’.

**string deliver-log-format** [paramètre de dovecot-configuration]  
 Format à utiliser pour enregistrer les livraisons de courriels. Vous pouvez utiliser ces variables :

%\$	Message de statut de la livraison (p. ex. ‘saved to INBOX’)
%m	Message-ID
%s	Objet
%f	Adresse « de »
%p	Taille physique
%w	Taille virtuelle.

La valeur par défaut est ‘“msgid=%m: %\$”’.

**string mail-location** [paramètre de dovecot-configuration]  
 Emplacement des boîtes à lettre des utilisateurs. La valeur par défaut est vide, ce qui signifie que Dovecot essaiera de trouver les boîte aux lettres automatiquement. Cela ne fonctionnera pas si l'utilisateur n'a aucun courriel, donc il vaut mieux indiquer explicitement le bon emplacement à Dovecot.

Si vous utilisez mbox, il ne suffit pas de donner le chemin vers le fichier INBOX (p. ex. /var/mail/%u). Vous devrez aussi dire à Dovecot où les autres boîtes aux lettres se trouvent. Cela s'appelle le « répertoire racine des courriels » et il doit être le premier chemin donné à l'option ‘mail-location’.

Il y a quelques variables spéciales que vous pouvez utiliser, p. ex :

‘%u’	nom d'utilisateur
‘%n’	la partie « utilisateur » dans « utilisateur@domaine », comme %u s'il n'y a pas de domaine
‘%d’	la partie « domaine » dans « utilisateur@domaine », vide s'il n'y a pas de domaine
‘%h’	répertoire personnel

Voir doc/wiki/Variables.txt pour la liste complète. Quelques exemple :

```
‘maildir:~/Maildir’
‘mbox:~/mail:INBOX=/var/mail/%u’
‘mbox:/var/mail/%d/%1n/%n:INDEX=/var/indexes/%d/%1n/%’
```

La valeur par défaut est ‘“”’.

- string mail-uid** [paramètre de dovecot-configuration]  
Utilisateur et groupe système utilisé pour accéder aux courriels. Si vous utilisez multiple, userdb peut remplacer ces valeurs en renvoyant les champs uid et gid. Vous pouvez utiliser soit des nombres, soit des noms. <doc/wiki/UserIds.txt>. La valeur par défaut est `""`.
- string mail-gid** [paramètre de dovecot-configuration]  
La valeur par défaut est `""`.
- string mail-privileged-group** [paramètre de dovecot-configuration]  
Groupe à activer temporairement pour les opérations privilégiées. Actuellement cela est utilisé uniquement avec INBOX lors de sa création initiale et quand le verrouillage échoie. Typiquement, vous pouvez utiliser `"mail"` pour donner accès à `/var/mail`. La valeur par défaut est `""`.
- string mail-access-groups** [paramètre de dovecot-configuration]  
Donne l'accès à ces groupes supplémentaires aux processus de courriel. Ils sont typiquement utilisés pour mettre en place l'accès à des boîtes aux lettres partagées. Remarquez qu'il peut être dangereux d'utiliser cette option si l'utilisateur peut créer des liens symboliques (p. ex. si le groupe `'mail'` est utilisé ici, `ln -s /var/mail ~/mail/var` peut permettre à un utilisateur de supprimer les boîtes aux lettres des autres, ou `ln -s /secret/shared/box ~/mail/mybox` lui permettrait de la lire). La valeur par défaut est `""`.
- string mail-attribute-dict** [paramètre de dovecot-configuration]  
L'emplacement d'un dictionnaire utilisé pour stocker les IMAP METADATA définies par RFC 5464 (<https://tools.ietf.org/html/rfc5464>).  
Les commandes IMAP METADATA sont disponibles seulement si le champ `imap-metadata?` de la configuration du protocole « `imap` » vaut `#t`.  
La valeur par défaut est `""`.
- boolean mail-full-filesystem-access?** [paramètre de dovecot-configuration]  
Permet l'accès complet au système de fichiers pour les clients. Il n'y a pas de vérification d'accès autres que ce que le système d'exploitation fait avec les UID/GID. Cela fonctionne aussi bien avec maildir qu'avec mbox, ce qui vous permet de préfixer les noms des boîtes aux lettres avec p. ex. `/chemin/` ou `~utilisateur/`. La valeur par défaut est `#f`.
- boolean mmap-disable?** [paramètre de dovecot-configuration]  
Ne pas du tout utiliser `mmap()`. Cela est requis si vous stockez les index dans des systèmes de fichiers partagés (NFS ou clusterfs). La valeur par défaut est `#f`.
- boolean dotlock-use-excl?** [paramètre de dovecot-configuration]  
S'appuyer sur `'O_EXCL'` lors de la création de fichiers de verrouillage. NFS supporte `'O_EXCL'` depuis la version 3, donc cette option est sûre de nos jours. La valeur par défaut est `#t`.

- string mail-fsync** [paramètre de dovecot-configuration]  
Quand utiliser les appels à `fsync()` ou `fdatsync()` :
- optimized** Lorsque cela est nécessaire pour éviter de perdre des données importantes
  - always** Utile lorsque par exemple les `write()` de NFS sont retardées
  - never** Ne l'utilisez pas (ça a de meilleures performances, mais les crashes font perdre toutes les données).
- La valeur par défaut est `"optimized"`.
- boolean mail-nfs-storage?** [paramètre de dovecot-configuration]  
Le stockage des courriels se fait sur NFS. Utilisez cette option pour que Dovecot vide les caches NFS lorsque c'est nécessaire. Si vous utilisez seulement un simple serveur de courriel, ce n'est pas nécessaire. La valeur par défaut est `#f`.
- boolean mail-nfs-index?** [paramètre de dovecot-configuration]  
Les fichiers d'index de courriels sont sur un système de fichiers NFS. Pour utiliser cette option, vous aurez besoin de `mmap-disable? #t` et `fsync-disable? #f`. La valeur par défaut est `#f`.
- string lock-method** [paramètre de dovecot-configuration]  
Méthode de verrouillage des fichiers d'index. Les alternatives sont `fcntl`, `flock` et `dotlock`. Le verrouillage-point (`dotlocking`) utilise des astuces qui peuvent créer plus d'utilisation du disque que les autres méthodes de verrouillage. Pour les utilisateurs de NFS, `flock` ne marche pas, et rappelez-vous de modifier `mmap-disable`. La valeur par défaut est `"fcntl"`.
- file-name mail-temp-dir** [paramètre de dovecot-configuration]  
Le répertoire dans lequel LDA/LMTP stockent temporairement les courriels de plus de 128 Ko. La valeur par défaut est `"/tmp"`.
- entier-non-négatif first-valid-uid** [paramètre de dovecot-configuration]  
L'intervalle d'UID valides pour les utilisateurs. Cette option est surtout utile pour s'assurer que les utilisateurs ne peuvent pas s'authentifier en tant que démon ou qu'un autre utilisateur système. Remarquez que la connexion en root est interdite en dur dans le binaire de dovecot et qu'on ne peut pas l'autoriser même si `first-valid-uid` vaut 0. La valeur par défaut est `'500'`.
- entier-non-négatif last-valid-uid** [paramètre de dovecot-configuration]  
La valeur par défaut est `'0'`.
- entier-non-négatif first-valid-gid** [paramètre de dovecot-configuration]  
L'intervalle de GID valides pour les utilisateurs. Les utilisateurs qui ont un GID non-valide comme numéro de groupe primaire ne peuvent pas se connecter. Si l'utilisateur appartient à un groupe avec un GID non valide, ce groupe n'est pas utilisable. La valeur par défaut est `'1'`.
- entier-non-négatif last-valid-gid** [paramètre de dovecot-configuration]  
La valeur par défaut est `'0'`.

**entier-non-négatif** [paramètre de dovecot-configuration]  
**mail-max-keyword-length**

Longueur maximale autorisée pour les mots-clefs. Elle n'est utilisée que lors de la création de nouveaux mots-clefs. La valeur par défaut est '50'.

**colon-separated-file-name-list** [paramètre de dovecot-configuration]  
**valid-chroot-dirs**

Liste des répertoires sous lesquels le chroot est permis pour les processus de traitement des courriels (c.-à-d. /var/mail permettra aussi de se chrooter dans /var/mail/toto/titi). Ce paramètre n'affecte pas 'login-chroot' 'mail-chroot' ou les paramètres de chroot de l'authentification. Si ce paramètre est vide, './' dans les répertoires personnels sont ignorés. ATTENTION : n'ajoutez jamais de répertoires ici que les utilisateurs locaux peuvent modifier, puisque ça pourrait permettre d'escalader les privilèges. Normalement vous ne devriez le faire que si les utilisateurs n'ont pas d'accès shell. <doc/wiki/Chrooting.txt>. La valeur par défaut est '()'.

**string mail-chroot** [paramètre de dovecot-configuration]

Répertoire chroot par défaut pour les processus de traitement des courriels. Cela peut être modifié pour des utilisateurs particuliers dans la base de donnée en donnant './' dans le répertoire personnel (p. ex. '/home/./utilisateur' permet de se chrooter dans /home). Remarquez qu'il n'y a d'habitude pas besoin de se chrooter. Dovecot ne permet pas aux utilisateurs d'accéder aux fichiers en dehors de leur répertoire de courriels de toute façon. Si vos répertoires personnels sont préfixés par le répertoire de chroot, ajoutez './' à 'mail-chroot'. <doc/wiki/Chrooting.txt>. La valeur par défaut est "".

**file-name auth-socket-path** [paramètre de dovecot-configuration]

Chemin de socket UNIX vers le serveur d'authentification maître pour trouver les utilisateurs. C'est utilisé par imap (pour les utilisateurs partagés) et lda. La valeur par défaut est "/var/run/dovecot/auth-userdb".

**file-name mail-plugin-dir** [paramètre de dovecot-configuration]

Répertoire où trouver les greffons. La valeur par défaut est "/usr/lib/dovecot".

**space-separated-string-list** [paramètre de dovecot-configuration]  
**mail-plugins**

Liste des greffons à charger pour tous les services. Les greffons spécifiques à IMAP, LDA, etc. sont ajoutés à cette liste dans leur propre fichiers .conf. La valeur par défaut est '()'.

**entier-non-négatif** [paramètre de dovecot-configuration]

**mail-cache-min-mail-count**

Le nombre minimal de courriels dans une boîte aux lettres avant de mettre à jour le fichier de cache. Cela permet d'optimiser le comportement de Dovecot pour qu'il fasse moins d'écriture disque contre plus de lecture disque. La valeur par défaut est '0'.

- string mailbox-idle-check-interval** [paramètre de `dovecot-configuration`]  
 Lorsque la commande IDLE est lancée, la boîte aux lettres est vérifiée de temps en temps pour voir s'il y a de nouveaux messages ou d'autres changements. Ce paramètre définit le temps d'attente minimum entre deux vérifications. Dovecot peut aussi utiliser `dnotify`, `inotify` et `kqueue` pour trouver immédiatement les changements. La valeur par défaut est `"30 secs"`.
- boolean mail-save-crlf?** [paramètre de `dovecot-configuration`]  
 Sauvegarder les courriels avec CR+LF plutôt que seulement LF. Cela permet de consommer moins de CPU en envoyant ces courriels, surtout avec l'appel système `sendfile()` de Linux et FreeBSD. Mais cela crée un peu plus d'utilisation du disque, ce qui peut aussi le ralentir. Remarquez aussi que si d'autres logiciels lisent les `mbox/mailldirs`, ils peuvent se tromper dans leur traitement de ces CR supplémentaires et causer des problèmes. La valeur par défaut est `'#f'`.
- boolean maildir-stat-dirs?** [paramètre de `dovecot-configuration`]  
 Par défaut la commande LIST renvoie toutes les entrées du maildir qui commencent par un point. Activer cette option permet à Dovecot de renvoyer uniquement les entrées qui sont des répertoires. Cela se fait avec `stat()` sur chaque entrée, ce qui cause plus d'utilisation du disque. For systems setting struct `'dirent->d_type'` this check is free and it's done always regardless of this setting). La valeur par défaut est `'#f'`.
- boolean maildir-copy-with-hardlinks?** [paramètre de `dovecot-configuration`]  
 Lors de la copie d'un message, le faire avec des liens en dur si possible. Cela améliore un peu la performance et n'a que peu de chance d'avoir des effets secondaires.
- boolean maildir-very-dirty-syncs?** [paramètre de `dovecot-configuration`]  
 Suppose que Dovecot est le seul MUA qui accède à Maildir : scanne le répertoire cur/ seulement lorsque son mtime change de manière inattendue ou lorsqu'il ne peut pas trouver le courriel autrement. La valeur par défaut est `'#f'`.
- space-separated-string-list mailbox-read-locks** [paramètre de `dovecot-configuration`]  
 La méthode de verrouillage à utiliser pour verrouiller les boîtes aux lettres `mbox`. Il y en a quatre :
- dotlock** Crée un fichier `<mailbox>.lock`. C'est la solution la plus ancienne et la plus sûre pour NFS. Si vous voulez utiliser `/var/mail/`, les utilisateurs auront besoin de l'accès en écriture à ce répertoire.
  - dotlock-try** Comme pour `dotlock`, mais si elle échoue à cause d'un problème de permission ou parce qu'il n'y a pas assez d'espace disque, l'ignore.
  - fcntl** Utilisez cette méthode si possible. Elle fonctionne aussi avec NFS si vous utilisez `lockd`.
  - flock** Peut ne pas exister sur tous les systèmes. Ne fonctionne pas avec NFS.

**lockf** Peut ne pas exister sur tous les systèmes. Ne fonctionne pas avec NFS.

Vous pouvez utiliser plusieurs méthodes de verrouillage ; dans ce cas l'ordre dans lequel elles sont déclarées est important pour éviter des interblocages si d'autres MTA/MUA utilisent aussi plusieurs méthodes. Certains systèmes d'exploitation ne permettent pas d'utiliser certaines méthodes en même temps.

**space-separated-string-list** [paramètre de dovecot-configuration]  
**mbox-write-locks**

**string mbox-lock-timeout** [paramètre de dovecot-configuration]  
 Temps d'attente maximal pour un verrou (tous les verrous) avant d'abandonner. La valeur par défaut est "5 mins".

**string mbox-dotlock-change-timeout** [paramètre de dovecot-configuration]  
 Si le fichier dotlock existe mais que la boîte aux lettres n'est pas modifiée, remplacer le fichier de verrouillage après ce temps d'attente. La valeur par défaut est "2 mins".

**boolean mbox-dirty-syncs?** [paramètre de dovecot-configuration]  
 Lorsqu'un mbox change de manière inattendue, il faut le lire en entier pour savoir ce qui a changé. Si le mbox est assez grand cela peut prendre beaucoup de temps. Comme le changement est habituellement un simple courriel supplémentaire, il serait plus rapide de lire les nouveaux courriels. Si ce paramètre est activé, Dovecot fait cela mais revient toujours à relire le fichier mbox complet si le fichier n'est pas comme attendu. Le seul réel inconvénient à ce paramètre est que certains MUA changent les drapeaux des messages, et dans ce cas Dovecot ne s'en rend pas immédiatement compte. Remarquez qu'une synchronisation complète est effectuée avec les commandes SELECT, EXAMINE, EXPUNGE et CHECK. La valeur par défaut est '#t'.

**boolean mbox-very-dirty-syncs?** [paramètre de dovecot-configuration]  
 Comme 'mbox-dirty-syncs', mais ne synchronise pas complètement même avec les commandes SELECT, EXAMINE, EXPUNGE ou CHECK. Si l'option n'est pas activée, 'mbox-dirty-syncs' est ignorée. La valeur par défaut est '#f'.

**boolean mbox-lazy-writes?** [paramètre de dovecot-configuration]  
 Attendre avant d'écrire les en-têtes mbox jusqu'à la prochaine synchronisation des écritures (les commandes EXPUNGE et CHECK et quand on ferme la boîte aux lettres). C'est surtout utile pour POP3 où les clients suppriment souvent tous les courriels. L'inconvénient c'est que vos changements ne sont pas immédiatement visibles pour les autres MUA. La valeur par défaut est '#t'.

**entier-non-négatif** [paramètre de dovecot-configuration]  
**mbox-min-index-size**  
 Si la taille du fichier mbox est plus petite que cela (p. ex. 100k), ne pas écrire de fichier d'index. Si un fichier d'index existe déjà il est toujours lu, mais pas mis à jour. La valeur par défaut est '0'.

**entier-non-négatif** [paramètre de dovecot-configuration]  
**mbox-rotate-size**  
 Taille du fichier mbox maximale avant rotation. La valeur par défaut est '1000000'.

**string mbox-rotate-interval** [paramètre de dovecot-configuration]  
 Âge maximum du fichier mbox avant rotation. Typiquement en jours. Les jours commencent à minuit, donc 1d signifie aujourd'hui, 2d pour hier, etc. 0 pour désactiver la vérification. La valeur par défaut est `"1d"`.

**boolean mbox-preallocate-space?** [paramètre de dovecot-configuration]  
 Lors de la création des fichiers mbox, préallouer immédiatement leur taille à `'mbox-rotate-size'`. Ce paramètre ne fonctionne actuellement que dans Linux avec certains systèmes de fichiers (ext4, xfs). La valeur par défaut est `'#f'`.

**string mail-attachment-dir** [paramètre de dovecot-configuration]  
 sdbx et mbox supportent la sauvegarde des pièces-jointes dans des fichiers externes, ce qui permet de les stocker une seule fois. Les autres moteurs ne le supportent pas pour le moment.

ATTENTION : Cette fonctionnalité n'a pas été beaucoup testée. Utilisez-la à vos risques et périls.

Racine du répertoire où stocker les pièces-jointes. Désactivé si vide. La valeur par défaut est `""`.

**entier-non-négatif** [paramètre de dovecot-configuration]  
**mail-attachment-min-size**  
 Les pièces-jointes plus petites que cela ne sont pas enregistrées à part. Il est aussi possible d'écrire un greffon pour désactiver l'enregistrement externe de certaines pièces-jointes spécifiques. La valeur par défaut est `'128000'`.

**string mail-attachment-fs** [paramètre de dovecot-configuration]  
 Moteur du système de fichier à utiliser pour sauvegarder les pièces-jointes :

**posix** Pas de SiS (single instance storage) par Dovecot (mais cela peut aider la déduplication du système de fichier)

**sis posix** SiS avec comparaison bit-à-bit immédiate pendant la sauvegarde

**sis-queue posix**  
 SiS avec déduplication et comparaison différées.

La valeur par défaut est `"sis posix"`.

**string mail-attachment-hash** [paramètre de dovecot-configuration]  
 Format de hash à utiliser dans les noms de fichiers des pièces-jointes. Vous pouvez ajouter n'importe quel texte ou variable : `%{md4}`, `%{md5}`, `%{sha1}`, `%{sha256}`, `%{sha512}`, `%{size}`. Les variables peuvent être tronquées, p. ex. `%{sha256:80}` renvoie seulement les 80 premiers bits. La valeur par défaut est `"%{sha1}"`.

**entier-non-négatif** [paramètre de dovecot-configuration]  
**default-process-limit**  
 La valeur par défaut est `'100'`.

**entier-non-négatif** [paramètre de dovecot-configuration]  
**default-client-limit**  
 La valeur par défaut est `'1000'`.



- entier-non-négatif** `default-vsyz-limit` [paramètre de `dovecot-configuration`]  
Limite VSZ (taille mémoire virtuelle) par défaut pour les processus de service. C'est surtout pour attraper et tuer les processus qui font fuiter la mémoire avant qu'ils ne l'utilisent en entier. La valeur par défaut est '256000000'.
- string** `default-login-user` [paramètre de `dovecot-configuration`]  
Utilisateur de connexion utilisé en interne par les processus de connexion. C'est l'utilisateur avec la confiance minimale pour Dovecot. Il ne devrait avoir accès à rien du tout. La valeur par défaut est "dovenull".
- string** `default-internal-user` [paramètre de `dovecot-configuration`]  
Utilisateur utilisé en interne par les processus non privilégiés. Il devrait être différent de l'utilisateur de connexion, pour que les processus de connexion ne puissent pas perturber les autres processus. La valeur par défaut est "dovecot".
- string** `ssl?` [paramètre de `dovecot-configuration`]  
Support SSL/TLS : yes, no, required. <doc/wiki/SSL.txt>. La valeur par défaut est "required".
- string** `ssl-cert` [paramètre de `dovecot-configuration`]  
Certificat SSL/TLS X.509 encodé en PEM (clef publique). La valeur par défaut est "</etc/dovecot/default.pem".
- string** `ssl-key` [paramètre de `dovecot-configuration`]  
Clef privée SSL/TLS encodée en PEM. La clef est ouverte avant l'abandon des privilèges root, donc laissez-la non-lisible pour les utilisateurs. La valeur par défaut est "</etc/dovecot/private/default.pem".
- string** `ssl-key-password` [paramètre de `dovecot-configuration`]  
Si le fichier de clef est protégé par un mot de passe, donnez-le ici. Autrement, donnez-le en démarrant dovecot avec le paramètre -p. Comme ce fichier est souvent lisible pour tout le monde, vous pourriez vouloir placer ce paramètre dans un autre fichier. La valeur par défaut est "".
- string** `ssl-ca` [paramètre de `dovecot-configuration`]  
Certificat de l'autorité de confiance encodé en PEM. Indiquez cette valeur si vous voulez utiliser 'ssl-verify-client-cert? #t'. Le fichier devrait contenir les certificats de CA suivi par les CRL correspondants (p. ex. 'ssl-ca </etc/ssl/certs/ca.pem'). La valeur par défaut est "".
- boolean** `ssl-require-crl?` [paramètre de `dovecot-configuration`]  
Indique si les certificats clients doivent réussir la vérification du CRL. La valeur par défaut est '#t'.
- boolean** `ssl-verify-client-cert?` [paramètre de `dovecot-configuration`]  
Demande aux clients d'envoyer un certificat. Si vous voulez aussi le requérir, indiquez 'auth-ssl-require-client-cert? #t' dans la section auth. La valeur par défaut est '#f'.

**string ssl-cert-username-field** [paramètre de dovecot-configuration]  
Le champ du certificat à utiliser pour le nom d'utilisateur. Les choix habituels sont `commonName` et `X500UniqueIdentifier`. Vous devrez aussi indiquer `'auth-ssl-username-from-cert? #t'`. La valeur par défaut est `"commonName"`.

**string ssl-min-protocol** [paramètre de dovecot-configuration]  
Version minimale de SSL à accepter. La valeur par défaut est `"TLSv1"`.

**string ssl-cipher-list** [paramètre de dovecot-configuration]  
Méthodes de chiffrement à utiliser. La valeur par défaut est `"ALL:!kRSA:!SRP:!kDHd:!DSS:!aNULL:!eN`

**string ssl-crypto-device** [paramètre de dovecot-configuration]  
Moteur cryptographique SSL à utiliser. Pour les valeur valides, lancez « `openssl engine` ». La valeur par défaut est `""`.

**string postmaster-address** [paramètre de dovecot-configuration]  
Adresse à utiliser pour envoyer les courriels de rejet. `%d` correspond au domaine du destinataire. La valeur par défaut est `"postmaster%d"`.

**string hostname** [paramètre de dovecot-configuration]  
Nom d'hôte à utiliser dans diverses parties des courriels envoyés (p. ex. dans `Message-Id`) et dans les réponses LMTP. La valeur par défaut est le nom d'hôte@domaine réel du système. La valeur par défaut est `""`.

**boolean quota-full-tempfail?** [paramètre de dovecot-configuration]  
Si l'utilisateur dépasse le quota, renvoie un échec temporaire au lieu de rejeter le courriel. La valeur par défaut est `#f`.

**file-name sendmail-path** [paramètre de dovecot-configuration]  
Binaire à utiliser pour envoyer des courriels. La valeur par défaut est `"/usr/sbin/sendmail"`.

**string submission-host** [paramètre de dovecot-configuration]  
Si la valeur est non vide, envoyer les courriels à ce serveur SMTP hôte[:port] au lieu de `sendmail`. La valeur par défaut est `""`.

**string rejection-subject** [paramètre de dovecot-configuration]  
En-tête d'objet à utiliser pour les courriels de rejet. Vous pouvez utiliser les mêmes variables que pour `'rejection-reason'` ci-dessous. La valeur par défaut est `"Rejected: %s"`.

**string rejection-reason** [paramètre de dovecot-configuration]  
Message d'erreur pour les humains dans les courriels de rejet. Vous pouvez utiliser ces variables :

<code>%n</code>	CRLF
<code>%r</code>	raison
<code>%s</code>	objet du courriel de départ
<code>%t</code>	destinataire

La valeur par défaut est `"Your message to <%t> was automatically rejected:%n%r"`.

- string recipient-delimiter** [paramètre de dovecot-configuration]  
Caractère de délimitation entre la partie locale et le détail des adresses de courriel. La valeur par défaut est "+".
- string lda-original-recipient-header** [paramètre de dovecot-configuration]  
En-tête où l'adresse du destinataire d'origine (l'adresse RCPT TO de SMTP) est récupérée si elle n'est pas disponible ailleurs. Le paramètre -a de dovecot-lda le remplace. L'en-tête couramment utilisée pour cela est X-Original-To. La valeur par défaut est "".
- boolean lda-mailbox-autocreate?** [paramètre de dovecot-configuration]  
Sauvegarder un courriel dans un fichier qui n'existe pas devrait-il le créer ? La valeur par défaut est '#f'.
- boolean lda-mailbox-autosubscribe?** [paramètre de dovecot-configuration]  
Devrait-on aussi se inscrire aux boîtes aux lettres nouvellement créées ? La valeur par défaut est '#f'.
- entier-non-négatif imap-max-line-length** [paramètre de dovecot-configuration]  
Longueur maximale de la ligne de commande IMAP. Certains clients génèrent des lignes de commandes très longues avec des boîtes aux lettres énormes, donc vous pourriez avoir besoin d'augmenter cette limite si vous obtenez les erreurs « Too long argument » ou « IMAP command line too large ». La valeur par défaut est '64000'.
- string imap-logout-format** [paramètre de dovecot-configuration]  
Format de la chaîne de déconnexion IMAP :
- %i        nombre d'octets lus par le client  
%o        nombre total d'octets envoyés au client.
- Voir `doc/wiki/Variables.txt` pour une liste de toutes les variables utilisables. La valeur par défaut est "in=%i out=%o deleted=%{deleted} expunged=%{expunged} trashed=%{trashed} hdr\_count=%{fetch\_hdr\_count} hdr\_bytes=%{fetch\_hdr\_bytes} body\_count=%{fetch\_body\_count} body\_bytes=%{fetch\_body\_bytes}".
- string imap-capability** [paramètre de dovecot-configuration]  
Remplace la réponse CAPABILITY d'IMAP. Si la valeur commence par « + », ajoute les capacités données en haut des valeur par défaut (p. ex. +XFOO XBAR). La valeur par défaut est "".
- string imap-idle-notify-interval** [paramètre de dovecot-configuration]  
Temps d'attente entre les notifications « OK Still here » lorsque le client est en IDLE. La valeur par défaut est "2 mins".
- string imap-id-send** [paramètre de dovecot-configuration]  
Noms des champs ID et de leur valeur à envoyer aux clients. « \* » signifie la valeur par défaut. Les champs suivants ont actuellement des valeurs par défaut : name, version, os, os-version, support-url, support-email. La valeur par défaut est "".

**string imap-id-log** [paramètre de `dovecot-configuration`]  
Champs ID envoyés par le client à enregistrer. « \* » signifie tout. La valeur par défaut est « '' ».

**space-separated-string-list** [paramètre de `dovecot-configuration`]  
**imap-client-workarounds**  
Contournements pour divers bogues de certains client :

**delay-newmail**

Envoi des notifications de nouveau message EXISTS/RECENT seulement en réponse aux commandes NOOP et CHECK. Certains clients les ignorent autrement, par exemple OSX Mail (< v2.1). Outlook Express est encore plus cassé, sans cela il peut montrer des erreurs de type « Le message n'est plus sur le serveur ». Remarquez que OE6 est toujours cassé même avec ce contournement si la synchronisation est à « En-têtes seulement ».

**tb-extra-mailbox-sep**

Thunderbird se mélange les pincesaux avec LAYOUT=fs (mbox et mbox) et ajoute un suffixe '/' supplémentaire sur les noms des boîtes aux lettres. Cette option fait que dovecot ignore le '/' supplémentaire au lieu de le traiter comme un nom de boîte aux lettres invalide.

**tb-lsub-flags**

Montre les drapeaux \Noselect pour les réponses LSUB avec LAYOUT=fs (p. ex. mbox). Cela fait que Thunderbird réalise qu'ils ne sont pas sélectionnables et les montre en grisé, au lieu de montrer un popup « non sélectionnable » après coup.

La valeur par défaut est '()'.

**string imap-urlauth-host** [paramètre de `dovecot-configuration`]  
Hôte autorisé dans les URL URLAUTH envoyés par les clients. « \* » les autorise tous. La valeur par défaut est « '' ».

Ouf ! Tant d'options de configuration. La bonne nouvelle, c'est que Guix a une interface complète avec le langage de configuration de Dovecot. Cela permet non seulement de déclarer la configuration de manière agréable, mais aussi d'offrir des capacités de réflexion : les utilisateurs peuvent écrire du code pour inspecter et transformer les configuration depuis Scheme.

Cependant, vous pourriez avoir un fichier `dovecot.conf` déjà tout prêt. Dans ce cas, vous pouvez passer un objet `opaque-dovecot-configuration` comme paramètre `#:config` à `dovecot-service`. Comme son nom l'indique, une configuration opaque n'a pas les capacités de réflexions.

Les champs de `opaque-dovecot-configuration` disponibles sont :

**package dovecot** [paramètre de `opaque-dovecot-configuration`]  
Le paquet dovecot.

**string string** [paramètre de `opaque-dovecot-configuration`]  
Le contenu de `dovecot.conf`, en tant que chaîne de caractères.

Par exemple, si votre `dovecot.conf` est simplement la chaîne vide, vous pouvez instancier un service `dovecot` comme cela :

```
(dovecot-service #:config
  (opaque-dovecot-configuration
   (string "")))
```

## Service OpenSMTPD

`opensmtpd-service-type` [Variable Scheme]

C'est le type de service de OpenSMTPD (<https://www.opensmtpd.org>), dont la valeur devrait être un objet `opensmtpd-configuration` comme dans cet exemple :

```
(service opensmtpd-service-type
  (opensmtpd-configuration
   (config-file (local-file "./my-smtpd.conf"))))
```

`opensmtpd-configuration` [Type de données]

Type de données représentant la configuration de `opensmtpd`.

`package` (par défaut : `opensmtpd`)

Objet de paquet du serveur SMTP OpenSMTPD.

`config-file` (par défaut : `%default-opensmtpd-config-file`)

Objet simili-fichier du fichier de configuration de OpenSMTPD à utiliser. Par défaut il écoute sur l'interface de boucle locale et accepte les courriels des utilisateurs et des démons de la machine locale, et autorise l'envoi de courriels à des serveurs distants. Lancez `man smtpd.conf` pour plus d'information.

`setgid-commands?` (par défaut : `#t`)

Rend les commandes suivantes `setgid` à `smtpq` pour qu'elles puissent être exécutées : `smtpctl`, `sendmail`, `send-mail`, `makemap`, `mailq` et `newalsiases`. Voir Section 12.10 [Programmes `setuid`], page 571, pour plus d'informations sur les programmes `setgid`.

## Service Exim

`exim-service-type` [Variable Scheme]

C'est le type de l'agent de transfert de courriel (MTA) Exim (<https://exim.org>), dont la valeur devrait être un objet `exim-configuration` comme dans cet exemple :

```
(service exim-service-type
  (exim-configuration
   (config-file (local-file "./my-exim.conf"))))
```

Pour utiliser le service `exim-service-type` vous devez aussi avoir un service `mail-aliases-service-type` dans votre `operating-system` (même sans alias).

`exim-configuration` [Type de données]

Type de données représentant la configuration d'Exim.

`package` (par défaut : `exim`)

Objet de paquet du serveur Exim.

`config-file` (par défaut : `#f`)

Objet simili-fichier du fichier de configuration d'Exim à utiliser. Si sa valeur est `#f` alors le service utilisera la configuration par défaut du paquet fourni dans `package`. Le fichier de configuration qui en résulte est chargé après avoir mis en place les variables de configuration `exim_user` et `exim_group`.

## Service Getmail

`getmail-service-type` [Variable Scheme]

C'est le type du service de récupération de courriels Getmail (<http://pyropus.ca/software/getmail/>), dont la valeur devrait être un objet `getmail-configuration`.

Les champs de `getmail-configuration` disponibles sont :

`symbol name` [paramètre de `getmail-configuration`]

Un symbole pour identifier le service getmail.

La valeur par défaut est `"unset"`.

`package package` [paramètre de `getmail-configuration`]

Le paquet getmail à utiliser.

`string user` [paramètre de `getmail-configuration`]

L'utilisateur qui lance getmail.

La valeur par défaut est `"getmail"`.

`string group` [paramètre de `getmail-configuration`]

Le groupe qui lance getmail.

La valeur par défaut est `"getmail"`.

`string directory` [paramètre de `getmail-configuration`]

Le répertoire getmail à utiliser.

La valeur par défaut est `"/var/lib/getmail/default"`.

`getmail-configuration-file rcfile` [paramètre de `getmail-configuration`]

Le fichier de configuration de getmail à utiliser.

Les champs de `getmail-configuration-file` disponibles sont :

`getmail-retriever-configuration` [paramètre de `getmail-configuration-file`]  
`retriever`

Le compte de courriel duquel récupérer les courriel, et comme accéder à ce compte.

Les champs de `getmail-retriever-configuration` disponibles sont :

`string type` [paramètre de `getmail-retriever-configuration`]

Le type de récupérateur de courriel à utiliser. Des valeurs valides sont `'passwd'` et `'static'`.

La valeur par défaut est `"SimpleIMAPSSLRetriever"`.

**string server** [paramètre de `getmail-retriever-configuration`]  
 Utilisateur avec lequel se connecter au serveur de courriel.  
 La valeur par défaut est `'unset'`.

**string username** [paramètre de `getmail-retriever-configuration`]  
 Utilisateur avec lequel se connecter au serveur de courriel.  
 La valeur par défaut est `'unset'`.

**entier-non-négatif port** [paramètre de `getmail-retriever-configuration`]  
 Numéro de port sur lequel se connecter.  
 La valeur par défaut est `'#f'`.

**string password** [paramètre de `getmail-retriever-configuration`]  
 Remplace des champs de passwd.  
 La valeur par défaut est `''`.

**list password-command** [paramètre de `getmail-retriever-configuration`]  
 Remplace des champs de passwd.  
 La valeur par défaut est `(')`.

**string keyfile** [paramètre de `getmail-retriever-configuration`]  
 Fichier de clés au format PEM pour la négociation TLS.  
 La valeur par défaut est `''`.

**string certfile** [paramètre de `getmail-retriever-configuration`]  
 Fichier de certificat au format PEM à utiliser pour la négociation TLS.  
 La valeur par défaut est `''`.

**string ca-certs** [paramètre de `getmail-retriever-configuration`]  
 Certificats d'autorité à utiliser.  
 La valeur par défaut est `''`.

**parameter-alist extra-parameters** [paramètre de `getmail-retriever-configuration`]  
 Paramètres supplémentaires pour le récupérateur.  
 La valeur par défaut est `(')`.

**getmail-destination-configuration** [paramètre de `getmail-configuration-file`]  
**destination**  
 Que faire avec les message récupérés.  
 Les champs de `getmail-destination-configuration` disponibles sont :

**string type** [paramètre de `getmail-destination-configuration`]  
 Le type de destination des courriels. Des valeurs valides sont `'Maildir'`,  
`'Mboxrd'` et `'MDA external'`.  
 La valeur par défaut est `'unset'`.

**string-or-filelike** [paramètre de `getmail-destination-configuration`]  
**path**

L'option de chemin pour la destination du courriel. Le comportement dépend du type choisi.

La valeur par défaut est `''`.

**parameter-alist** [paramètre de `getmail-destination-configuration`]  
**extra-parameters**

Paramètres supplémentaires pour la destination

La valeur par défaut est `()`.

**getmail-options-configuration** [paramètre de `getmail-destination-configuration`]  
**options**

Configurer getmail.

Les champs de `getmail-options-configuration` disponibles sont :

**non-negative-integer** [paramètre de `getmail-options-configuration`]  
**verbose**

Si la valeur est `'0'`, getmail n'affichera que les avertissements et les erreurs. Une valeur de `'1'` signifie que des messages à propos de la récupération et de la suppression de messages seront affichés. Si la valeur est `'2'`, getmail affichera des messages pour chacune de ses actions.

La valeur par défaut est `'1'`.

**boolean read-all** [paramètre de `getmail-options-configuration`]

Si la valeur est vraie, getmail récupérera tous les messages disponibles. Sinon, il récupérera les message qu'il n'a pas déjà vu.

La valeur par défaut est `'#t'`.

**boolean delete** [paramètre de `getmail-options-configuration`]

Si la valeur est vraie, les messages seront supprimées du serveur après les avoir récupérés et les avoir livrés correctement. Sinon, les messages seront laissés sur le serveur.

La valeur par défaut est `'#f'`.

**entier-non-négatif** [paramètre de `getmail-options-configuration`]  
**delete-after**

Getmail supprimera les messages après un certain nombre de jours après les avoir vu, s'ils ont été livrés. Cela signifie que les messages resteront sur le serveur quelques jours après qu'ils ont été livrés. Une valeur de `'0'` désactive cette fonctionnalité.

La valeur par défaut est `'0'`.

**entier-non-négatif** [paramètre de `getmail-options-configuration`]  
**delete-bigger-than**

Supprime les messages plus grands que cette quantité d'octets après les avoir récupérés, même si les options `delete` et `delete-after` sont désactivées. La valeur `'0'` permet de désactiver cette fonctionnalité.

La valeur par défaut est `'0'`.



**entier-non-négatif** [paramètre de `getmail-options-configuration`]  
**max-bytes-per-session**

Récupère les message jusqu'à ce nombre d'octets avant de fermer la session avec le serveur. La valeur '0' désactive cette fonctionnalité.

La valeur par défaut est '0'.

**entier-non-négatif** [paramètre de `getmail-options-configuration`]  
**max-message-size**

Ne récupère pas les messages plus grands que cette valeur en octets. La valeur '0' désactive cette fonctionnalité.

La valeur par défaut est '0'.

**boolean** [paramètre de `getmail-options-configuration`]  
**delivered-to**

Si la valeur est vraie, getmail ajoutera un en-tête Delivered-To aux messages.

La valeur par défaut est '#t'.

**boolean received** [paramètre de `getmail-options-configuration`]

Si la valeur est indiquée, getmail ajoute un en-tête Received aux messages.

La valeur par défaut est '#t'.

**string message-log** [paramètre de `getmail-options-configuration`]

Getmail enregistrera un journal de ses actions dans le fichier ainsi nommé. La valeur "" désactive la fonctionnalité.

La valeur par défaut est "".

**boolean** [paramètre de `getmail-options-configuration`]  
**message-log-syslog**

Si la valeur est vraie, getmail enregistrera un journal de ses actions en utilisant le démon de journalisation du système.

La valeur par défaut est '#f'.

**boolean** [paramètre de `getmail-options-configuration`]  
**message-log-verbose**

Si la valeur est vraie, getmail enregistrera un journal de ses action à propos des messages non-récupérés et la raison pour laquelle ils n'ont pas été récupérés, en plus de lignes d'informations sur le début et la fin des opérations.

La valeur par défaut est '#f'.

**parameter-alist** [paramètre de `getmail-options-configuration`]  
**extra-parameters**

Options supplémentaires à inclure.

La valeur par défaut est '()'.

**list idle** [paramètre de `getmail-options-configuration`]  
 Une liste de boîtes de courriels pour lesquels `getmail` devrait attendre une notification de nouveaux messages sur le serveur. Cela demande que le serveur prenne en charge l'extension IDLE.  
 La valeur par défaut est `()`.

**list environment-variables** [paramètre de `getmail-configuration`]  
 Une liste de variables d'environnement à définir pour `getmail`.  
 La valeur par défaut est `()`.

## Service d'alias de courriel

**mail-aliases-service-type** [Variable Scheme]  
 C'est le type de service qui fournit `/etc/aliases` et qui spécifie comment délivrer les courriels aux utilisateurs du système.

```
(service mail-aliases-service-type
  (('("postmaster" "bob")
    ("bob" "bob@example.com" "bob@example2.com"))))
```

La configuration pour un service `mail-aliases-service-type` est une liste associative qui dénote comment délivrer les courriels qui arrivent au système. Chaque entrée est de la forme `(alias adresses ...)` avec `alias` qui spécifie l'alias local et `adresses` qui spécifie où délivrer les courriels de cet utilisateur.

Les alias n'ont pas besoin de correspondre à des utilisateurs locaux du système. Dans l'exemple au-dessus, il n'y a pas besoin d'une entrée `postmaster` dans la liste `user-accounts` du `operating-system` pour délivrer les courriels à destination de `postmaster` à `bob` (qui ensuite délivrerait le courriel à `bob@example.com` et `bob@example2.com`).

## Démon IMAP4 GNU Mailutils

**imap4d-service-type** [Variable Scheme]  
 C'est le type du démon IMAP4 GNU Mailutils, dont la valeur devrait être un objet `imap4d-configuration` comme dans cet exemple :

```
(service imap4d-service-type
  (imap4d-configuration
    (config-file (local-file "imap4d.conf"))))
```

**imap4d-configuration** [Type de données]  
 Type de données représentant la configuration de `imap4d`.

**package** (par défaut : `mailutils`)  
 Le paquet qui fournit `imap4d`.

**config-file** (par défaut : `%default-imap4d-configuration`)  
 Objet simili-fichier du fichier de configuration à utiliser. Par défaut, la configuration fera écouter sur le port TCP 143 sur `localhost`. Voir Section "Conf-`imap4d`" dans *GNU Mailutils Manual*, pour les détails.

## Service Radicale

**radicale-service-type** [Variable Scheme]

C'est le type du service de récupération de courriels Radicale (<https://radicale.org>), dont la valeur devrait être un objet `radicale-configuration`.

**radicale-configuration** [Type de données]

Type de données représentant la configuration de `radicale`.

`package` (par défaut : `radicale`)

Le paquet qui fournit `radicale`.

`config-file` (par défaut : `%default-radicale-config-file`)

Objet simili-fichier du fichier de configuration à utiliser. Par défaut, la configuration fera écouter sur le port TCP 5232 sur `localhost` et utiliser le fichier `htpasswd` dans `/var/lib/radicale/users` sans chiffrement (`plain`).

### 12.9.13 Services de messagerie

Le module (`gnu services messaging`) fournit des définitions de services Guix pour les services de messageries instantanées. Actuellement il fournit les services suivants :

#### Service Prosody

**prosody-service-type** [Variable Scheme]

C'est le type pour le le serveur de communication XMPP Prosody (<https://prosody.im>). Sa valeur doit être un enregistrement `prosody-configuration` comme dans cet exemple :

```
(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"))))))
```

Voir plus bas pour des détails sur `prosody-configuration`.

Par défaut, Prosody n'a pas besoin de beaucoup de configuration. Seul un champ `virtualhosts` est requis : il spécifie le domaine que vous voulez voir Prosody servir.

Vous pouvez effectuer plusieurs vérifications de la configuration générée avec la commande `prosodyctl check`.

Prosodyctl vous aidera aussi à importer des certificats du répertoire `letsencrypt` pour que l'utilisateur `prosody` puisse y accéder. Voir <https://prosody.im/doc/letsencrypt>.

```
prosodyctl --root cert import /etc/letsencrypt/live
```

Les paramètres de configuration disponibles sont les suivants. Chaque définition des paramètres est précédé par son type ; par exemple, `'liste-de-chaines toto'` indique que le paramètre `toto` devrait être spécifié comme une liste de chaînes de caractères. Les types précédés de `peut-être-` signifient que le paramètre n'apparaîtra pas dans `prosody.cfg.lua` lorsque sa valeur est non spécifiée.

Il y a aussi une manière de spécifier la configuration en tant que chaîne de caractères si vous avez un vieux fichier `prosody.cfg.lua` que vous voulez porter depuis un autre système ; voir la fin pour plus de détails.

Le type `file-object` désigne soit un objet simili-fichier (voir Section 9.12 [G-Expressions], page 166), soit un nom de fichier.

Les champs de `prosody-configuration` disponibles sont :

`package prosody` [paramètre de `prosody-configuration`]  
Le paquet Prosody.

`file-name data-path` [paramètre de `prosody-configuration`]  
Emplacement du répertoire de stockage des données de Prosody. Voir <https://prosody.im/doc/configure>. La valeur par défaut est `"/var/lib/prosody"`.

`file-object-list plugin-paths` [paramètre de `prosody-configuration`]  
Répertoires de greffons supplémentaires. Ils sont analysés dans l'ordre spécifié. Voir [https://prosody.im/doc/plugins\\_directory](https://prosody.im/doc/plugins_directory). La valeur par défaut est `('')`.

`file-name certificates` [paramètre de `prosody-configuration`]  
Chaque hôte virtuel et composant a besoin d'un certificat pour que les clients et les serveurs puissent vérifier son identité. Prosody chargera automatiquement les clefs et les certificats dans le répertoire spécifié ici. La valeur par défaut est `"/etc/prosody/certs"`.

`string-list admins` [paramètre de `prosody-configuration`]  
C'est une liste des comptes administrateurs de ce serveur. Remarquez que vous devez créer les comptes séparément. Voir <https://prosody.im/doc/admins> et [https://prosody.im/doc/creating\\_accounts](https://prosody.im/doc/creating_accounts). Par exemple : `(admins ('("user1@example.com" "user2@example.net")))`. La valeur par défaut est `('')`.

`boolean use-libevent?` [paramètre de `prosody-configuration`]  
Active l'utilisation de `libevent` pour de meilleures performances sous une forte charge. Voir <https://prosody.im/doc/libevent>. La valeur par défaut est `'#f'`.

`module-list modules-enabled` [paramètre de `prosody-configuration`]  
C'est la liste des modules que Prosody chargera au démarrage. Il cherchera `mod_modulename.lua` dans le répertoire des greffons, donc assurez-vous qu'il existe aussi. La documentation des modules se trouve sur <https://prosody.im/doc/modules>. La valeur par défaut est `('("roster" "saslauth" "tls" "dialback" "disco" "carbons" "private" "blocklist" "vcard" "version" "uptime" "time" "ping" "pep" "register" "admin_adhoc")')`.

**string-list modules-disabled** [paramètre de `prosody-configuration`]  
 "offline", "c2s" et "s2s" sont chargés automatiquement, mais si vous voulez les désactiver, ajoutez-les à cette liste. La valeur par défaut est '()'.

**file-object groups-file** [paramètre de `prosody-configuration`]  
 Chemin vers un fichier texte où les groupes partagés sont définis. Si ce chemin est vide alors 'mod\_groups' ne fait rien. Voir [https://prosody.im/doc/modules/mod\\_groups](https://prosody.im/doc/modules/mod_groups). La valeur par défaut est `"/var/lib/prosody/sharedgroups.txt"`.

**boolean allow-registration?** [paramètre de `prosody-configuration`]  
 Désactive la création de compte par défaut, pour la sécurité. Voir [https://prosody.im/doc/creating\\_accounts](https://prosody.im/doc/creating_accounts). La valeur par défaut est '#f'.

**peut-être-ssl-configuration ssl** [paramètre de `prosody-configuration`]  
 Ce sont les paramètres liés à SSL/TLS. La plupart sont désactivés pour pouvoir utiliser les paramètres par défaut de Prosody. Si vous ne comprenez pas complètement ces options, ne les ajoutez pas à votre configuration, il est aisé de diminuer la sécurité de votre serveur en les modifiant. Voir [https://prosody.im/doc/advanced\\_ssl\\_config](https://prosody.im/doc/advanced_ssl_config).

Les champs de `ssl-configuration` disponibles sont :

**peut-être-chaine protocol** [paramètre de `ssl-configuration`]  
 Cela détermine la poignée de main à utiliser.

**peut-être-nom-de-fichier key** [paramètre de `ssl-configuration`]  
 Chemin vers votre fichier de clef privée.

**peut-être-nom-de-fichier certificate** [paramètre de `ssl-configuration`]  
 Chemin vers votre fichier de certificat.

**file-object capath** [paramètre de `ssl-configuration`]  
 Chemin vers le répertoire contenant les certificats racines que vous voulez voir Prosody utiliser lors de la vérification des certificats des serveurs distants. La valeur par défaut est `"/etc/ssl/certs"`.

**peut-être-fichier-objet cafile** [paramètre de `ssl-configuration`]  
 Chemin vers un fichier contenant les certificats racines auxquels Prosody devra faire confiance. Comme `capath` mais avec les certificats concaténés ensemble.

**peut-être-liste-de-chaine verify** [paramètre de `ssl-configuration`]  
 Une liste d'options de vérification (qui correspondent globalement aux drapeaux `set_verify()` d'OpenSSL).

**peut-être-liste-de-chaine options** [paramètre de `ssl-configuration`]  
 Une liste d'options générales liées à SSL/TLS. Elles correspondent globalement à `set_options()` d'OpenSSL. Pour une liste complète des options disponibles dans LuaSec, voir les sources de LuaSec.

**peut-être-entier-non-négatif depth** [paramètre de `ssl-configuration`]  
 Longueur maximale d'une chaîne d'autorités de certifications avant la racine.

- peut-être-chaine ciphers** [paramètre de `ssl-configuration`]  
Une chaîne de méthodes de chiffrement OpenSSL. Cela choisi les méthodes de chiffrement que Prosody offrira aux clients, et dans quel ordre de préférence.
- peut-être-nom-de-fichier dhparam** [paramètre de `ssl-configuration`]  
Un chemin vers un fichier contenant les paramètres pour l'échange de clef Diffie-Hellman. Vous pouvez créer un tel fichier avec : `openssl dhparam -out /etc/prosody/certs/dh-2048.pem 2048`
- peut-être-chaine curve** [paramètre de `ssl-configuration`]  
Courbe pour Diffie-Hellman sur courbe elliptique. La valeur par défaut de Prosody est `"secp384r1"`.
- peut-être-liste-de-chaine verifyext** [paramètre de `ssl-configuration`]  
Une liste d'options de vérification « supplémentaires ».
- peut-être-chaine password** [paramètre de `ssl-configuration`]  
Mot de passe pour les clefs privées chiffrées.
- boolean c2s-require-encryption?** [paramètre de `prosody-configuration`]  
S'il faut forcer toutes les connexions client-serveur à être chiffrées ou non. Voir [https://prosody.im/doc/modules/mod\\_tls](https://prosody.im/doc/modules/mod_tls). La valeur par défaut est `'#f'`.
- string-list disable-sasl-mechanisms** [paramètre de `prosody-configuration`]  
Ensemble de mécanismes qui ne seront jamais offerts. Voir [https://prosody.im/doc/modules/mod\\_saslauth](https://prosody.im/doc/modules/mod_saslauth). La valeur par défaut est `('DIGEST-MD5')`.
- boolean s2s-require-encryption?** [paramètre de `prosody-configuration`]  
S'il faut forcer toutes les connexion serveur-serveur à être chiffrées ou non. Voir [https://prosody.im/doc/modules/mod\\_tls](https://prosody.im/doc/modules/mod_tls). La valeur par défaut est `'#f'`.
- boolean s2s-secure-auth?** [paramètre de `prosody-configuration`]  
S'il faut requérir le chiffrement et l'authentification du certificat. Cela fournit une sécurité idéale, mais demande que les serveurs avec lesquels vous communiquez supportent le chiffrement ET présentent un certificat valide et de confiance. Voir <https://prosody.im/doc/s2s#security>. La valeur par défaut est `'#f'`.
- string-list s2s-insecure-domains** [paramètre de `prosody-configuration`]  
Beaucoup de serveurs ne supportent pas le chiffrement ou ont un certificat invalide ou auto-signé. Vous pouvez lister les domaines ici qui n'ont pas besoin de s'authentifier avec des certificats. Ils seront authentifiés par DNS. Voir <https://prosody.im/doc/s2s#security>. La valeur par défaut est `('')`.
- string-list s2s-secure-domains** [paramètre de `prosody-configuration`]  
Même si vous laissez `s2s-secure-auth?` désactivé, vous pouvez toujours demander un certificat valide pour certains domaine en spécifiant la liste ici. Voir <https://prosody.im/doc/s2s#security>. La valeur par défaut est `('')`.

**string authentication** [paramètre de prosody-configuration]  
 Choisi le moteur d'authentification à utiliser. Le moteur par défaut stocke les mots de passes en texte clair et utilise la configuration de stockage des données de Prosody pour stocker les données authentifiées. Si vous n'avez pas confiance dans le serveur, lisez [https://prosody.im/doc/modules/mod\\_auth\\_internal\\_hashed](https://prosody.im/doc/modules/mod_auth_internal_hashed) pour plus d'information sur l'utilisation du moteur hashed. Voir aussi <https://prosody.im/doc/authentication>. La valeur par défaut est "internal\_plain".

**peut-être-chaine log** [paramètre de prosody-configuration]  
 Indique les options de journalisation. La configuration avancée des journaux n'est pas encore supportée par le service Prosody. Voir <https://prosody.im/doc/logging>. La valeur par défaut est "\*syslog".

**file-name pidfile** [paramètre de prosody-configuration]  
 Fichier où écrire le PID. Voir [https://prosody.im/doc/modules/mod\\_posix](https://prosody.im/doc/modules/mod_posix). La valeur par défaut est "/var/run/prosody/prosody.pid".

**peut-être-entier-non-négatif http-max-content-size** [paramètre de prosody-configuration]  
 Taille maximum autorisée pour le corps HTTP (en octets).

**peut-être-chaine http-external-url** [paramètre de prosody-configuration]  
 Certains modules exposent leur propre URL de diverses manières. Cette URL est construite à partir du protocole, de l'hôte et du port utilisé. Si Prosody se trouve derrière un proxy, l'URL publique sera `http-external-url` à la place. Voir [https://prosody.im/doc/http#external\\_url](https://prosody.im/doc/http#external_url).

**virtualhost-configuration-list virtualhosts** [paramètre de prosody-configuration]  
 Un hôte dans Prosody est un domaine sur lequel les comptes utilisateurs sont créés. Par exemple si vous voulez que vos utilisateurs aient une adresse comme "john.smith@example.com" vous devrez ajouter un hôte "example.com". Toutes les options de cette liste seront appliquées uniquement à cet hôte.

**Remarque:** Le nom d'hôte *virtuel* est utilisé dans la configuration pour éviter de le confondre avec le nom d'hôte physique réel de la machine qui héberge Prosody. Une seule instance de Prosody peut servir plusieurs domaines, chacun défini comme une entrée VirtualHost dans la configuration de Prosody. Ainsi, un serveur qui n'héberge qu'un seul domaine n'aura qu'une entrée VirtualHost.

Voir [https://prosody.im/doc/configure#virtual\\_host\\_settings](https://prosody.im/doc/configure#virtual_host_settings).

Les champs de `virtualhost-configuration` disponibles sont :

tous ces champs de `prosody-configuration` : `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`, plus :

**string domain** [paramètre de `virtualhost-configuration`]  
 Domaine que vous souhaitez que Prosody serve.

**int-component-configuration-list** [paramètre de `prosody-configuration`]  
**int-components**

Les composants sont des services supplémentaires qui sont disponibles pour les clients, habituellement sur un sous-domaine du serveur principal (comme `"mycomponent.example.com"`). Des exemples de composants sont des serveurs de chatroom, des répertoires utilisateurs ou des passerelles vers d'autres protocoles.

Les composants internes sont implémentés dans des greffons spécifiques à Prosody. Pour ajouter un composant interne, vous n'avez qu'à remplir le champ de nom d'hôte et le greffon que vous voulez utiliser pour le composant.

Voir <https://prosody.im/doc/components>. La valeur par défaut est `()`.

Les champs de `int-component-configuration` disponibles sont :

tous ces champs de `prosody-configuration` : `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`, plus :

**string hostname** [paramètre de `int-component-configuration`]  
 Nom d'hôte du composant.

**string plugin** [paramètre de `int-component-configuration`]  
 Greffon que vous voulez utiliser pour ce composant.

**peut-être-mod-muc-configuration** [paramètre de `int-component-configuration`]  
**mod-muc**

Le chat multi-utilisateur (MUC) est le module de Prosody qui vous permet de créer des chatrooms/conférences pour les utilisateurs XMPP.

Des informations générales sur la configuration des salons de discussion multi-utilisateurs se trouvent dans la documentation sur les salons (<https://prosody.im/doc/chatrooms>), que vous devriez lire si vous découvrez les salons XMPP.

Voir aussi [https://prosody.im/doc/modules/mod\\_muc](https://prosody.im/doc/modules/mod_muc).

Les champs de `mod-muc-configuration` disponibles sont :

**string name** [paramètre de `mod-muc-configuration`]  
 Le nom à renvoyer dans les réponses de découverte de services. La valeur par défaut est `"Prosody Chatrooms"`.

**string-or-boolean** [paramètre de `mod-muc-configuration`]  
**restrict-room-creation**

Si la valeur est `#t`, cela permettra uniquement aux admins de créer de nouveaux salons. Sinon n'importe qui peut créer un salon. La valeur `"local"` restreint la création aux utilisateurs du domaine parent du service. P. ex. `'user@example.com'` peut créer des salons sur `'rooms.example.com'`. La valeur `"admin"` restreint ce service aux administrateurs. La valeur par défaut est `#f`.



**entier-non-négatif** [paramètre de `mod-muc-configuration`]  
**max-history-messages**

Nombre maximum de messages d'historique qui seront envoyés aux membres qui viennent de rejoindre le salon. La valeur par défaut est '20'.

**ext-component-configuration-list** [paramètre de `prosody-configuration`]  
**ext-components**

Les composants externes utilisent XEP-0114, que la plupart des composants supportent. Pour ajouter un composant externe, vous remplissez simplement le champ de nom d'hôte. Voir <https://prosody.im/doc/components>. La valeur par défaut est '()'.

Les champs de `ext-component-configuration` disponibles sont :

tous ces champs de `prosody-configuration` : `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`, plus :

**string component-secret** [paramètre de `ext-component-configuration`]  
 Mot de passe que le composant utilisera pour s'authentifier.

**string hostname** [paramètre de `ext-component-configuration`]  
 Nom d'hôte du composant.

**liste-d'entiers-non-négatifs** [paramètre de `prosody-configuration`]  
**component-ports**

Ports sur lesquels Prosody écoutera les connexions des composants. La valeur par défaut est '(5347)'.

**string component-interface** [paramètre de `prosody-configuration`]  
 Interface sur laquelle Prosody écoutera les connexions des composants. La valeur par défaut est '"127.0.0.1"'.

**peut-être-contenu-brut raw-content** [paramètre de `prosody-configuration`]  
 Contenu brut qui sera ajouté au fichier de configuration.

Il se peut que vous ayez juste envie de lancer un fichier `prosody.cfg.lua` directement. Dans ce cas, vous pouvez passer un enregistrement `opaque-prosody-configuration` comme valeur à `prosody-service-type`. Comme son nom l'indique, une configuration opaque n'a pas de capacités de réflexion simples. Les champs disponibles de `opaque-prosody-configuration` sont :

**package prosody** [paramètre de `opaque-prosody-configuration`]  
 Le paquet prosody.

**string prosody.cfg.lua** [paramètre de `opaque-prosody-configuration`]  
 Le contenu de `prosody.cfg.lua` à utiliser.

Par exemple, si votre `prosody.cfg.lua` est juste la chaîne vide, vous pouvez instancier un service prosody comme ceci :

```
(service prosody-service-type
  (opaque-prosody-configuration
    (prosody.cfg.lua "")))
```

## Service BitlBee

BitlBee (<https://bitlbee.org>) est une passerelle qui fournit une interface IRC vers une variété de protocoles de messagerie instantanée comme XMPP.

**bitlbee-service-type** [Variable Scheme]

C'est le type de service pour le démon de passerelle IRC BitlBee (<https://bitlbee.org>). Sa valeur est un `bitlbee-configuration` (voir plus bas).

Pour que BitlBee écoute sur le port 6667 sur localhost, ajoutez cette ligne à vos services :

```
(service bitlbee-service-type)
```

**bitlbee-configuration** [Type de données]

C'est la configuration de BitlBee, avec les champs suivants :

**interface** (par défaut : "127.0.0.1")

**port** (par défaut : 6667)

Écoute sur l'interface réseau correspondant à l'adresse IP dans *interface*, sur *port*.

Lorsque *interface* vaut 127.0.0.1, seuls les clients locaux peuvent se connecter ; lorsqu'elle vaut 0.0.0.0, les connexions peuvent venir de n'importe quelle interface réseau.

**bitlbee** (par défaut : `bitlbee`)

Le paquet BitlBee à utiliser.

**plugins** (par défaut : '()')

Liste des paquets de greffons à utiliser — p. ex. `bitlbee-discord`.

**extra-settings** (par défaut : "")

Partie de configuration ajoutée telle-quelle au fichier de configuration de BitlBee.

## Service Quassel

Quassel (<https://quassel-irc.org/>) est un client IRC distribué, ce qui signifie qu'un client ou plus peuvent s'attacher et se détacher du cœur central.

**quassel-service-type** [Variable Scheme]

C'est le type de service pour le démon IRC Quassel (<https://quassel-irc.org/>).

Sa valeur est un `quassel-configuration` (voir plus bas).

**quassel-configuration** [Type de données]

C'est la configuration de Quassel, avec les champs suivants :

**quassel** (par défaut : `quassel`)

Le paquet Quassel à utiliser.

```
interface (par défaut : "::,0.0.0.0")
port (par défaut : 4242)
    Écoute sur les interfaces réseau correspondant à l'adresse IPv4 ou IPv6
    des interfaces spécifiées dans interface, une liste de chaînes délimitées par
    des virgules, sur port.
```

```
loglevel (par défaut : "info")
    Le niveau de journalisation souhaité. Les valeurs acceptées sont « Debug
    », « Info », « Warning » et « Error ».
```

### 12.9.14 Services de téléphonie

Le module (`gnu services telephony`) contient des définitions de services Guix pour les services de téléphonie. Actuellement il fournit les services suivants :

#### Jami

Cette section décrit comment configurer un serveur Jami que vous pouvez utiliser pour héberger des conférence vidéo (ou audio) entre autres choses. L'exemple suivant montre comment spécifier les archives de compte Jami (les sauvegardes) automatiquement :

```
(service jami-service-type
  (jami-configuration
    (accounts
      (list (jami-account
              (archive "/etc/jami/unencrypted-account-1.gz"))
            (jami-account
              (archive "/etc/jami/unencrypted-account-2.gz"))))))))
```

Lorsque le champ de comptes est spécifié, les fichiers de compte Jami du service trouvés dans `/var/lib/jami` seront recréés à chaque redémarrage du service.

Les comptes Jami et leurs sauvegardes correspondantes peuvent être générés avec les clients `jami` ou `jami-gnome`. Les comptes ne devraient pas être protégés par un mot de passe, mais il vaut mieux vous assurer que leurs fichiers ne sont lisibles que par `'root'`.

L'exemple suivant montre comme déclarer que seuls certains contact devraient pouvoir communiquer avec un compte donné :

```
(service jami-service-type
  (jami-configuration
    (accounts
      (list (jami-account
              (archive "/etc/jami/unencrypted-account-1.gz")
              (peer-discovery? #t)
              (rendezvous-point? #t)
              (allowed-contacts
                ('("1dbcb0f5f37324228235564b79f2b9737e9a008f"
                  "2dbcb0f5f37324228235564b79f2b9737e9a008f"))))))))
```

dans ce mode, seuls les `allowed-contacts` déclarés peuvent débiter une communication avec le compte Jami. Vous pouvez utiliser cela par exemple avec des comptes de point de rendez-vous pour créer un espace de vidéo-conférence privé.

Pour que l'administrateur système ait le contrôle complet des conférences hébergées sur son système, le service Jami prend en charge les actions suivantes :

```
# 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)
```

Les actions ci-dessus ont pour but de fournir les actions les plus utiles pour la modération, pas de couvrir l'ensemble de l'API de Jami. Si vous souhaitez interagir avec le démon Jami depuis Guile, vous pouvez expérimenter le module (`gnu build jami-service`), qui propulse les actions Shepherd ci-dessus.

Les action `add-moderator` et `ban-contact` acceptent *l'empreinte* d'un contact (un hash long de 40 caractères) comme premier argument et l'empreinte d'un compte ou un nom d'utilisateur comme second argument :

```
# herd add-moderator jami 1dbcb0f5f37324228235564b79f2b9737e9a008f \
 f3345f2775ddfe07a4b0d95daea111d15fbc1199

# herd list-moderators jami
Moderators for account f3345f2775ddfe07a4b0d95daea111d15fbc1199:
- 1dbcb0f5f37324228235564b79f2b9737e9a008f
```

dans le cas de `ban-contact`, le second argument est facultatif ; lorsqu'il est omis, le compte est banni de tous les comptes de Jami :

```
# herd ban-contact jami 1dbcb0f5f37324228235564b79f2b9737e9a008f

# herd list-banned-contacts jami
Banned contacts for account f3345f2775ddfe07a4b0d95daea111d15fbc1199:
- 1dbcb0f5f37324228235564b79f2b9737e9a008f
```

Les contacts bannis n'ont plus non plus de privilèges administrateur.

L'action `disable-account` permet de déconnecter entièrement un compte du réseau, le rendant injoignable, tandis que `enable-account` fait l'inverse. Ils acceptent un seul nom de compte ou empreinte comme premier argument :

```
# herd disable-account jami f3345f2775ddfe07a4b0d95daea111d15fbc1199

# herd list-accounts jami
The following Jami accounts are available:
- f3345f2775ddfe07a4b0d95daea111d15fbc1199 (dummy) [disabled]
```

L'action `list-account-details` affiche les paramètres détaillés de chaque compte au format Recutils, ce qui signifie que vous pouvez utiliser la commande `recsel` pour choisir les comptes qui vous intéressent (voir Section “Selection Expressions” dans *GNU recutils manual*). Remarquez que les caractères point (‘.’) qui se trouvent la les clés des paramètres deviennent des underscores (‘\_’) dans la sortie, pour satisfaire les exigences du format Recutils. L'exemple suivant vous montre comment afficher les empreintes des comptes qui opèrent en mode rendez-vous :

```
# herd list-account-details jami | \
  recsel -p Account.username -e 'Account.rendezVous ~ "true"'
Account_username: f3345f2775ddfe07a4b0d95daea111d15fbc1199
```

Les autres actions devraient être évidentes.

L'ensemble des options de configuration disponibles est donné ci-dessous.

**jami-configuration** [Type de données]

Les champs de `jami-configuration` disponibles sont :

`libjami` (par défaut : `libjami`) (type : paquet)

Le paquet du démon Jami à utiliser.

`dbus` (par défaut : `dbus-for-jami`) (type : paquet)

Le paquet D-Bus à utiliser pour démarrer la session D-Bus requise.

`nss-certs` (par défaut : `nss-certs`) (type : paquet)

Le paquet `nss-certs` à utiliser pour utiliser les certificats TLS fournis.

`enable-logging?` (par défaut : `#t`) (type : booléen)

Indique s'il faut activer la journalisation vers `syslog`.

`debug?` (par défaut : `#f`) (type : booléen)

Indique s'il faut activer les messages de débogage.

`auto-answer?` (par défaut : `#f`) (type : booléen)

Indique s'il faut forcer la réponse automatique aux appels entrants.

`accounts` (type : peut-être-liste-de-jami-account)

Une liste de comptes Jami à (re-)provisionner à chaque fois que le service du démon Jami démarre. Lorsque le champ est spécifié, les répertoires des comptes dans `/var/lib/jami/` sont recréés à chaque fois que le service démarre, pour assurer un état cohérent.

**jami-account** [Type de données]

Les champs de `jami-account` disponibles sont :

`archive` (type : string-or-computed-file)

Le nom de fichier de l'archive (sauvegarde) du compte. C'est utilisé pour provisionner le compte quand le service démarre. L'archive du compte ne devrait *pas* être chiffrée. Il est fortement recommandé de la rendre lisible uniquement à l'utilisateur ‘`root`’ (c.-à-d. pas dans le dépôt), pour vous prémunir contre la fuite de la clé secrète du compte Jami sauvegardé.

`allowed-contacts` (type : peut-être-liste-de-account-fingerprint)

La liste des contacts permis pour le compte, saisis avec leur empreinte longue de 40 caractères. Les messages et les appels des comptes qui ne

sont pas dans la liste seront rejetés. Lorsque la valeur n'est pas spécifiée, la configuration de l'archive du compte est utilisée telle-quelle par rapport aux permissions d'appels et de messages entrants, ce qui se résume par défaut à accepter tous les contacts à communiquer avec le compte.

**moderators** (type : peut-être-liste-de-account-fingerprint)

La liste des contacts qui devraient avoir des privilèges de modération (pour bannir, rendre muet, etc les autres utilisateur-rices) dans les conférences en rendez-vous, saisis avec leur empreinte longue de 40 caractères. Lorsque la valeur n'est pas spécifiée, la configuration de l'archive du compte est utilisée telle-quelle pour la modération, ce qui se résume par défaut à permettre à n'importe qui de modérer.

**rendezvous-point?** (type : peut-être-booléen)

Indique si le compte devrait opérer en mode rendez-vous. Dans ce mode, tous les appels audio et vidéos entrants sont fusionnés en une conférence. Lorsque la valeur n'est pas spécifiée, la valeur de l'archive du compte est utilisée.

**peer-discovery?** (type : peut-être-booléen)

Indique si la découverte de pairs devrait être activée. La découverte de pairs est utilisée pour découvrir d'autres nœuds OpenDHT sur le réseau local, ce qui peut être utile pour maintenir une connexion entre appareils locaux sur le réseau même si la connexion internet est perdue. Lorsque la valeur n'est pas spécifiée, celle de l'archive du compte est utilisée.

**bootstrap-hostnames** (type : peut-être-liste-de-chaine)

Une liste des noms d'hôte ou IP pointant vers des nœuds OpenDHT, qui devraient être utilisés pour rejoindre le réseau OpenDHT au départ. Lorsque la valeur n'est pas spécifiée, la valeur de l'archive du compte est utilisée.

**name-server-uri** (type : peut-être-chaine)

L'URI du serveur de nom à utiliser, qui peut être utilisé pour récupérer l'empreinte du compte pour un nom d'utilisateur enregistré.

## Serveur Mumble

Cette section décrit comment configurer et lancer un serveur Mumble (<https://mumble.info>) (précédemment connu sous le nom de Murmur).

**mumble-server-configuration** [Type de données]

Le type de service pour le serveur Mumble. Voici un exemple de configuration :

```
(service mumble-server-service-type
  (mumble-server-configuration
    (welcome-text
      "Bienvenue sur ce serveur Mumble qui tourne sur Guix !")
    (cert-required? #t) ;désactive les connexions par mot de passe
    (ssl-cert "/etc/letsencrypt/live/mumble.example.com/fullchain.pem")
    (ssl-key "/etc/letsencrypt/live/mumble.example.com/privkey.pem")))
```

Après avoir reconfiguré votre système, vous pouvez manuellement indiquer le mot de passe `SuperUser` de `mumble-server` avec la commande qui s'affiche pendant la phase d'activation.

Il est recommandé d'enregistrer un compte utilisateur Mumble normal et de lui donner les droits admin ou modérateur. Vous pouvez utiliser le client `mumble` pour vous connecter en tant que nouvel utilisateur normal, vous enregistrer et vous déconnecter. Pour l'étape suivante, connectez-vous avec le nom `SuperUser` en utilisant le mot de passe `SuperUser` que vous avez indiqué précédemment et accordez les droits administrateur ou modérateur à vous utilisateur mumble nouvellement enregistré et créez quelques salons.

Les champs de `mumble-server-configuration` disponibles sont :

- `package` (par défaut : `mumble`)  
Paquet qui contient `bin/mumble-server`.
- `user` (par défaut : `"mumble-server"`)  
Utilisateur qui lancera le serveur Mumble.
- `group` (par défaut : `"mumble-server"`)  
Groupe de l'utilisateur qui lancera le serveur Mumble.
- `port` (par défaut : `64738`)  
Port sur lequel le serveur écoutera.
- `welcome-text` (par défaut : `""`)  
Texte de bienvenue envoyé aux clients lors de leur connexion.
- `server-password` (par défaut : `""`)  
Mot de passe que les clients devront entrer pour se connecter.
- `max-users` (par défaut : `100`)  
Nombre maximum d'utilisateurs qui peuvent se connecter à ce serveur en même temps.
- `max-user-bandwidth` (par défaut : `#f`)  
Trafic de voix maximum qu'un utilisateur peut envoyer par seconde.
- `database-file` (par défaut : `"/var/lib/mumble-server/db.sqlite"`)  
Nom de fichier de la base de données sqlite. L'utilisateur du service deviendra propriétaire du répertoire.
- `log-file` (par défaut : `"/var/log/mumble-server/mumble-server.log"`)  
Nom du fichier de journal. L'utilisateur du service deviendra propriétaire du répertoire.
- `autoban-attempts` (par défaut : `10`)  
Nombre maximum de connexions qu'un utilisateur peut faire pendant `autoban-timeframe` sans être banni automatiquement pour `autoban-time`.
- `autoban-timeframe` (par défaut : `120`)  
Durée du temps pendant lequel le nombre de connexions est compté.

- autoban-time** (par défaut : 300)  
Durée du bannissement automatique en secondes.
- opus-threshold** (par défaut : 100)  
Pourcentage des clients qui doivent supporter opus avant de passer sur le codec audio opus.
- channel-nesting-limit** (par défaut : 10)  
Profondeur maximum des canaux.
- channelname-regex** (par défaut : #f)  
Une chaîne de la forme d'une expression régulière Qt que les noms de canaux doivent respecter.
- username-regex** (par défaut : #f)  
Une chaîne de la forme d'une expression régulière Qt que les noms d'utilisateurs doivent respecter.
- text-message-length** (par défaut : 5000)  
Taille maximum en octets qu'un utilisateur peut envoyer en un seul message textuel.
- image-message-length** (par défaut : (\* 128 1024))  
Taille maximum en octets qu'un utilisateur peut envoyer en une seule image.
- cert-required?** (par défaut : #f)  
Si la valeur est #t les clients utilisant une authentification par mot de passe faible ne seront pas acceptés. Les utilisateurs doivent compléter l'assistant de configuration des certificats pour rejoindre le serveur.
- remember-channel?** (paramètre de : #f)  
Indique si mumble-server devrait se rappeler du dernier canal dans lequel étaient les utilisateurs au moment de leur déconnexion et les y remettre lorsqu'ils se reconnectent.
- allow-html?** (par défaut : #f)  
Indique si le html est autorisé dans les messages textuels, les commentaires utilisateurs et les descriptions des canaux.
- allow-ping?** (par défaut : #f)  
Mettre à vrai expose le nombre d'utilisateurs, le nombre d'utilisateurs maximum et la bande passante maximale du serveur par client aux utilisateurs non connectés. Dans le client Mumble, cette information est affichée dans la boîte de dialogue de connexion.  
Désactiver ce paramètre empêchera le serveur d'être publiquement listé.
- bonjour?** (par défaut : #f)  
Indique si le serveur s'annonce sur le réseau local à travers le protocole bonjour.
- send-version?** (par défaut : #f)  
Indique si la version du serveur mumble-server doit être exposée dans les requêtes ping.



`log-days` (par défaut : `31`)

Mumble stocke aussi les journaux en base de données, qui sont accessible via RPC. La valeur par défaut est 31 jours, mais vous pouvez le mettre à 0 pour les garder pour toujours ou à -1 pour désactiver la journalisation dans la base de données.

`obfuscate-ips?` (par défaut : `#t`)

Indique si les IP enregistrées doivent être cachées pour protéger la vie privée des utilisateurs.

`ssl-cert` (par défaut : `#f`)

Nom de fichier du certificat SSL/TLS utilisé pour les connexions chiffrées.

```
(ssl-cert "/etc/letsencrypt/live/example.com/fullchain.pem")
```

`ssl-key` (par défaut : `#f`)

Chemin de fichier vers la clef privée ssl pour les connexions chiffrées.

```
(ssl-key "/etc/letsencrypt/live/example.com/privkey.pem")
```

`ssl-dh-params` (par défaut : `#f`)

Nom de fichier d'un fichier encodé en PEM avec les paramètres Diffie-Hellman pour le chiffrement SSL/TLS. Autrement vous pouvez indiquer "`@ffdhe2048`", "`@ffdhe3072`", "`@ffdhe4096`", "`@ffdhe6144`" ou "`@ffdhe8192`" pour utiliser les paramètres inclus de la RFC 7919.

`ssl-ciphers` (par défaut : `#f`)

L'option `ssl-ciphers` permet de choisir les suites de chiffrement disponibles pour SSL/TLS.

Cette option est spécifiée en utilisant l'OpenSSL cipher list notation (<https://www.openssl.org/docs/apps/ciphers.html#CIPHER-LIST-FORMAT>).

Nous vous recommandons d'essayer votre chaîne de suites de chiffrements avec « `openssl ciphers <chaîne>` » avant de l'indiquer ici, pour avoir une idée des suites de chiffrement que vous aurez. Après avoir indiqué cette option, nous vous recommandons d'inspecter les journaux du serveur Mumble pour vous assurer que Mumble utilise les suites de chiffrements auxquelles vous vous attendez.

**Remarque:** Modifier cette option peut impacter la rétrocompatibilité de votre serveur Mumble, et peut empêcher que des clients Mumble anciens se connectent.

`public-registration` (par défaut : `#f`)

Doit être un enregistrement `<mumble-server-public-registration-configuration>` ou `#f`.

Vous pouvez aussi enregistrer votre serveur dans la liste des serveurs publiques que le client `mumble` affiche au démarrage. Vous ne pouvez pas enregistrer votre serveur si vous avez un `server-password` ou `allow-ping` à `#f`.

Cela peut prendre quelques heures avant d'arriver sur la liste publique.

`file` (par défaut : `#f`)

Version alternative de cette configuration : si vous indiquez quelque chose, le reste est ignoré.

`mumble-server-public-registration-configuration` [Type de données]  
Configuration pour l'enregistrement public du service `mumble-server`.

`name` C'est le nom d'affichage de votre serveur. Ne pas le confondre avec le nom d'hôte.

`password` Un mot de passe pour identifier votre enregistrement. Les mises à jours suivantes devront utiliser le même mot de passe. Ne le perdez pas.

`url` Cela devrait être le lien `http://` ou `https://` vers votre site web.

`hostname` (par défaut : `#f`)

Par défaut votre serveur sera listé par son adresse IP. Si cette option est indiquée votre serveur sera listé par son nom d'hôte.

**Avertissement d'obsolescence:** Pour des raisons historiques, toutes les procédures `mumble-server-` ci-dessus sont aussi exportées avec le préfixe `murmur-` à la place. Il est recommandé de commencer à utiliser `mumble-server-` à partir de maintenant.

### 12.9.15 Services de partage de fichiers

Le module (`gnu services file-sharing`) fournit des services qui aident au transfert de fichiers sur des réseaux de partage en pair-à-pair.

#### Service du démon Transmission

Transmission (<https://transmissionbt.com/>) est un client BitTorrent flexible qui propose un éventail d'interfaces graphiques et en ligne de commande. Un service `transmission-daemon-service-type` fournit la variante sans interface de Transmission, `transmission-daemon`, en tant que service du système, ce qui permet aux utilisateurs de partager des fichiers via BitTorrent même lorsqu'ils ne sont pas connectés.

`transmission-daemon-service-type` [Variable Scheme]

Le type de service pour le démon du client BitTorrent Transmission. Sa valeur doit être un enregistrement `transmission-daemon-configuration` comme dans cet exemple :

```
(service transmission-daemon-service-type
  (transmission-daemon-configuration
    ;; Restreint l'accès à l'interface RPC ("control")
    (rpc-authentication-required? #t)
    (rpc-username "transmission")
    (rpc-password
      (transmission-password-hash
        "transmission" ; mot de passe souhaité
        "uKd1uMs9")) ; valeur de sel arbitraire

    ;; Accepte les requêtes de cet hôte et d'autres sur le
```

```
;; réseau local
(rpc-whitelist-enabled? #t)
(rpc-whitelist '("::1" "127.0.0.1" "192.168.0.*"))

;; Limite la bande passant utilisée pendant les heures de bureau
(alt-speed-down (* 1024 2)) ; 2 Mo/s
(alt-speed-up 512) ; 512 ko/s

(alt-speed-time-enabled? #t)
(alt-speed-time-day 'weekdays)
(alt-speed-time-begin
 (+ (* 60 8) 30)) ; 8:30
(alt-speed-time-end
 (+ (* 60 (+ 12 5)) 30))) ; 17:30
```

Une fois le service démarré, vous pouvez interagir avec le démon à travers son interface web (sur `https://localhost:9091` ou en utilisant la ligne de commande `transmission-remote`, disponible dans le paquet `transmission`. (les utilisateurs d'Emacs peuvent aussi regarder le paquet `emacs-transmission`). Les deux communiquent avec le démon à travers son interface d'appel de procédure distante (RPC), qui est disponible par défaut pour tous les utilisateurs du système ; vous voudrez peut-être changer cela en indiquant des valeurs dans les paramètres `rpc-authentication-required?`, `rpc-username` et `rpc-password`, comme cela est montré et documenté plus bas.

La valeur de `rpc-password` doit toujours être un hash de mot de passe du type généré et utilisé par les clients Transmission. Il peut être copié directement d'un fichier `settings.json` existant, si au autre client Transmission est déjà utilisé. Sinon, les procédures `transmission-password-hash` et `transmission-random-salt` fournies par ce module peuvent être utilisées pour obtenir un hash convenable.

`transmission-password-hash` *password salt* [Procédure Scheme]

Renvoie une chaîne contenant le résultat du hash de *password* avec *salt*, au format reconnu par les clients Transmission pour leur paramètre `rpc-password`.

*salt* doit être une chaîne de huit caractères. La procédure `transmission-random-salt` est utile pour générer une valeur correcte au hasard.

`transmission-random-salt` [Procédure Scheme]

Renvoie une chaîne contenant une valeur de sel à huit caractères aléatoires du type généré et utilisé par les clients de Transmission, utilisable avec la procédure `transmission-password-hash`.

Ces procédures sont disponibles dans la REPL Guile démarrée avec `guix repl` (voir Section 9.13 [Invoquer `guix repl`], page 176). C'est utile pour obtenir une valeur de sel aléatoire à donner au second paramètres de `transmission-password-hash` comme dans cet exemple de session :

```
$ guix repl
scheme@(guix-user)> ,use (gnu services file-sharing)
scheme@(guix-user)> (transmission-random-salt)
```

```
$1 = "uKd1uMs9"
```

Autrement, vous pouvez générer un hash de mot de passe complet en une seule étape :

```
scheme@(guix-user)> (transmission-password-hash "transmission"
(transmission-random-salt))
$2 = "{c8bbc6d1740cd8dc819a6e25563b67812c1c19c9VtFPfdsX"
```

La chaîne qui en résulte peut être utilisée telle quelle comme valeur de `rpc-password`, ce qui permet de garder les mots de passes cachés même dans la configuration du système d'exploitation.

Les fichiers torrent téléchargés par le démon ne sont directement accessibles qu'aux utilisateurs dans le groupe « `transmission` », qui reçoivent l'accès en lecture seule au répertoire spécifié dans le paramètre `download-dir` (et aussi le répertoire spécifié par `incomplete-dir` si `incomplete-dir-enabled?` vaut `#t`). Les fichiers téléchargés peuvent être déplacés vers un autre répertoire ou supprimés complètement avec `transmission-remote` avec les options `--move` et `--remove-and-delete`.

Si le paramètre `watch-dir-enabled?` vaut `#t`, les utilisateurs du groupe « `transmission` » peuvent aussi placer des fichiers `.torrent` dans le répertoire spécifié par `watch-dir` pour que les torrents correspondants soient ajoutés par le démon. Le paramètre `trash-original-torrent-files?` indique si le démon doit supprimer ces fichiers après les avoir traités.

Certains paramètres de configuration du démon peuvent être changés temporairement par `transmission-remote` et des outils similaires. Pour annuler ces changements, utiliser l'action `reload` du service pour que le démon recharge ses paramètres sur disque :

```
# herd reload transmission-daemon
```

La liste complète des paramètres disponibles est définie par le type de données `transmission-daemon-configuration`.

`transmission-daemon-configuration` [Type de données]

Le type de données représentant les paramètres de configuration du démon `Transmission`. Ils correspondent directement aux paramètres reconnus par les clients `Transmission` dans le fichier `settings.json`.

Les champs de `transmission-daemon-configuration` disponibles sont :

`package transmission` [paramètre de `transmission-daemon-configuration`]

Le paquet `Transmission` à utiliser.

`entier-non-négatif stop-wait-period` [paramètre de `transmission-daemon-configuration`]

`stop-wait-period`

Le temps d'attente à l'arrêt du service avant que `transmission-daemon` ne termine et tue ses processus. Cela permet au démon de tout mettre en ordre et d'envoyer une dernière mise à jour aux trackers avant de s'arrêter. Vous devrez peut-être augmenter cette valeur pour les hôtes lents, ou les hôtes avec une connexion réseau lente.

La valeur par défaut est '10'.

`string download-dir` [paramètre de `transmission-daemon-configuration`]

Le répertoire dans lequel les fichiers torrent sont téléchargés.

La valeur par défaut est `"/var/lib/transmission-daemon/downloads"`.

**boolean** [paramètre de transmission-daemon-configuration]  
**incomplete-dir-enabled?**

Si la valeur est **#t**, les fichiers seront gardés dans **incomplete-dir** tant que leur torrent est en cours de téléchargement, puis déplacés vers **download-dir** une fois le torrent terminé. Sinon, les fichiers pour tous les torrents (dont ceux qui sont toujours en cours) seront placés dans **download-dir**.

La valeur par défaut est **'#f'**.

**peut-être-chaine** [paramètre de transmission-daemon-configuration]  
**incomplete-dir**

Le répertoire dans lequel les fichiers des torrents incomplets seront gardé si **incomplete-dir-enabled?** vaut **#t**.

La valeur par défaut est **'disabled'**.

**umask umask** [paramètre de transmission-daemon-configuration]

Le masque de droits d'accès à la création des fichiers téléchargés. voir la page de manuel de **umask** pour plus d'informations.

La valeur par défaut est **'18'**.

**boolean** [paramètre de transmission-daemon-configuration]  
**name-partial-files?**

Lorsque la valeur est **#t**, **« .part »** est ajouté à la fin du nom des fichiers partiellement téléchargés.

La valeur par défaut est **'#t'**.

**preallocation-mode** [paramètre de transmission-daemon-configuration]  
**preallocation**

Le mode de préallocation d'espace pour les fichiers téléchargés, parmi **none**, **fast** (ou **sparse**) ou **full**. Spécifiez **full** pour minimiser la fragmentation du disque en sacrifiant un peu de vitesse de création du fichier.

La valeur par défaut est **'fast'**.

**boolean** [paramètre de transmission-daemon-configuration]  
**watch-dir-enabled?**

Si la valeur est **#t**, le répertoire spécifié par **watch-dir** sera surveillé en attendant de nouveaux fichiers **.torrent** et les torrents qu'ils décrivent seront ajoutés automatiquement (et le fichier d'origine supprimé, si **trash-original-torrent-files?** vaut **#t**).

La valeur par défaut est **'#f'**.

**peut-être-chaine** [paramètre de transmission-daemon-configuration]  
**watch-dir**

Le répertoire à surveiller pour les fichiers **.torrent** indiquant de nouveaux torrents à ajouter, lorsque **watch-dir-enabled** vaut **#t**.

La valeur par défaut est **'disabled'**.

**boolean** [paramètre de transmission-daemon-configuration]  
**transh-original-torrent-files?**

Lorsque la valeur est **#t**, les fichiers **.torrent** seront supprimés du répertoire de surveillance une fois leur torrent ajouté (voir **watch-directory-enabled?**).

La valeur par défaut est **'#f'**.

**boolean** [paramètre de transmission-daemon-configuration]  
**speed-limit-down-enabled?**

Lorsque la valeur est **#t**, la vitesse de téléchargement du démon sera limitée par la vitesse spécifiée par **speed-limit-down**.

La valeur par défaut est **'#f'**.

**entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**speed-limit-down**

La vitesse de téléchargement par défaut maximale, en kilooctets par seconde.

La valeur par défaut est **'100'**.

**boolean** [paramètre de transmission-daemon-configuration]  
**speed-limit-up-enabled?**

Lorsque la valeur est **#t**, la vitesse de téléversement du démon sera limitée par la vitesse spécifiée par **speed-limit-up**.

La valeur par défaut est **'#f'**.

**entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**speed-limit-up**

La vitesse globale de téléversement maximale par défaut, en kilooctets par seconde.

La valeur par défaut est **'100'**.

**boolean** [paramètre de transmission-daemon-configuration]  
**alt-speed-enabled?**

Lorsque la valeur est **#t**, les limites de vitesse alternatives **alt-speed-down** et **alt-speed-up** sont utilisées (au lieu de **speed-limit-down** et **speed-limit-up**, si elles sont activées) pour contraindre l'utilisation de la bande passante par le démon. Elles peuvent être programmées automatiquement à certains moments de la semaine ; voir **alt-speed-time-enabled?**.

La valeur par défaut est **'#f'**.

**entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**alt-speed-down**

La vitesse de téléchargement globale alternative maximale, en kilooctets par seconde.

La valeur par défaut est **'50'**.

**entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**alt-speed-up**

La vitesse de téléversement globale alternative maximale, en kilooctets par seconde.

La valeur par défaut est **'50'**.

- boolean** [paramètre de transmission-daemon-configuration]  
**alt-speed-time-enabled?**  
Lorsque la valeur est **#t**, les limites de vitesses alternatives **alt-speed-down** et **alt-speed-up** seront activées automatiquement aux périodes spécifiées par **alt-speed-time-day**, **alt-speed-time-begin** et **alt-time-speed-end**.  
La valeur par défaut est **'#f'**.
- day-list** [paramètre de transmission-daemon-configuration]  
**alt-speed-time-day**  
Les jours de la semaine pendant lesquels le programme de vitesse alternative devrait être utilisé, spécifié soit comme une liste de jours (**sunday**, **monday**, etc), soit en utilisant les symboles **weekdays**, **weekends** ou **all**.  
La valeur par défaut est **'all'**.
- entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**alt-speed-time-begin**  
Le moment de la journée auquel activer les limites de vitesse alternatives, en nombre de minutes à partir de minuit.  
La valeur par défaut est **'540'**.
- entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**alt-speed-time-end**  
Le moment de la journée auquel désactiver les limites de vitesses alternatives, en nombre de minutes à partir de minuit.  
La valeur par défaut est **'1020'**.
- string bind-address-ipv4** [paramètre de transmission-daemon-configuration]  
L'IP sur laquelle écouter les connexions de pairs, ou « 0.0.0.0 » pour écouter sur toutes les interfaces réseaux.  
La valeur par défaut est **"0.0.0.0"**.
- string bind-address-ipv6** [paramètre de transmission-daemon-configuration]  
L'adresse IPv6 sur laquelle écouter les connexions de pairs, ou « :: » pour écouter sur toutes les interfaces réseaux.  
La valeur par défaut est **"::"**.
- boolean** [paramètre de transmission-daemon-configuration]  
**peer-port-random-on-start?**  
Si la valeur est **#t**, lorsque le démon démarre il choisira un port au hasard sur lequel écouter les connexions des pairs, à partir d'un intervalle spécifié (inclusivement) par **peer-port-random-low** et **peer-port-random-high**. Sinon, il écoutera sur le port spécifié par **peer-port**.  
La valeur par défaut est **'#f'**.
- port-number** [paramètre de transmission-daemon-configuration]  
**peer-port-random-low**  
Le plus petit numéro de port possible lorsque **peer-port-random-on-start?** vaut **#t**.  
La valeur par défaut est **'49152'**.

`port-number` [paramètre de transmission-daemon-configuration]  
`peer-port-random-high`

Le plus grand numéro de port possible lorsque `peer-port-random-on-start` vaut `#t`.

La valeur par défaut est `'65535'`.

`port-number peer-port` [paramètre de transmission-daemon-configuration]

Le port sur lequel écouter pour les connexions des pairs lorsque `peer-port-random-on-start?` vaut `#f`.

La valeur par défaut est `'51413'`.

`boolean` [paramètre de transmission-daemon-configuration]

`port-forwarding-enabled?`

Si la valeur est `#t`, le démon essayera de configurer le renvoi de port vers une passerelle amont automatiquement avec UPnP et NAT-PMP.

La valeur par défaut est `'#t'`.

`encryption-mode` [paramètre de transmission-daemon-configuration]

`encryption`

Vos préférences de chiffrement pour les connexions avec les pairs, parmi `prefer-unencrypted-connections`, `prefer-encrypted-connections` et `require-encrypted-connections`.

La valeur par défaut est `'prefer-encrypted-connections'`.

`peut-être-chaîne` [paramètre de transmission-daemon-configuration]

`peer-congestion-algorithm`

L'algorithme de contrôle de congestion TCP à utiliser pour les connexions avec les pairs, spécifié avec une chaîne reconnue par le système d'exploitation dans les appels à `setsockopt`. Lorsque la valeur n'est pas spécifiée, le système d'exploitation utilise la valeur par défaut.

Remarquez que sur les systèmes GNU/Linux, le noyau doit être configuré pour permettre aux processus d'utiliser un algorithme de contrôle de congestion qui n'est pas dans l'ensemble par défaut ; sinon, il refusera les demande avec « Opération non autorisée ». Pour voir les algorithmes disponibles sur votre système et ceux qui sont actuellement autorisés, regardez le contenu des fichiers `tcp_available_congestion_control` et `tcp_allowed_congestion_control` dans le répertoire `/proc/sys/net/ipv4`.

Par exemple, pour que le démon Transmission utilise l'algorithme de contrôle de congestion TCP de basse priorité (<http://www-ece.rice.edu/networks/TCP-LP/>), vous devrez modifier la configuration de votre noyau pour construire la prise en charge de cet algorithme, puis mettre à jour votre configuration de système d'exploitation pour permettre son utilisation en ajoutant un service `sysctl-service-type` (ou en mettant à jour la configuration de ce service s'il existe déjà) avec les lignes suivantes :

```
(service sysctl-service-type
  (sysctl-configuration
    (settings
```



```
    ("net.ipv4.tcp_allowed_congestion_control" .
     "reno cubic lp"))))
```

La configuration de démon Transmission peut être mise à jour avec

```
    (peer-congestion-algorithm "lp")
```

et le système peut être reconfiguré pour que les changements prennent effet.

La valeur par défaut est ‘disabled’.

```
tcp-type-of-service      [paramètre de transmission-daemon-configuration]
    peer-socket-tos
```

Le type de service à demander pour les paquets TCP sortants, entre `default`, `low-cost`, `throughput`, `low-delay` et `reliability`.

La valeur par défaut est ‘default’.

```
entier-non-négatif      [paramètre de transmission-daemon-configuration]
    peer-limit-global
```

La limite globale pour le nombre de pairs connectés.

La valeur par défaut est ‘200’.

```
entier-non-négatif      [paramètre de transmission-daemon-configuration]
    peer-limit-per-torrent
```

La limite par torrent de pairs connectés.

La valeur par défaut est ‘50’.

```
entier-non-négatif      [paramètre de transmission-daemon-configuration]
    upload-slots-per-torrent
```

Le nombre maximum de pairs auxquels le démon enverra des données en même temps pour chaque torrent.

La valeur par défaut est ‘14’.

```
entier-non-négatif      [paramètre de transmission-daemon-configuration]
    peer-id-ttl-hours
```

La durée maximum, en heures, de l’identifiant de pair associé à chaque torrent public avant qu’il ne soit régénéré.

La valeur par défaut est ‘6’.

```
boolean                  [paramètre de transmission-daemon-configuration]
    blocklist-enabled?
```

Lorsque la valeur est `#t`, le démon ignorera les pairs mentionnés dans la dernière liste de blocage téléchargée depuis `blocklist-url`.

La valeur par défaut est ‘#f’.

```
peut-être-chaine        [paramètre de transmission-daemon-configuration]
    blocklist-url
```

L’URL d’une liste de blocage de pairs (au format P2P-plaintext ou eMule `.dat`) périodiquement téléchargés et appliqués lorsque `blocklist-enabled?` vaut `#t`.

La valeur par défaut est ‘disabled’.

**boolean** [paramètre de transmission-daemon-configuration]  
**download-queue-enabled?**

Si la valeur est **#t**, le démon sera limité à télécharger au plus **download-queue-size** torrents non bloqués en même temps.

La valeur par défaut est '**#t**'.

**entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**download-queue-size**

La taille de la queue de téléchargement du démon, qui limite le nombre de torrents non bloqués qu'il télécharge en même temps lorsque **download-queue-enabled?** vaut **#t**.

La valeur par défaut est '**5**'.

**boolean** [paramètre de transmission-daemon-configuration]  
**seed-queue-enabled?**

Si la valeur est **#t**, le démon sera limité à envoyer au plus **seed-queue-size** torrents non bloqués en même temps.

La valeur par défaut est '**#f**'.

**entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**seed-queue-size**

La taille de la queue d'envoi du démon, qui limite le nombre de torrents non bloqué qu'il enverra en même temps lorsque **seed-queue-enabled?** vaut **#t**.

La valeur par défaut est '**10**'.

**boolean** [paramètre de transmission-daemon-configuration]  
**queue-stalled-enabled?**

Lorsque la valeur est **#t**, le démon considère les torrents pour lesquels il n'a pas partagé de données dans les dernières **queue-stalled-minutes** comme bloqué et ne les contera pas dans les limites **download-queue-size** et **seed-queue-size**.

La valeur par défaut est '**#t**'.

**entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**queue-stalled-minutes**

La période maximale, en minutes, pendant laquelle un torrent peut être inactif avant d'être considéré comme bloqué, lorsque **queue-stalled-enabled?** vaut **#t**.

La valeur par défaut est '**30**'.

**boolean** [paramètre de transmission-daemon-configuration]  
**ratio-limit-enabled?**

Lorsque la valeur est **#t**, un torrent téléchargé sera automatiquement mis en pause une fois qu'il atteint le ratio spécifié par **ratio-limit**.

La valeur par défaut est '**#f**'.

**non-negative-rational** [paramètre de transmission-daemon-configuration]  
**ratio-limit**

Le ratio auquel un torrent téléchargé sera mis en pause, lorsque **ratio-limit-enabled?** vaut **#t**.

La valeur par défaut est '**2.0**'.

**boolean** `idle-seeding-limit-enabled?` [paramètre de transmission-daemon-configuration]

Lorsque la valeur est `#t`, un torrent téléchargé sera automatiquement mis en pause s'il est inactif depuis `idle-seeding-limit` minutes.

La valeur par défaut est `'#f'`.

**entier-non-négatif** `idle-seeding-limit` [paramètre de transmission-daemon-configuration]

La durée maximum, en minutes, pendant laquelle un torrent téléchargé peut être inactif avant d'être mis en pause, quand `idle-seeding-limit-enabled?` vaut `#t`.

La valeur par défaut est `'30'`.

**boolean** `dht-enabled?` [paramètre de transmission-daemon-configuration]

Active le protocole de table de hash (DHT) distribuée ([http://bittorrent.org/beps/bep\\_0005.html](http://bittorrent.org/beps/bep_0005.html)), qui prend en charge les torrents sans tracker.

La valeur par défaut est `'#t'`.

**boolean** `lpd-enabled?` [paramètre de transmission-daemon-configuration]

Active la découverte de pairs locaux ([https://en.wikipedia.org/wiki/Local\\_Peer\\_Discovery](https://en.wikipedia.org/wiki/Local_Peer_Discovery)) (LPD), qui permet de découvrir des pairs sur le réseau local et peut réduire la quantité de données envoyées sur Internet.

La valeur par défaut est `'#f'`.

**boolean** `pex-enabled?` [paramètre de transmission-daemon-configuration]

Active l'échange de pairs ([https://en.wikipedia.org/wiki/Peer\\_exchange](https://en.wikipedia.org/wiki/Peer_exchange)) (PEX), qui réduit la dépendance du démon aux trackers externes et peut améliorer les performances.

La valeur par défaut est `'#t'`.

**boolean** `utp-enabled?` [paramètre de transmission-daemon-configuration]

Active le protocole de micro transport ([http://bittorrent.org/beps/bep\\_0029.html](http://bittorrent.org/beps/bep_0029.html)) (uTP), qui cherche à réduire l'impact du trafic BitTorrent sur les autres utilisateurs du réseau local tout en utilisant toute la bande passante disponible.

La valeur par défaut est `'#t'`.

**boolean** `rpc-enabled?` [paramètre de transmission-daemon-configuration]

Si la valeur est `#t`, active l'interface d'appel de procédures distantes (RPC), qui permet de contrôler le démon à distance via son interface web, le client `transmission-remote` en ligne de commande et des outils similaires.

La valeur par défaut est `'#t'`.

**string** `rpc-bind-address` [paramètre de transmission-daemon-configuration]

L'adresse IP sur laquelle écouter les connexions RPC, ou « 0.0.0.0 » pour écouter sur toutes les adresses IP.

La valeur par défaut est `"0.0.0.0"`.

**port-number** `rpc-port` [paramètre de transmission-daemon-configuration]

Le port sur lequel écouter les connexions RPC entrantes.

La valeur par défaut est `'9091'`.

**string** `rpc-url` [paramètre de `transmission-daemon-configuration`]  
Le préfixe de chemin à utiliser dans l'URL du point d'accès RPC.

La valeur par défaut est `"/transmission/"`.

**boolean** `rpc-authentication-required?` [paramètre de `transmission-daemon-configuration`]

Lorsque la valeur est `#t`, les clients doivent s'authentifier (voir `rpc-username` et `rpc-password`) quand ils utilisent l'interface RPC. Remarquez que cela a pour effet de bord de désactiver la liste d'hôtes (voir `rpc-host-whitelist-enabled?`).

La valeur par défaut est `#f`.

**peut-être-chaîne** `rpc-username` [paramètre de `transmission-daemon-configuration`]

Le nom d'utilisateur requis par les clients pour accéder à l'interface RPC lorsque `rpc-authentication-required?` vaut `#t`.

La valeur par défaut est `'disabled'`.

**peut-être-transmission-password-hash** `rpc-password` [paramètre de `transmission-daemon-configuration`]

Le mot de passe requis pour permettre aux clients d'accéder à l'interface RPC lorsque `rpc-authentication-required?` vaut `#t`. Il doit être spécifié avec un hash de mot de passe au format reconnu par les clients de Transmission, soit copié d'un fichier `settings.json` existant, soit généré avec la procédure `transmission-password-hash`.

La valeur par défaut est `'disabled'`.

**boolean** `rpc-whitelist-enabled?` [paramètre de `transmission-daemon-configuration`]

Lorsque la valeur est `#t`, les requêtes de RPC seront acceptées seulement si elles proviennent d'une adresse spécifiée dans `rpc-whitelist`.

La valeur par défaut est `#t`.

**string-list** `rpc-whitelist` [paramètre de `transmission-daemon-configuration`]

La liste des adresse IP et IPv6 pour lesquelles les requêtes RPC sont acceptées lorsque `rpc-whitelist-enabled?` vaut `#t`. Vous pouvez spécifier des jokers avec `*`.

La valeur par défaut est `("127.0.0.1" ":::1")`.

**boolean** `rpc-host-whitelist-enabled?` [paramètre de `transmission-daemon-configuration`]

Lorsque la valeur est `#t`, les requêtes RPC seront acceptée lorsqu'elles sont adressées à un hôte nommé dans `rpc-host-whitelist`. Remarquez que les requêtes adressées à « localhost » ou « localhost. » ou à une adresse numérique, sont toujours acceptées indépendamment de ce paramètre.

Remarquez aussi que cette fonctionnalité est désactivée lorsque `rpc-authentication-required?` vaut `#t`.

La valeur par défaut est `#t`.

- string-list** [paramètre de transmission-daemon-configuration]  
**rpc-host-whitelist**  
La liste des noms d'hôte reconnus par le serveur RPC lorsque `rpc-host-whitelist-enabled?` vaut `#t`.  
La valeur par défaut est `('')`.
- message-level** [paramètre de transmission-daemon-configuration]  
**message-level**  
Le niveau de sévérité minimum des messages à enregistrer dans les journaux (dans `/var/log/transmission.log`) par le démon, entre `none` (par de journalisation), `error`, `info` et `debug`.  
La valeur par défaut est `'info'`.
- boolean** [paramètre de transmission-daemon-configuration]  
**start-added-torrents?**  
Lorsque la valeur est `#t`, les torrents sont démarrés dès qu'ils sont ajoutés ; sinon ils sont ajouté et mis en pause.  
La valeur par défaut est `'#t'`.
- boolean** [paramètre de transmission-daemon-configuration]  
**script-torrent-done-enabled?**  
Lorsque la valeur est `#t`, le script spécifié par `script-torrent-done-filename` sera invoqué à chaque fois qu'un torrent termine.  
La valeur par défaut est `'#f'`.
- peut-être-objet-fichier** [paramètre de transmission-daemon-configuration]  
**script-torrent-done-filename**  
Un nom de fichier ou un simili-fichier spécifiant un script à lancer à chaque fois qu'un torrent termine, lorsque `script-torrent-done-enabled?` est `#t`.  
La valeur par défaut est `'disabled'`.
- boolean** [paramètre de transmission-daemon-configuration]  
**scrape-paused-torrents-enabled?**  
Lorsque la valeur est `#t`, le démon contactera les trackers pour le torrent même si le torrent est en pause.  
La valeur par défaut est `'#t'`.
- entier-non-négatif** [paramètre de transmission-daemon-configuration]  
**cache-size-mb**  
La quantité de mémoire, en mégaoctets, à allouer pour le cache mémoire du démon.  
Une grande valeur peut améliorer les performances en réduisant le fréquence des entrées-sorties sur le disque.  
La valeur par défaut est `'4'`.
- boolean** [paramètre de transmission-daemon-configuration]  
**prefetch-enabled?**  
Lorsque la valeur est `#t`, le démon essaiera d'améliorer les performances des entrées-sorties en disant au système quelles données seront probablement bientôt lues pour satisfaire les requêtes des pairs.

La valeur par défaut est `#t`.

## 12.9.16 Services de surveillance

### Service Tailon

Tailon (<https://tailon.readthedocs.io/>) est une application web pour visualiser et chercher des fichiers de journaux.

L'exemple suivant configurera le service avec les valeurs par défaut. Par défaut, on peut accéder à Tailon sur le port 8080 (<http://localhost:8080>).

```
(service tailon-service-type)
```

L'exemple suivant personnalise un peu plus la configuration de Tailon, en ajoutant `sed` à la liste des commandes autorisées.

```
(service tailon-service-type
  (tailon-configuration
    (config-file
      (tailon-configuration-file
        (allowed-commands '("tail" "grep" "awk" "sed"))))))
```

**tailon-configuration** [Type de données]

Type de données représentant la configuration de Tailon. Ce type a les paramètres suivants :

**config-file** (par défaut : `(tailon-configuration-file)`)

Le fichier de configuration à utiliser pour Tailon. Ce champ peut contenir un enregistrement *tailon-configuration-file* ou n'importe quelle gexp (voir Section 9.12 [G-Expressions], page 166).

Par exemple, pour utiliser un fichier local à la place, on peut utiliser la fonction `local-file` :

```
(service tailon-service-type
  (tailon-configuration
    (config-file (local-file "./my-tailon.conf"))))■
```

**package** (par défaut : `tailon`)

Le paquet tailon à utiliser.

**tailon-configuration-file** [Type de données]

Type de données représentant les options de configuration de Tailon. Ce type a les paramètres suivants :

**files** (par défaut : `(list "/var/log")`)

Liste des fichiers à afficher. La liste peut inclure des chaînes pour des fichiers simple ou des répertoires, ou une liste, où le premier élément est le nom d'une sous-section et le reste des fichiers ou des répertoires de cette sous-section.

**bind** (par défaut : `"localhost:8080"`)

Adresse et port sur lesquels Tailon écoute.

**relative-root** (par défaut : **#f**)  
Chemin de l'URL à utiliser pour Tailon, ou **#f** pour ne pas utiliser de chemin.

**allow-transfers?** (par défaut : **#t**)  
Permet de télécharger les journaux dans l'interface web.

**follow-names?** (par défaut : **#t**)  
Permet de surveiller des fichiers qui n'existent pas encore.

**tail-lines** (par défaut : 200)  
Nombre de lignes à lire initialement dans chaque fichier.

**allowed-commands** (par défaut : (list "tail" "grep" "awk"))  
Commandes autorisées. Par défaut, **sed** est désactivé.

**debug?** (par défaut : **#f**)  
Configurez **debug?** à **#t** pour montrer les messages de débogage.

**wrap-lines** (par défaut : **#t**)  
État initial du retour à la ligne dans l'interface web. Configurez l'option à **#t** pour retourner à la ligne (par défaut) ou à **#f** pour ne pas retourner à la ligne au début.

**http-auth** (par défaut : **#f**)  
Type d'authentification HTTP à utiliser. Indiquez **#f** pour désactiver l'authentification (par défaut). Les valeurs supportées sont "digest" et "basic".

**users** (par défaut : **#f**)  
Si l'authentification HTTP est activée (voir **http-auth**), l'accès sera restreint aux identifiants fournis ici. Pour configurer des utilisateurs, utilisez une liste de paires, où le premier élément de la paire est le nom d'utilisateur et le second élément est le mot de passe.

```
(tailon-configuration-file
  (http-auth "basic")
  (users      '(("user1" . "password1")
                ("user2" . "password2"))))
```

## Service Darkstat

Darkstat est un « renifleur de paquets » qui capture le trafic réseau, calcul des statistiques sur l'utilisation et sert des rapport sur HTTP.

**Variable Scheme** *darkstat-service-type* [Variable]  
C'est le type de service pour le service darkstat (<https://unix4lyfe.org/darkstat/>), sa valeur doit être un enregistrement **darkstat-configuration** comme dans cet exemple :

```
(service darkstat-service-type
  (darkstat-configuration
    (interface "eno1")))
```

<b>darkstat-configuration</b>	[Type de données]
Type de données représentant la configuration de <code>darkstat</code> .	
<b>package</b> (par défaut : <code>darkstat</code> )	Le paquet <code>darkstat</code> à utiliser.
<b>interface</b>	Capture le trafic sur l'interface réseau spécifiée.
<b>port</b> (par défaut : <code>"667"</code> )	Lie l'interface web sur le port spécifié.
<b>bind-address</b> (par défaut : <code>"127.0.0.1"</code> )	Lie l'interface web sur l'adresse spécifiée.
<b>base</b> (par défaut : <code>"/"</code> )	Spécifie le chemin de base des URL. C'est utile si on accède à <code>darkstat</code> à travers un proxy inverse.

## Service d'export de nœud de Prometheus

L'exportateur de nœuds de Prometheus rend disponible les statistiques sur le matériel et le système d'exploitation fournies par le noyau Linux pour le système de surveillance Prometheus. Ce service devrait être déployé sur tous les nœuds physiques et les machines virtuelles, où vous voulez surveiller ces statistiques.

<b>Variable Scheme</b> <i>prometheus-node-exporter-service-type</i>	[Variable]
C'est le type de service pour le service <code>prometheus-node-exporter</code> ( <a href="https://github.com/prometheus/node_exporter/">https://github.com/prometheus/node_exporter/</a> ), sa valeur doit être un enregistrement <code>prometheus-node-exporter-configuration</code> .	
(service <code>prometheus-node-exporter-service-type</code> )	

<b>prometheus-node-exporter-configuration</b>	[Type de données]
Type de données représentant la configuration de <code>node_exporter</code> .	
<b>package</b> (par défaut : <code>go-github-com-prometheus-node-exporter</code> )	Le paquet <code>prometheus-node-exporter</code> à utiliser.
<b>web-listen-address</b> (par défaut : <code>":9100"</code> )	Lie l'interface web sur l'adresse spécifiée.
<b>textfile-directory</b> (par défaut : <code>"/var/lib/prometheus/node-exporter"</code> )	Ce répertoire est utilisable pour exporter des métriques spécifiques à cette machine. Les fichiers contenant les métriques au format texte, dont le nom de fichier termine par <code>.prom</code> devraient être placés dans ce répertoire.
<b>extra-options</b> (par défaut : <code>'()</code> )	Options supplémentaires à passer au lancement de l'exportateur de nœuds de Prometheus.

## Server zabbix

Zabbix est un système de surveillance très performant qui peut récupérer des données de plusieurs sources et fournir les résultats sur une interface web. Les alertes et les rapport



sont inclus, ainsi que les *modèles* pour les métriques communes des systèmes d'exploitation comme l'utilisation du réseau, la charge CPU et la consommation de l'espace disque.

Ce service fournit le service de surveillance Zabbix central. Vous aurez aussi besoin de [zabbix-front-end], page 432, pour configurer Zabbix et afficher les résultats et éventuellement de [zabbix-agent], page 431, sur les machines qui doivent être surveillées (d'autres sources de données sont prises en charge, comme [prometheus-node-exporter], page 429).

**Variable** `Scheme zabbix-server-service-type` [Variable]

C'est le type de service pour le service du serveur Zabbix. Sa valeur doit être un enregistrement `zabbix-serveur-configuration`, décrit ci-dessous.

**zabbix-server-configuration** [Type de données]

Les champs de `zabbix-server-configuration` disponibles sont :

`zabbix-server` (par défaut : `zabbix-server`) (type : simili-fichier)

Le paquet `zabbix-server`.

`user` (par défaut : `"zabbix"`) (type : chaîne)

Utilisateur qui lancera le serveur Zabbix.

`group` (par défaut : `"zabbix"`) (type : groupe)

Groupe qui lancera le serveur Zabbix.

`db-host` (par défaut : `"127.0.0.1"`) (type : chaîne)

Le nom d'hôte de la base de données.

`db-name` (par défaut : `"zabbix"`) (type : chaîne)

Nom de la base de données.

`db-user` (par défaut : `"zabbix"`) (type : chaîne)

Utilisateur de la base de données.

`db-password` (par défaut : `""`) (type : chaîne)

Mot de passe de la base de données. Utilisez plutôt `include-files` avec `DBPassword=SECRET` dans le fichier spécifié à la place.

`db-port` (par défaut : `5432`) (type : entier)

Port de la base de données.

`log-type` (par défaut : `""`) (type : chaîne)

Spécifie où les messages de journalisation seront écrits :

- `system` - syslog.
- `file` - fichier spécifié par le paramètre `log-file`.
- `console` - sortie standard.

`log-file` (par défaut : `"/var/log/zabbix/server.log"`) (type : chaîne)

Nom du fichier de journal lorsque le paramètre `log-type` vaut `file`.

`pid-file` (par défaut : `"/var/run/zabbix/zabbix_server.pid"`) (type : chaîne)

Nom du fichier de PID.

- ssl-ca-location** (par défaut : `"/etc/ssl/certs/ca-certificates.crt"`) (type : chaîne)  
 Emplacement des fichiers d'autorités de certification (AC) pour la vérification des certificats SSL du serveur.
- ssl-cert-location** (par défaut : `"/etc/ssl/certs"`) (type : chaîne)  
 Emplacement des certificats SSL des clients.
- extra-options** (par défaut : `""`) (type : extra-options)  
 Options supplémentaires ajoutées à la fin du fichier de configuration du serveur Zabbix.
- include-files** (par défaut : `()`) (type : include-files)  
 Vous pouvez inclure des fichiers individuels ou tous les fichiers d'un répertoire dans le fichier de configuration.

## Agent zabbix

L'agent Zabbix récupère les informations sur le système pour le serveur de surveillance Zabbix. Il a plusieurs vérificateurs intégrés et peut être étendu avec des *paramètres utilisateurs* (<https://www.zabbix.com/documentation/current/en/manual/config/items/userparameters>) personnalisés.

**Variable Scheme *zabbix-agent-service-type*** [Variable]  
 C'est le type de service du service de l'agent Zabbix. Sa valeur doit être un enregistrement `zabbix-agent-configuration`, décrit ci-dessous.

***zabbix-agent-configuration*** [Type de données]  
 Les champs de `zabbix-agent-configuration` disponibles sont :

- zabbix-agent** (par défaut : `zabbix-agentd`) (type : simili-fichier)  
 Le paquet zabbix-agent.
- user** (par défaut : `"zabbix"`) (type : chaîne)  
 Utilisateur qui lancera l'agent Zabbix.
- group** (par défaut : `"zabbix"`) (type : groupe)  
 Groupe qui lancera l'agent Zabbix.
- hostname** (par défaut : `""`) (type : chaîne)  
 Noms d'hôte unique et sensible à la casse requis pour les vérifications actives et qui doit correspondre au nom d'hôte configuré sur le serveur.
- log-type** (par défaut : `""`) (type : chaîne)  
 Spécifie où les messages de journalisation seront écrits :
- **system** - syslog.
  - **file** — fichier spécifié par le paramètre `log-file`.
  - **console** - sortie standard.
- log-file** (par défaut : `"/var/log/zabbix/agent.log"`) (type : chaîne)  
 Nom du fichier de journal lorsque le paramètre `log-type` vaut `file`.
- pid-file** (par défaut : `"/var/run/zabbix/zabbix_agent.pid"`) (type : chaîne)  
 Nom du fichier de PID.

- server** (par défaut : ("127.0.0.1")) (type : liste)  
Liste d'adresses IP, éventuellement en notation CIDR ou de noms d'hôtes de serveurs Zabbix et de mandataires Zabbix. Les connexions entrantes ne seront acceptées que si elles viennent des hôtes listés ici.
- server-active** (par défaut : ("127.0.0.1")) (type : liste)  
Liste de paires d'IP:port (ou nom d'hôte:port) de serveurs Zabbix et de mandataires Zabbix pour les vérifications actives. Si le port n'est pas spécifié, le port par défaut est utilisé. Si ce paramètre n'est pas spécifié, les vérifications actives sont désactivées.
- extra-options** (par défaut : "") (type : extra-options)  
Options supplémentaires ajoutées à la fin du fichier de configuration du serveur Zabbix.
- include-files** (par défaut : ()) (type : include-files)  
Vous pouvez inclure des fichiers individuels ou tous les fichiers d'un répertoire dans le fichier de configuration.

## Interface utilisateur Zabbix

L'interface Zabbix fournit une interface web à Zabbix. Il n'a pas besoin de tourner sur la même machine que le serveur Zabbix. Ce service fonctionne en étendant les service [PHP-FPM], page 454, et [NGINX], page 444, avec la configuration nécessaire à charger l'interface utilisateur Zabbix.

**Variable** *Scheme zabbix-front-end-service-type* [Variable]  
C'est le type de service de l'interface web de Zabbix. Sa valeur doit être un enregistrement *virtlog-configuration*, décrit plus bas.

**zabbix-front-end-configuration** [Type de données]  
Les champs de *zabbix-front-end-configuration* disponibles sont :

- zabbix-server** (par défaut : *zabbix-server*) (type : simili-fichier)  
Le paquet du serveur Zabbix à utiliser.
- nginx** (par défaut : ()) (type : liste)  
Liste de blocs [*nginx-server-configuration*], page 447, pour l'interface Zabbix. Lorsque la valeur est vide, un bloc par défaut qui écoute sur le port 80 est utilisé.
- db-host** (par défaut : "localhost") (type : chaîne)  
Le nom d'hôte de la base de données.
- db-port** (par défaut : 5432) (type : entier)  
Port de la base de données.
- db-name** (par défaut : "zabbix") (type : chaîne)  
Nom de la base de données.
- db-user** (par défaut : "zabbix") (type : chaîne)  
Utilisateur de la base de données.
- db-password** (par défaut : "") (type : chaîne)  
Mot de passe de la base de données. Utilisez plutôt *db-secret-file*.

- db-secret-file** (par défaut : "") (type : chaîne)  
Fichier de secrets qui sera ajouté au fichier `zabbix.conf.php`. Ce fichier contient les paramètres d'authentification utilisés par Zabbix. On s'attend à ce que vous le créiez manuellement.
- zabbix-host** (par défaut : "localhost") (type : chaîne)  
Nom d'hôte du serveur Zabbix.
- zabbix-port** (par défaut : 10051) (type : nombre)  
Port du serveur Zabbix.

### 12.9.17 Services Kerberos

Le module (`gnu services kerberos`) fournit des services liés au protocole d'authentification *Kerberos*.

#### Service Krb5

Les programmes qui utilisent une bibliothèque cliente Kerberos s'attendent à trouver un fichier de configuration dans `/etc/krb5.conf`. Ce service génère un tel fichier à partir d'une définition fournie par la déclaration de système d'exploitation. Il ne démarre aucun démon.

Aucun fichier « keytab » n'est fourni par ce service — vous devez les créer explicitement. Ce service est connu pour fonctionner avec la bibliothèque cliente MIT, `mit-krb5`. Les autres implémentations n'ont pas été testées.

**krb5-service-type** [Variable Scheme]  
Un type de service pour les clients Kerberos 5.

Voici un exemple d'utilisation :

```
(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"))))))))
```

Cet exemple fournit une configuration cliente Kerberos 5 qui :

- Reconnait deux domaines : « EXAMPLE.COM » et « ARGREX.EDU », tous deux aillant des serveurs d'administration et des centres de distribution de clefs distincts ;
- Utilisera le domaine « EXAMPLE.COM » pr défaut si le domaine n'est pas spécifié explicitement par les clients ;
- Acceptera les services qui ne supportent que des types de chiffrements connus pour être faibles.

Les types `krb5-realm` et `krb5-configuration` ont de nombreux champs. Seuls les plus communs sont décrits ici. Pour une liste complète, et plus de détails sur chacun d'entre eux, voir la documentation de MIT `krb5.conf`.

**krb5-realm** [Type de données]

**name** Ce champ est une chaîne identifiant le nom d'un domaine. Une convention courante est d'utiliser le nom pleinement qualifié de votre organisation, converti en majuscule.

**admin-server** Ce champ est une chaîne identifiant l'hôte où le serveur d'administration tourne.

**kdc** Ce champ est une chaîne identifiant le centre de distribution de clefs pour ce domaine.

**krb5-configuration** [Type de données]

**allow-weak-crypto?** (par défaut : `#f`)  
Si ce drapeau est `#t` les services qui n'offrent que des algorithmes de chiffrement faibles seront acceptés.

**default-realm** (par défaut : `#f`)  
Ce champ devrait être une chaîne identifiant le domaine Kerberos par défaut pour le client. Vous devriez mettre le nom de votre domaine Kerberos dans ce champ. Si cette valeur est `#f` alors un domaine doit être spécifié pour chaque principal Kerberos à l'invocation des programmes comme `kinit`.

**realms** Cela doit être une liste non-vide d'objets `krb5-realm`, auxquels les clients peuvent accéder. Normalement, l'un d'entre eux aura un champ `name` qui correspond au champ `default-realm`.

## Service PAM `krb5`

Le service `pam-krb5` permet la connexion et la gestion des mots de passe par Kerberos. Vous aurez besoin de ce service si vous voulez que les applications qui utilisent PAM puissent authentifier automatiquement les utilisateurs avec Kerberos.

**pam-krb5-service-type** [Variable Scheme]  
Un type de service pour le module PAM Kerberos 5.

**pam-krb5-configuration** [Type de données]  
Type de données représentant la configuration du module PAM Kerberos 5. Ce type a les paramètres suivants :

**pam-krb5** (par défaut : `pam-krb5`)  
Le paquet `pam-krb5` à utiliser.

**minimum-uid** (par défaut : 1000)  
Le plus petite ID utilisateur pour lequel les authentifications Kerberos devraient être tentées. Les comptes locaux avec une valeur plus petite échoueront silencieusement leur authentification Kerberos.

### 12.9.18 Services LDAP

Le module (`gnu services authentication`) fournit le type de service `nslcd-service-type`, qui peut être utilisé pour l'authentification par LDAP. En plus de configurer le service lui-même, vous pouvez ajouter `ldap` comme service de noms au Name Service Switch. Voir Section 12.12 [Name Service Switch], page 573, pour des informations détaillées.

Voici une déclaration de système d'exploitation simple avec une configuration par défaut pour `nslcd-service-type` et une configuration du Name Service Switch qui consulte le service de noms `ldap` en dernier :

```
(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))))))
```

Les champs de `nslcd-configuration` disponibles sont :

**package nss-pam-ldapd** [paramètre de `nslcd-configuration`]

Le paquet `nss-pam-ldapd` à utiliser.

**peut-être-nombre threads** [paramètre de `nslcd-configuration`]

Le nombre de threads à démarrer qui peuvent gérer les requête et effectuer des requêtes LDAP. Chaque thread ouvre une connexion séparée au serveur LDAP. La valeur par défaut est de 5 threads.

La valeur par défaut est `'disabled'`.

**string uid** [paramètre de `nslcd-configuration`]

Cela spécifie l'id de l'utilisateur sous lequel le démon devrait tourner.

La valeur par défaut est `"nslcd"`.

**string gid** [paramètre de `nslcd-configuration`]

Cela spécifie l'id du groupe sous lequel le démon devrait tourner.

La valeur par défaut est `"nslcd"`.

- log-option log** [paramètre de `nslcd-configuration`]  
Cette option contrôle la journalisation via une liste contenant le schéma et le niveau. Le schéma peut être soit un symbole `'none'`, `'syslog'`, soit un nom de fichier absolu. Le niveau est facultatif et spécifie le niveau de journalisation. Le niveau de journalisation peut être l'un des symboles suivants : `'crit'`, `'error'`, `'warning'`, `'notice'`, `'info'` ou `'debug'`. Tous les messages avec le niveau spécifié ou supérieurs sont enregistrés.  
La valeur par défaut est `'("/var/log/nslcd" info)'`.
- list uri** [paramètre de `nslcd-configuration`]  
La liste des URI des serveurs LDAP. Normalement, seul le premier serveur sera utilisé avec les serveurs suivants comme secours.  
La valeur par défaut est `'("ldap://localhost:389/")'`.
- peut-être-chaine ldap-version** [paramètre de `nslcd-configuration`]  
La version du protocole LDAP à utiliser. La valeur par défaut est d'utiliser la version maximum supportée par la bibliothèque LDAP.  
La valeur par défaut est `'disabled'`.
- peut-être-chaine binddn** [paramètre de `nslcd-configuration`]  
Spécifie le nom distingué avec lequel se lier au serveur de répertoire pour les recherches.  
La valeur par défaut est de se lier anonymement.  
La valeur par défaut est `'disabled'`.
- peut-être-chaine bindpw** [paramètre de `nslcd-configuration`]  
Spécifie le mot de passe avec lequel se lier. Cette option n'est valable que lorsqu'elle est utilisée avec `binddn`.  
La valeur par défaut est `'disabled'`.
- peut-être-chaine rootpwmoddn** [paramètre de `nslcd-configuration`]  
Spécifie le nom distingué à utiliser lorsque l'utilisateur `root` essaye de modifier le mot de passe d'un utilisateur avec le module PAM.  
La valeur par défaut est `'disabled'`.
- peut-être-chaine rootpwmodpw** [paramètre de `nslcd-configuration`]  
Spécifie le mot de passe à utiliser pour se lier si l'utilisateur `root` essaye de modifier un mot de passe utilisateur. Cette option n'est valable que si elle est utilisée avec `rootpwmoddn`.  
La valeur par défaut est `'disabled'`.
- peut-être-chaine sasl-mech** [paramètre de `nslcd-configuration`]  
Spécifie le mécanisme SASL à utiliser lors de l'authentification SASL.  
La valeur par défaut est `'disabled'`.
- peut-être-chaine sasl-realm** [paramètre de `nslcd-configuration`]  
Spécifie le royaume SASL à utiliser pour effectuer une authentification SASL.  
La valeur par défaut est `'disabled'`.

- peut-être-chaine sasl-authcid** [paramètre de `nslcd-configuration`]  
Spécifie l'identité d'authentification à utiliser pour effectuer une authentification SASL.  
La valeur par défaut est `'disabled'`.
- peut-être-chaine sasl-authzid** [paramètre de `nslcd-configuration`]  
Spécifie l'identité d'autorisation à utiliser lors d'une authentification SASL.  
La valeur par défaut est `'disabled'`.
- peut-être-booléen sasl-canonicalize?** [paramètre de `nslcd-configuration`]  
Détermine si le nom d'hôte du serveur LDAP devrait être canonalisé. Si c'est activé la bibliothèque LDAP effectuera une recherche de nom d'hôte inversée. Par défaut, il est laissé à la bibliothèque LDAP le soin de savoir si la vérification doit être effectuée ou non.  
La valeur par défaut est `'disabled'`.
- peut-être-chaine krb5-ccname** [paramètre de `nslcd-configuration`]  
Indique le nom du cache d'informations de connexion de GSS-API Kerberos.  
La valeur par défaut est `'disabled'`.
- string base** [paramètre de `nslcd-configuration`]  
La base de recherche de répertoires.  
La valeur par défaut est `"dc=example,dc=com"`.
- scope-option scope** [paramètre de `nslcd-configuration`]  
Spécifie la portée de la recherche (`subtree`, `onelevel`, `base` ou `children`). La portée par défaut est `subtree` ; la portée `base` n'est presque jamais utile pour les recherches de service de noms ; la portée `children` n'est pas prise en charge par tous les serveurs.  
La valeur par défaut est `'(subtree)'`.
- peut-être-deref-option deref** [paramètre de `nslcd-configuration`]  
Spécifie la politique de déréférencement des alias. La politique par défaut est de ne jamais déréférencer d'alias.  
La valeur par défaut est `'disabled'`.
- peut-être-booléen referrals** [paramètre de `nslcd-configuration`]  
Spécifie s'il faut activer le suivi de référence. Le comportement par défaut est de suivre les références.  
La valeur par défaut est `'disabled'`.
- list-of-map-entries maps** [paramètre de `nslcd-configuration`]  
Cette option permet d'ajouter des attributs personnalisés à rechercher à la place des attributs par défaut de la RFC 2307. C'est une liste de correspondances, consistant chacune en un nom, en l'attribut RFC 2307 à utiliser et l'expression de la requête pour l'attribut tel qu'il sera disponible dans le répertoire.  
La valeur par défaut est `'()''`.



- list-of-filter-entries filters** [paramètre de `nslcd-configuration`]  
Une liste de filtres consistant en le nom d'une correspondance à laquelle applique le filtre et en une expression de filtre de recherche LDAP.  
La valeur par défaut est '()'.  
**peut-être-nombre bind-timelimit** [paramètre de `nslcd-configuration`]  
Spécifie la limite de temps en seconds à utiliser lors de la connexion au serveur de répertoire. La valeur par défaut est de 10 secondes.  
La valeur par défaut est 'disabled'.  
**peut-être-nombre timelimit** [paramètre de `nslcd-configuration`]  
Spécifie la limite de temps (en secondes) à attendre une réponse d'un serveur LDAP.  
La valeur de zéro, par défaut, permet d'attendre indéfiniment la fin des recherches.  
La valeur par défaut est 'disabled'.  
**peut-être-nombre idle-timelimit** [paramètre de `nslcd-configuration`]  
Spécifie la période d'inactivité (en seconde) après laquelle la connexion au serveur LDAP sera fermée. La valeur par défaut est de ne jamais la fermer.  
La valeur par défaut est 'disabled'.  
**peut-être-nombre reconnect-sleeptime** [paramètre de `nslcd-configuration`]  
Spécifie le nombre de secondes pendant laquelle attendre lorsque la connexion à tous les serveurs LDAP a échoué. Par défaut, il y a une seconde d'attente entre le premier échec et la tentative suivante.  
La valeur par défaut est 'disabled'.  
**peut-être-nombre reconnect-retrytime** [paramètre de `nslcd-configuration`]  
Spécifie la durée après laquelle le serveur LDAP est considéré comme définitivement inatteignable. Une fois cette durée atteinte, les tentatives de connexions n'auront plus lieu qu'une fois par cet intervalle de temps. La valeur par défaut est de 10 secondes.  
La valeur par défaut est 'disabled'.  
**peut-être-ssl-option ssl** [paramètre de `nslcd-configuration`]  
Spécifie s'il faut utiliser SSL/TLS ou non (la valeur par défaut est non). Si 'start-tls' est spécifié alors StartTLS est utilisé à la place de LDAP sur SSL.  
La valeur par défaut est 'disabled'.  
**peut-être-tls-reqcert-option** [paramètre de `nslcd-configuration`]  
**tls-reqcert**  
Spécifie quelles vérifications effectuer sur les certificats donnés par les serveurs. La signification des valeurs est décrite dans la page de manuel de `ldap.conf(5)`.  
La valeur par défaut est 'disabled'.  
**peut-être-chaine tls-cacertdir** [paramètre de `nslcd-configuration`]  
Spécifie le répertoire contenant les certificats X.509 pour l'authentification des pairs.  
Ce paramètre est ignoré quand il est utilisé avec GnuTLS.  
La valeur par défaut est 'disabled'.

**peut-être-chaine** `tls-cacertfile` [paramètre de `nslcd-configuration`]  
Spécifie le chemin des certificats X.509 pour l'authentification des pairs.

La valeur par défaut est 'disabled'.

**peut-être-chaine** `tls-randfile` [paramètre de `nslcd-configuration`]  
Spécifie le chemin d'une source d'entropie. Ce paramètre est ignoré quand il est utilisé avec GnuTLS.

La valeur par défaut est 'disabled'.

**peut-être-chaine** `tls-ciphers` [paramètre de `nslcd-configuration`]  
Spécifie les suites de chiffrements à utiliser pour TLS en tant que chaîne de caractères.

La valeur par défaut est 'disabled'.

**peut-être-chaine** `tls-cert` [paramètre de `nslcd-configuration`]  
Spécifie le chemin vers le fichier contenant le certificat local pour l'authentification TLS du client.

La valeur par défaut est 'disabled'.

**peut-être-chaine** `tls-key` [paramètre de `nslcd-configuration`]  
Spécifie le chemin du fichier contenant la clef privée pour l'authentification TLS du client.

La valeur par défaut est 'disabled'.

**peut-être-nombre** `pagesize` [paramètre de `nslcd-configuration`]  
Indiquez un nombre plus grand que 0 pour demander des résultats paginés au serveur LDAP en accord avec la RFC 2696. La valeur par défaut (0) est de ne pas demander de pagination des résultats.

La valeur par défaut est 'disabled'.

**peut-être-ignore-users-option** [paramètre de `nslcd-configuration`]  
`nss-initgroups-ignoreusers`

Cette option évite les recherches d'appartenance au groupe à travers le LDAP pour les utilisateurs spécifiés. Autrement, la valeur 'all-local peut être utilisée. Avec cette valeur nslcd construit une liste complète des utilisateurs non-LDAP au démarrage.

La valeur par défaut est 'disabled'.

**peut-être-nombre** `nss-min-uid` [paramètre de `nslcd-configuration`]  
Cette option s'assure que les utilisateurs LDAP avec un id utilisateur numérique plus petit que la valeur spécifiée sont ignorés.

La valeur par défaut est 'disabled'.

**peut-être-nombre** `nss-uid-offset` [paramètre de `nslcd-configuration`]  
Cette option spécifie un décalage à ajouter à tous les id utilisateurs numériques LDAP. Cela peut être utile pour éviter des collisions d'id utilisateurs avec des utilisateurs locaux.

La valeur par défaut est 'disabled'.

**peut-être-nombre nss-gid-offset** [paramètre de `nslcd-configuration`]

Cette option spécifie un décalage à ajouter à tous les id de groupe numériques LDAP. Cela peut être utile pour éviter des collisions d'id utilisateurs avec des groupes locaux. La valeur par défaut est 'disabled'.

**peut-être-booléen nss-nested-groups** [paramètre de `nslcd-configuration`]

Si cette option est indiquée, l'attribut de membre de groupe peut pointer vers un autre groupe. Les membres de groupes imbriqués sont aussi renvoyés dans le groupe de haut-niveau et les groupes parents sont renvoyés lorsqu'on recherche un utilisateur spécifique. La valeur par défaut est de ne pas effectuer de recherche supplémentaire sur les groupes imbriqués.

La valeur par défaut est 'disabled'.

**peut-être-booléen** [paramètre de `nslcd-configuration`]

**nss-getgrent-skipmembers**

Si cette option est indiquée, la liste de membres du groupe n'est pas récupérée lorsqu'on cherche un groupe. Les recherches pour trouver les groupes auxquels un utilisateur appartient resteront fonctionnelles donc l'utilisateur obtiendra probablement les bons groupes à la connexion.

La valeur par défaut est 'disabled'.

**peut-être-booléen** [paramètre de `nslcd-configuration`]

**nss-disable-enumeration**

Si cette option est indiquée, les fonctions qui causent le chargement de toutes les entrées d'utilisateur et de groupe depuis le répertoire ne pourront pas le faire. Cela peut grandement diminuer la charge du serveur LDAP dans des situations où il y a beaucoup d'utilisateurs et de groupes. Cette option n'est pas recommandées pour la plupart des configurations.

La valeur par défaut est 'disabled'.

**peut-être-chaîne validnames** [paramètre de `nslcd-configuration`]

Cette option peut être utilisée pour spécifier comment les noms d'utilisateurs et de groupes sont vérifiés sur le système. Ce motif est utilisé pour vérifier tous les noms d'utilisateurs et de groupes qui sont demandés et renvoyés par le LDAP.

La valeur par défaut est 'disabled'.

**peut-être-booléen ignorecase** [paramètre de `nslcd-configuration`]

Cela spécifie s'il faut ou non effectuer des recherches avec une correspondance sensible à la casse. Activer cela pourrait mener à des vulnérabilités de type contournement d'authentification sur le système et introduire des vulnérabilité d'empoisonnement de cache `nscd` qui permettent un déni de service.

La valeur par défaut est 'disabled'.

**peut-être-booléen pam-authc-ppolicy** [paramètre de `nslcd-configuration`]

Cette option spécifie si des contrôles de la politique de mots de passe sont demandés et gérés par le serveur LDAP à l'authentification de l'utilisateur.

La valeur par défaut est 'disabled'.

**peut-être-chaine pam-authc-search** [paramètre de `nslcd-configuration`]

Par défaut `nslcd` effectue une recherche LDAP avec le mot de passe de l'utilisateur après BIND (authentification) pour s'assurer que l'opération BIND a bien réussi. La recherche par défaut est une simple vérification que le DN de l'utilisateur existe. Un filtre de recherche peut être spécifié pour l'utiliser à la place. Il devrait renvoyer au moins une entrée.

La valeur par défaut est `'disabled'`.

**peut-être-chaine pam-authz-search** [paramètre de `nslcd-configuration`]

Cette option permet la configuration fine et flexible de la vérification d'autorisation qui devrait être effectuée. Le filtre de recherche est exécuté et si une entrée correspond, l'accès est autorisé, sinon il est refusé.

La valeur par défaut est `'disabled'`.

**peut-être-chaine pam-password-prohibit-message** [paramètre de `nslcd-configuration`]

Si cette option est indiquée, la modification de mot de passe par `pam_ldap` sera refusée et le message spécifié sera présenté à l'utilisateur à la place. Le message peut être utilisé pour rediriger les utilisateurs vers une autre méthode pour changer leur mot de passe.

La valeur par défaut est `'disabled'`.

**list pam-services** [paramètre de `nslcd-configuration`]

Liste de noms de service pam pour lesquels l'authentification LDAP devrait suffire.

La valeur par défaut est `'()'`.

### 12.9.19 Services web

Le module (`gnu services web`) fournit le serveur Apache HTTP, le serveur web `nginx` et aussi un démon `fastcgi`.

#### Serveur Apache HTTP

**httpd-service-type** [Variable Scheme]

Type de service pour le serveur Apache HTTP (<https://httpd.apache.org/>) (`httpd`). La valeur de ce type de service est un enregistrement `httpd-configuration`.

Un exemple de configuration simple est donné ci-dessous.

```
(service httpd-service-type
  (httpd-configuration
    (config
      (httpd-config-file
        (server-name "www.example.com")
        (document-root "/srv/http/www.example.com")))))
```

D'autres services peuvent aussi étendre `httpd-service-type` pour être ajouté à la configuration.

```
(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")))))
```

Les détails des types d'enregistrement `httpd-configuration`, `httpd-module`, `httpd-config-file` et `httpd-virtualhost` sont donnés plus bas.

**httpd-configuration** [Type de données]

Ce type de données représente la configuration du service `httpd`.

**package** (par défaut : `httpd`)

Le paquet `httpd` à utiliser.

**pid-file** (par défaut : `"/var/run/httpd"`)

Le fichier de pid utilisé par le service `shepherd`.

**config** (par défaut : `(httpd-config-file)`)

Le fichier de configuration à utiliser avec le service `httpd`. La valeur par défaut est un enregistrement `httpd-config-file` mais cela peut aussi être un G-expression qui génère un fichier, par exemple un `plain-file`. Un fichier en dehors du dépôt peut aussi être spécifié avec une chaîne de caractères.

**httpd-module** [Type de données]

Ce type de données représente un module pour le service `httpd`.

**name** Le nom du module.

**file** Le fichier pour le module. Cela peut être relatif au paquet `httpd` utilisé, l'emplacement absolu d'un fichier ou une G-expression pour un fichier dans le dépôt, par exemple `(file-append mod-wsgi "/modules/mod_wsgi.so")`.

**%default-httpd-modules** [Variable Scheme]

Une liste par défaut des objets `httpd-module`.

**httpd-config-file** [Type de données]

Ce type de données représente un fichier de configuration pour le service `httpd`.

**modules** (par défaut : `%default-httpd-modules`)

Les modules à charger. Les modules supplémentaires peuvent être ajoutés ici ou chargés par des configuration supplémentaires.

Par exemple, pour gérer les requêtes pour des fichiers PHP, vous pouvez utiliser le module `mod_proxy_fcgi` d'Apache avec `php-fpm-service-type` :

```
(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** (par défaut : `httpd`)

Le `ServerRoot` dans le fichier de configuration, par défaut le paquet `httpd`. Les directives comme `Include` et `LoadModule` sont prises relativement à la racine du serveur.

**server-name** (par défaut : `#f`)

Le `ServerName` dans le fichier de configuration, utilisé pour spécifier le schéma de requête, le nom d'hôte et le port que le serveur utilise pour s'identifier.

Cela n'a pas besoin d'être dans la configuration du serveur, et peut être spécifié dans les hôtes virtuels. La valeur par défaut est `#f` pour ne pas spécifier de `ServerName`.

**document-root** (par défaut : `"/srv/http"`)

Le `DocumentRoot` depuis lequel les fichiers seront servis.

**listen** (par défaut : `'("80")`)

La liste des valeurs pour les directives `Listen` dans le fichier de configuration. La valeur devrait être une liste de chaînes, où chacune spécifie le port sur lequel écouter et éventuellement une adresse IP et un protocole à utiliser.

**pid-file** (par défaut : `"/var/run/httpd"`)

Le `PidFile` à utiliser. Cela devrait correspondre à `pid-file` indiqué dans `httpd-configuration` pour que le service `Shepherd` soit correctement configuré.

**error-log** (par défaut : `"/var/log/httpd/error_log"`)

Le `ErrorLog` où le serveur écrit les journaux d'erreurs.

**user** (par défaut : `"httpd"`)

Le `User` en tant que lequel le serveur répondra aux requêtes.

**group** (par défaut : `"httpd"`)

Le `Group` que le serveur utilisera pour répondre aux requêtes.

**extra-config** (par défaut : (list "TypesConfig etc/httpd/mime.types"))  
 Une liste plate de chaînes et de G-expressions qui seront ajoutées à la fin du fichier de configuration.  
 N'importe quelle valeur avec laquelle le service est étendu sera ajouté à cette liste.

**httpd-virtualhost** [Type de données]  
 Ce type de données représente la configuration d'un hôte virtuel pour le service httpd. Ils devraient être ajoutés à extra-config dans httpd-service.

```
(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**  
 L'adresse et le port pour la directive VirtualHost.

**contents** Le contenu de la directive VirtualHost, cela devrait être une liste de chaîne et de G-expressions.

## NGINX

**nginx-service-type** [Variable Scheme]  
 Type de service pour le serveur web NGinx (<https://nginx.org/>). La valeur de ce service est un enregistrement <nginx-configuration>.

Un exemple de configuration simple est donné ci-dessous.

```
(service nginx-service-type
  (nginx-configuration
    (server-blocks
      (list (nginx-server-configuration
        (server-name '("www.example.com"))
        (root "/srv/http/www.example.com")))))
```

En plus d'ajouter des blocs de serveurs dans la configuration du service directement, ce service peut être étendu par d'autres services pour ajouter des blocs de serveurs, comme dans cet exemple :

```
(simple-service 'my-extra-server nginx-service-type
  (list (nginx-server-configuration
    (root "/srv/http/extra-website")
    (try-files (list "$uri" "$uri/index.html")))))
```

Au démarrage, **nginx** n'a pas encore lu son fichier de configuration, donc il utilise les fichiers par défaut pour les messages d'erreur. S'il échoue à charger sa configuration, c'est là où les messages seront enregistrés. Après la lecture du fichier de configuration, le fichier de journal d'erreur par défaut change en fonction de celle-ci. Dans notre cas, les messages

d'erreur au démarrage se trouvent dans `/var/run/nginx/logs/error.log` et après la configuration dans `/var/log/nginx/error.log`. Ce second emplacement peut être modifié avec l'option de configuration `log-directory`.

**nginx-configuration** [Type de données]

Ce type de données représente la configuration de NGInx. Certaines configurations peuvent se faire ici et d'autres fournissent des types d'enregistrement ou éventuellement, on peut fournir un fichier de configuration.

**nginx** (par défaut : `nginx`)

Le paquet nginx à utiliser.

**rshepherd-requirement** (par défaut : `'()`)

C'est une liste de symboles nommant les services du Shepherd dont les services nginx dépendent.

C'est utile si vous voulez que `nginx` démarre après qu'un serveur web en arrière-plan ou qu'un service de journalisation comme Anonip a été démarré.

**log-directory** (par défaut : `"/var/log/nginx"`)

Le répertoire dans lequel NGInx écrira ses fichiers journaux.

**run-directory** (par défaut : `"/var/run/nginx"`)

Le répertoire dans lequel NGInx créera un fichier de pid et écrira des fichiers temporaires.

**server-blocks** (par défaut : `'()`)

Une liste de *blocs serveur* à créer dans le fichier de configuration généré, dont les éléments sont de type `<nginx-server-configuration>`.

L'exemple suivant paramètre NGInx pour servir `www.example.com` depuis le répertoire `/srv/http/www.example.com` sans utiliser HTTPS.

```
(service nginx-service-type
  (nginx-configuration
    (server-blocks
      (list (nginx-server-configuration
            (server-name '("www.example.com"))
            (root "/srv/http/www.example.com"))))))
```

**upstream-blocks** (par défaut : `'()`)

Une liste de *blocs amont* à créer dans le fichier de configuration généré, dont les éléments sont de type `<nginx-upstream-configuration>`.

Configurer les serveurs amont à travers les `upstream-blocks` peut être utile en combinaison avec `locations` dans les enregistrements `<nginx-server-configuration>`. L'exemple suivant crée une configuration de serveur avec une configuration « location » qui sera mandataire pour une configuration amont, qui gèrera les requêtes avec deux serveurs.

```
(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` (par défaut : `#f`)

Si un fichier de configuration *file* est fourni, il sera utilisé au lieu de générer un fichier de configuration à partir des `log-directory`, `run-directory`, `server-blocks` et `upstream-blocks` fournis. Pour un bon fonctionnement, ces arguments devraient correspondre à ce qui se trouve dans *file* pour s'assurer que les répertoires sont créés lorsque le service est activé.

Cela peut être utile si vous avez déjà un fichier de configuration existant ou s'il n'est pas possible de faire ce dont vous avez besoin avec les autres parties de l'enregistrement `nginx-configuration`.

`server-names-hash-bucket-size` (par défaut : `#f`)

Taille du seau pour les tables de hachage des noms de serveurs, par défaut `#f` pour utiliser la taille des lignes de cache du processeur.

`server-names-hash-bucket-max-size` (par défaut : `#f`)

Taille maximum des seaux pour les tables de hachage des serveurs de noms.

`modules` (par défaut : `'()`)

Liste de modules dynamiques de `nginx` à charger. Cela devrait être une liste de noms de fichiers de modules chargeables, comme dans cet exemple :

```
(modules
  (list
    (file-append nginx-accept-language-module "\
/etc/nginx/modules/nginx_http_accept_language_module.so")
    (file-append nginx-lua-module "\
/etc/nginx/modules/nginx_http_lua_module.so")))
```

`lua-package-path` (par défaut : `'()`)

Liste des paquets `lua` à charger. Cela devrait être une liste de noms de paquets de modules `lua` chargeables, comme dans cet exemple :

```
(lua-package-path (list lua-resty-core
```

```
lua-resty-lrucache
lua-resty-signal
lua-tablepool
lua-resty-shell))
```

`lua-package-cpath` (par défaut : '()')

Liste de paquets lua C à charger. cela devrait être une liste de noms de paquets de modules lua C, comme dans cet exemple :

```
(lua-package-cpath (list lua-resty-signal))
```

`global-directives` (par défaut : '((events . ())))

Liste d'association des directives globales pour le plus haut niveau de la configuration de nginx. Les valeurs peuvent elles-mêmes être des listes d'association.

```
(global-directives
  ~(worker_processes . 16)
  (pcre_jit . on)
  (events . ((worker_connections . 1024))))
```

`extra-content` (par défaut : "")

Contenu supplémentaire du bloc `http`. Cela devrait être une chaîne ou un G-expression.

`nginx-server-configuration` [Type de données]

Type de données représentant la configuration d'un bloc serveur de nginx. Ce type a les paramètres suivants :

`listen` (par défaut : '("80" "443 ssl")')

Chaque directive `listen` indique l'adresse et le port pour le protocole IP ou le chemin d'un socket UNIX-domain sur lequel le serveur acceptera les connexions. On peut spécifier l'adresse et le port, ou juste l'adresse ou juste le port. Une adresse peut aussi être un nom d'hôte, par exemple :

```
'("127.0.0.1:8000" "127.0.0.1" "8000" " *:8000" "localhost:8000")■
```

`server-name` (par défaut : (list 'default))

Une liste de noms de serveurs que ce serveur représente. `'default'` représente le serveur par défaut pour les connexions qui ne correspondent à aucun autre serveur.

`root` (par défaut : "/srv/http")

Racine du site web que sert nginx.

`locations` (par défaut : '()')

Une liste d'enregistrements *nginx-location-configuration* ou *nginx-named-location-configuration* à utiliser dans ce bloc serveur.

`index` (par défaut : (list "index.html"))

Fichiers d'index à chercher lorsque les clients demandent un répertoire. S'il ne peut pas être trouvé, Nginx enverra la liste des fichiers dans le répertoire.

**try-files** (par défaut : '()')  
 Une liste de fichiers dont l'existence doit être vérifiée dans l'ordre spécifié. **nginx** utilisera le premier fichier trouvé pour satisfaire la requête.

**ssl-certificate** (par défaut : #f)  
 Où trouver les certificats pour les connexions sécurisées. Indiquez #f si vous n'avez pas de certificats et que vous ne voulez pas utiliser HTTPS.

**ssl-certificate-key** (par défaut : #f)  
 Où trouver la clef privée pour les connexions sécurisées. Indiquez #f si vous n'avez pas de clef et que vous ne voulez pas utiliser HTTPS.

**server-tokens?** (par défaut : #f)  
 Indique si le serveur devrait ajouter sa configuration dans les réponses.

**raw-content** (par défaut : '()')  
 Une liste de lignes brutes à ajouter dans le bloc serveur.

**nginx-upstream-configuration** [Type de données]  
 Type de données représentant la configuration d'un bloc **upstream** **nginx**. Ce type a les paramètres suivants :

**name** Noms de ces groupes de serveurs.

**serveurs** Spécifie les adresses des serveurs dans le groupe. L'adresse peut être spécifiée avec une adresse IP (p. ex. '127.0.0.1'), un nom de domaine (p. ex. 'backend1.example.com') ou un chemin vers un socket UNIX avec le préfixe 'unix:'. Pour les adresses utilisant une adresse IP ou un nom de domaine, le port par défaut est 80 et un port différent peut être spécifié explicitement.

**extra-content**  
 Une chaîne ou une liste de chaînes à ajouter au bloc amont.

**nginx-location-configuration** [Type de données]  
 Type de données représentant la configuration d'un bloc **location** **nginx**. Ce type a les paramètres suivants :

**uri** URI qui correspond à ce bloc.

**body** Corps du bloc **location**, spécifié comme une liste de chaînes de caractères. Cela peut contenir de nombreuses directives de configuration. Par exemple, pour passer des requêtes à un groupe de serveurs amont définis dans un bloc **nginx-upstream-configuration**, la directive suivante peut être spécifiée dans le corps : '(list "proxy\_pass http://upstream-name;")'.

**nginx-named-location-configuration** [Type de données]  
 Type de données représentant la configuration d'un bloc **location** **nginx** nommé. Les blocs **location** nommés sont utilisés pour les redirections de requêtes et pas pour le traitement des requêtes normales. Ce type a les paramètres suivants :

**name** Nom pour identifier ce bloc **location**.

**body** Voir [nginx-location-configuration body], page 448, comme le corps d'un bloc location nommé peut être utilisé de la même manière que `nginx-location-configuration body`. Une restriction est que le corps d'un bloc location nommé ne peut pas contenir de bloc location.

## Cache Varnish

Varnish est un serveur de cache rapide qui se trouve entre les applications web et les utilisateurs. Il sert de serveur mandataire pour les requêtes des clients et met les URL accédées en cache pour que plusieurs requêtes à la même ressource ne crée qu'une requête au moteur.

**varnish-service-type** [Variable Scheme]

Type de service pour le démon Varnish.

**varnish-configuration** [Type de données]

Type de données représentant la configuration du service `varnish`. Ce type a les paramètres suivants :

**package** (par défaut : `varnish`)

Le paquet Varnish à utiliser.

**name** (par défaut : `"default"`)

Un nom pour cet instance de Varnish. Varnish va créer un répertoire dans `/var/varnish/` avec ce nom et gardera des fichiers temporaires à cet endroit. Si le nom commence par une barre oblique, il est interprété comme un nom de répertoire absolu.

Passez l'argument `-n` aux autres programmes Varnish pour vous connecter à l'instance nommée, p. ex. `varnishncsa -n default`.

**backend** (par défaut : `"localhost:8080"`)

Le moteur à utiliser. Cette option n'a pas d'effet si `vcl` est vrai.

**vcl** (par défaut : `#f`)

Le programme *VCL* (Varnish Configuration Language) à lancer. Si la valeur est `#f`, Varnish servira de mandataire pour `backend` avec la configuration par défaut. Sinon, ce doit être un objet simili-fichier avec une syntaxe VCL valide.

Par exemple, pour créer un miroir de `www.gnu.org` (`https://www.gnu.org`) avec VCL vous pouvez faire quelque chose comme cela :

```
(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)))
```

On peut inspecter la configuration d'une instance Varnish actuellement lancée en utilisant le programme `varnishadm`.

Consultez le guide utilisateur de varnish (<https://varnish-cache.org/docs/>) et le livre varnish (<https://book.varnish-software.com/4.0/>) pour une documentation complète sur Varnish et son langage de configuration.

```
listen (par défaut : '("localhost:80"))
```

Liste des adresses sur lesquelles écoute Varnish.

```
storage (par défaut : '("malloc,128m"))
```

Liste de moteurs de stockage qui seront disponibles en VCL.

```
parameters (par défaut : '())
```

Liste des paramètres à l'exécution de la forme '(("parameter" . "value"))'.

```
extra-options (par défaut : '())
```

Arguments supplémentaires à passer au processus `varnishd`.

## Patchwork

Patchwork est un système de suivi de correctifs. Il peut récupérer des correctifs envoyés à une liste de diffusion, et les afficher sur une interface web.

```
patchwork-service-type
```

[Variable Scheme]

Type de service pour Patchwork.

L'exemple suivant est un exemple de service Patchwork minimal, pour le domaine `patchwork.example.com`.

```
(service patchwork-service-type
  (patchwork-configuration
    (domain "patchwork.example.com")
    (settings-module
      (patchwork-settings-module
        (allowed-hosts (list domain))
        (default-from-email "patchwork@patchwork.example.com"))))
  (getmail-retriever-config
    (getmail-retriever-configuration
      (type "SimpleIMAPSSLRetriever")
      (server "imap.example.com")
      (port 993)
      (username "patchwork")
      (password-command
        (list (file-append coreutils "/bin/cat")
              "/etc/getmail-patchwork-imap-password"))
      (extra-parameters
        '((mailboxes . ("Patches"))))))))
```

Il y a trois enregistrements pour configurer le service Patchwork. Le `<patchwork-configuration>` correspond à la configuration de Patchwork dans le service HTTPD.

Le champ `settings-module` dans l'enregistrement `<patchwork-configuration>` peut être rempli avec l'enregistrement `<patchwork-settings-modules>`, qui décrit un module de paramètres généré dans le dépôt de Guix.

Pour le champ `database-configuration` dans `<patchwork-settings-module>`, il faut utiliser un `<patchwork-dataase-configuration>`.

**patchwork-configuration** [Type de données]

Type de données représentant la configuration du service Patchwork. Ce type a les paramètres suivants :

**patchwork** (par défaut : `patchwork`)

Le paquet Patchwork à utiliser.

**domain** Le domaine à utiliser pour Patchwork, utilisé comme hôte virtuel dans le service HTTPD.

**settings-module**

Le module de paramètres à utiliser pour Patchwork. En tant qu'application Django, Patchwork est configuré avec un module Python contenant les paramètres. Il peut s'agir d'une instance de l'enregistrement `<patchwork-settings-module>`, de n'importe quel enregistrement qui représente les paramètres dans le dépôt, ou un répertoire en dehors du dépôt.

**static-path** (par défaut : `"/static/"`)

Le chemin dans lequel le service HTTPD devrait servir les fichiers statiques.

**getmail-retriever-config**

La valeur d'enregistrement `getmail-retriever-configuration` à utiliser avec Patchwork. `getmail` sera configuré avec cette valeur, les messages seront livrés à Patchwork.

**patchwork-settings-module** [Type de données]

Type de données représentant le module de paramètres de Patchwork. Certains de ces paramètres sont liés à Patchwork, mais d'autres sont liés à Django, le cadriciel web utilisé par Patchwork, ou la bibliothèque Django Rest Framework. Ce type a les paramètres suivants :

**database-configuration** (par défaut : `(patchwork-database-configuration)`)

Les paramètres de connexion à la base de données à utiliser pour Patchwork. Voir le type d'enregistrement `<patchwork-database-configuration>` pour plus d'information.

**secret-key-file** (par défaut : `"/etc/patchwork/django-secret-key"`)

Patchwork, en tant qu'application web Django, utilise une clé secrète pour signer des valeurs avec de la cryptographie. Ce fichier devrait contenir une valeur unique imprévisible.

Si ce fichier n'existe pas, il sera créé avec une valeur aléatoire par le service patchwork-setup.

Ce paramètre est lié à Django.

**allowed-hosts**

Une liste des hôtes valides pour ce service Patchwork. Cela doit au minimum inclure le domaine spécifié dans l'enregistrement `<patchwork-configuration>`.

C'est un paramètre pour Django.

**default-from-email**

L'adresse de courriel sur laquelle Patchwork devrait envoyer les courriels par défaut.

C'est un paramètre de Patchwork.

**static-url** (par défaut : `#f`)

L'URL à utiliser pour servir les ressources statiques. Cela peut être un bout d'URL ou une URL complète, mais doit finir par `/`.

Si la valeur par défaut est utilisée, la valeur `static-path` de `<patchwork-configuration>` sera utilisée.

C'est un paramètre pour Django.

**admins** (par défaut : `'()'`)

Adresses de courriel auxquelles envoyer le détails des erreurs s'il y en a. Chaque valeur de la liste contient deux éléments, le nom et l'adresse de courriel.

C'est un paramètre pour Django.

**debug?** (par défaut : `#f`)

Indique s'il faut lancer Patchwork en mode de débogage. Si la valeur est `#t`, les messages d'erreur détaillés seront affichés.

C'est un paramètre pour Django.

**enable-rest-api?** (par défaut : `#t`)

Indique s'il faut activer l'API REST de Patchwork.

C'est un paramètre de Patchwork.

**enable-xmlrpc?** (par défaut : `#t`)

Indique s'il faut activer l'API RPC en XML.

C'est un paramètre de Patchwork.

**force-https-links?** (par défaut : `#t`)

Indique s'il faut utiliser des liens HTTPS sur les pages de Patchwork.

C'est un paramètre de Patchwork.

**extra-settings** (par défaut : `""`)

Code supplémentaire à placer à la fin du module de paramètres de Patchwork.

**patchwork-database-configuration** [Type de données]

Type de données représentant la configuration de la base de données de Patchwork.

**engine** (par défaut : "django.db.backends.postgresql\_psycopg2")

Le moteur de base de données à utiliser.

**name** (par défaut : "patchwork")

Le nom de la base de données à utiliser.

**user** (par défaut : "httpd")

L'utilisateur à utiliser pour se connecter à la base de données.

**password** (par défaut : "")

Le mot de passe à utiliser lors de la connexion à la base de données.

**host** (par défaut : "")

L'hôte sur lequel se connecter à la base de données.

**port** (par défaut : "")

Le port sur lequel se connecter à la base de données.

## Mumi

Mumi (<https://git.elephly.net/gitweb.cgi?p=software/mumi.git>) est une interface Web pour le logiciel de suivi de bogues Debbugs, par défaut pour l'instance de GNU (<https://bugs.gnu.org>). Mumi est un serveur Web, mais il récupère et indexe aussi les courriels provenant de Debbugs.

**mumi-service-type** [Variable Scheme]

Le type de service pour Mumi.

**mumi-configuration** [Type de données]

Type de données représentant la configuration du service Mumi. Ce type a les champs suivants :

**mumi** (par défaut : `mumi`)

Le paquet Mumi à utiliser.

**mailer?** (par défaut : `#true`)

Indique s'il faut activer le composant d'envoi de courriels.

**mumi-configuration-sender**

L'adresse de courriel utilisée comme expéditeur pour les commentaires.

**mumi-configuration-smtp**

Une URI pour configurer les paramètres SMTP pour Mailutils. Cela peut être quelque chose comme `sendmail:///path/to/bin/msmtp` ou toute autre URI prise en charge par Mailutils. Voir Section "SMTP Mailboxes" dans *GNU Mailutils*.

## FastCGI

FastCGI est une interface entre le frontal et le moteur d'un service web. C'est un dispositif quelque peu désuet ; les nouveaux services devraient généralement juste parler HTTP entre le frontal et le moteur. Cependant il y a un certain nombre de services de moteurs comme



PHP ou l'accès aux dépôts Git optimisé en HTTP qui utilisent FastCGI, donc nous le supportons dans Guix.

Pour utiliser FastCGI, vous configurez le serveur web frontal (p. ex. nginx) pour envoyer un sous-ensemble de ses requêtes au moteur fastcgi, qui écoute sur un socket UNIX ou TCP local. Il y a un programme `fcgiwrap` intermédiaire qui se trouve entre le processus du moteur et le serveur web. Le frontal indique quel moteur lancer, en passant cette information au processus `fcgiwrap`.

`fcgiwrap-service-type` [Variable Scheme]

Un type de service pour le mandataire FastCGI `fcgiwrap`.

`fcgiwrap-configuration` [Type de données]

Type de données représentant la configuration du service `fcgiwrap`. Ce type a les paramètres suivants :

`package` (par défaut : `fcgiwrap`)

Le paquet `fcgiwrap` à utiliser.

`socket` (par défaut : `tcp:127.0.0.1:9000`)

Le socket sur lequel le processus `fcgiwrap` écoute, en tant que chaîne de caractères. Les valeurs valides de `socket` sont `unix:/path/to/unix/socket`, `tcp:dot.ted.qu.ad:port` et `tcp6:[ipv6_addr]:port`.

`user` (par défaut : `fcgiwrap`)

`group` (par défaut : `fcgiwrap`)

Les noms de l'utilisateur et du groupe, en tant que chaînes de caractères, sous lesquels lancer le processus `fcgiwrap`. Le service `fastcgi` s'assurera que si l'utilisateur demande les noms d'utilisateurs et de groupes `fcgiwrap` l'utilisateur et le groupe correspondant seront présents sur le système.

Il est possible de configurer un service web soutenu par FastCGI pour passer les informations d'authentification HTTP depuis le frontal jusqu'au moteur, et de permettre à `fcgiwrap` dans lancer le processus de moteur avec l'utilisateur local correspondant. Pour activer cette fonctionnalité sur le moteur, lancez `fcgiwrap` en tant qu'utilisateur et groupe `root`. Remarquez que cette fonctionnalité doit aussi être configurée sur le frontal.

## PHP-FPM

PHP-FPM (FastCGI Process Manager) est une implémentation FastCGI de PHP alternative avec quelques fonctionnalités supplémentaires utiles pour les sites de toutes tailles.

Ces fonctionnalités comprennent :

- La création de processus adaptative
- Des statistiques de base (comme le `mod_status` d'Apache)
- La gestion des processus avancée avec arrêt et démarrage sans heurts
- La possibilité de démarrer des processus de travail avec différents `uid/gid/chroot/environnement` et différents `php.ini` (à la place de `safe_mode`)

- L'enregistrement des journaux sur stdout et stderr
- Le redémarrage d'urgence dans le cas de la destruction accidentelle du cache des opcodes
- Le support des téléversements accélérés
- Le support de « showlog »
- Des améliorations à FastCGI, comme `fastcgi_finish_request()` - une fonction spéciale pour terminer la requête et nettoyer toutes les données tout en continuant à faire d'autres choses qui prennent du temps (conversion vidéo, gestion des stats, etc.).

... et bien plus.

`php-fpm-service-type` [Variable Scheme]

Un type de service pour `php-fpm`.

`php-fpm-configuration` [Type de données]

Type de données pour la configuration du service `php-fpm`.

`php` (par défaut : `php`)

Le paquet `php` à utiliser.

`socket` (par défaut : `(string-append "/var/run/php" (version-major (package-version php)) "-fpm.sock")`)

L'adresse sur laquelle accepter les requêtes FastCGI. Les syntaxes valides sont :

`"ip.add.re.ss:port"`

Écoute sur un socket TCP sur l'adresse spécifiée sur un port spécifié.

`"port"` Écoute sur un socket TCP sur toutes les adresse sur un port spécifique.

`"/path/to/unix/socket"`

Écoute sur un socket unix.

`user` (par défaut : `php-fpm`)

Utilisateur à qui appartiendra le processus de travail de `php`.

`group` (par défaut : `php-fpm`)

Groupe du processus de travail.

`socket-user` (par défaut : `php-fpm`)

Utilisateur qui peut parler au socket `php-fpm`.

`socket-group` (par défaut : `nginx`)

Groupe qui peut parler au socket `php-fpm`.

`pid-file` (par défaut : `(string-append "/var/run/php" (version-major (package-version php)) "-fpm.pid")`)

Le pid de `php-fpm` est écrit dans ce fichier une fois que le service a démarré.

`log-file` (par défaut : `(string-append "/var/log/php" (version-major (package-version php)) "-fpm.log")`)

Fichier de journal pour le processus maître de `php-fpm`.

`process-manager` (par défaut :

`(php-fpm-dynamic-process-manager-configuration)`)

Configuration détaillée pour le gestionnaire de processus de php-fpm. Il doit s'agir soit de :

```
<php-fpm-dynamic-process-manager-configuration>
<php-fpm-static-process-manager-configuration> ou
<php-fpm-on-demand-process-manager-configuration>
```

`display-errors` (par défaut : `#f`)

Détermine si les erreurs et les avertissements php doivent être envoyés aux clients et affichés dans leur navigateur. Cela est utile pour un développement php local, mais un risque pour la sécurité pour les sites publics, comme les messages d'erreur peuvent révéler des mots de passes et des données personnelles.

`timezone` (par défaut : `#f`)

Spécifie le paramètre `php_admin_value[date.timezone]`.

`workers-logfile` (par défaut : `(string-append "/var/log/php" (version-major (package-version php)) "-fpm.www.log")`)

Ce fichier enregistrera la sortie `stderr` des processus de travail de php. On peut indiquer `#f` pour désactiver la journalisation.

`file` (par défaut : `#f`)

Une version alternative de la configuration complète. Vous pouvez utiliser la fonction `mixed-text-file` ou un chemin de fichier absolu.

`php-ini-file` (par défaut : `#f`)

Une version alternative de la configuration de php par défaut. Il peut s'agir de n'importe quel objet « simili-fichier » (voir Section 9.12 [G-Expressions], page 166). Vous pouvez utiliser la fonction `mixed-text-file` ou un chemin de fichier absolu.

Pour le développement local il est utile d'indiquer un long délai d'attente et une limite mémoire plus importante pour les processus php créés. Cela se fait avec les bouts de configuration systèmes suivants :

```
(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)))
```

Consultez les principales directives de `php.ini` (<https://www.php.net/manual/en/ini.core.php>) pour une documentation complète sur les directives `php.ini` possibles.



```

        (listen '("80"))
        (ssl-certificate #f)
        (ssl-certificate-key #f)))
    %base-services))

```

Le générateur d'avatar de chat est un simple service pour démontrer l'utilisation de php-fpm dans Nginx. Il permet de générer des avatars de chats à partir d'une graine, par exemple le hash de l'adresse de courriel d'un utilisateur.

```

cat-avatar-generator-service [#:cache-dir [Procédure Scheme]
    "/var/cache/cat-avatar-generator"] [#:package cat-avatar-generator]
    [#:configuration (nginx-server-configuration)]

```

Renvoie un nginx-server-configuration qui hérite de configuration. Il étend la configuration nginx pour ajouter un bloc de serveur qui sert package, une version de cat-avatar-generator. Pendant l'exécution, cat-avatar-generator pourra utiliser cache-dir comme répertoire de cache.

Une configuration simple de cat-avatar-generator ressemble à ceci :

```

    (services (cons* (cat-avatar-generator-service
        #:configuration
        (nginx-server-configuration
            (server-name '("example.com")))))
        ...
    %base-services))

```

## Hpcguix-web

Le programme hpcguix-web (<https://github.com/UMCUGenetics/hpcguix-web/>) est une interface web personnalisable pour naviguer dans les paquets Guix, initialement conçue pour les utilisateurs des grappes de calcul de haute performance (HPC).

```

hpcguix-web-service-type [Variable Scheme]
    Le type de service pour hpcguix-web.

```

```

hpcguix-web-configuration [Type de données]
    Type de données pour la configuration du service hpcguix-web.

```

**specs** Une gexp (voir Section 9.12 [G-Expressions], page 166) spécifiant la configuration du service hpcguix-web. Les éléments principaux disponibles dans cette spec sont :

```

title-prefix (par défaut : "hpcguix | ")
    Le préfixe du titre des pages.

```

```

guix-command (par défaut : "guix")
    La commande guix.

```

```

package-filter-proc (par défaut : (const #t))
    Une procédure qui spécifie comment filtrer les paquets qui
    seront affichés.

```

```

package-page-extension-proc (par défaut : (const '()))
    Paquet d'extensions pour hpcguix-web.

```

`menu` (par défaut : '() )  
 Entrée supplémentaire dans la page `menu`.

`channels` (par défaut : `%default-channels`)  
 Liste des canaux depuis lesquels la liste des paquets est construite (voir Chapitre 7 [Canaux], page 75).

`package-list-expiration` (par défaut : `(* 12 3600)`)  
 Le temps d'expiration, en secondes, après lequel la liste des paquets est reconstruite depuis les dernières instance des canaux donnés.

Voir le dépôt `hpcguix-web` pour un exemple complet (<https://github.com/UMCUGenetics/hpcguix-web/blob/master/hpcweb-configuration.scm>).

`package` (par défaut : `hpcguix-web`)  
 Le paquet `hpcguix-web` à utiliser.

`address` (par défaut : `"127.0.0.1"`)  
 L'adresse IP sur laquelle écouter.

`port` (par défaut : `5000`)  
 Le numéro de port sur lequel écouter.

Une déclaration de service `hpcguix-web` typique ressemble à cela :

```
(service hpcguix-web-service-type
  (hpcguix-web-configuration
    (specs
      #~(define site-config
        (hpcweb-configuration
          (title-prefix "Guix-HPC - ")
          (menu '(("/about" "ABOUT"))))))))
```

**Remarque:** Le service `hpcguix-web` met régulièrement à jour la liste des paquets qu'il publie en récupérant les canaux depuis Git. Pour cela, il doit accéder aux certificats X.509 pour qu'il puisse authentifier les serveurs Git quand il communique en HTTPS, et il suppose que `/etc/ssl/certs` contient ces certificats.

Ainsi, assurez-vous d'ajouter `nss-certs` ou un autre paquet de certificats dans le champ `packages` de votre configuration. Section 12.11 [Certificats X.509], page 572, pour plus d'informations sur les certificats X.509.

## gmnisrv

Le programme `gmnisrv` (<https://git.sr.ht/~sircmpwn/gmnisrv>) est un serveur simple pour le protocole Gemini (<https://gemini.circumlunar.space/>).

`gmnisrv-service-type` [Variable Scheme]

C'est le type du service `gmnisrv`, dont la valeur devrait être un objet `gmnisrv-configuration` comme dans cet exemple :

```
(service gmnisrv-service-type
  (gmnisrv-configuration
    (config-file (local-file "./my-gmnisrv.ini"))))
```

**gmnisrv-configuration** [Type de données]

Type données qui représente la configuration de gmnisrv.

**package** (par défaut : *gmnisrv*)

Objet de paquet du serveur gmnisrv.

**config-file** (par défaut : *%default-gmnisrv-config-file*)

Objet simili-fichier du fichier de configuration de gmnisrv à utiliser. La configuration par défaut écoute sur le port 1965 et sert les fichiers de */srv/gemini*. Les certificats sont stockés dans */var/lib/gemini/certs*. Pour plus d'informations, lancez `man gmnisrv` et `man gmnisrv.ini`.

## Agate

Le programme Agate (`gemini://qwertyqwefsdays.eu/agate.gmi`) (page GitHub sur HTTPS (`https://github.com/mbrubeck/agate`)) est un serveur simple pour le protocole Gemini (`https://gemini.circumlunar.space/`) écrit en Rust.

**agate-service-type** [Variable Scheme]

C'est le type du service agate, dont la valeur devrait être un objet `agate-service-type` comme dans cet exemple :

```
(service agate-service-type
  (agate-configuration
    (content "/srv/gemini")
    (cert "/srv/cert.pem")
    (key "/srv/key.rsa")))
```

L'exemple ci-dessus représente les changements minimaux nécessaires pour qu'Agate soit lancé. Spécifier le chemin du certificat et de la clé est toujours nécessaire, car le protocole Gemini nécessite TLS par défaut.

Pour obtenir un certificat et une clé, vous pouvez par exemple utiliser OpenSSL, en lançant une commande telle que l'exemple suivant :

```
openssl req -x509 -newkey rsa:4096 -keyout key.rsa -out cert.pem \
  -days 3650 -nodes -subj "/CN=example.com"
```

Bien sûr, vous devez remplacer *example.com* par votre propre nom de domaine, puis pointer la configuration d'Agate vers le chemin de la clé et du certificats générés.

**agate-configuration** [Type de données]

Type de données représentant la configuration d'Agate.

**package** (par défaut : *agate*)

Objet de paquet du serveur Agate.

**content** (par défaut : *"/srv/gemini"*)

Le répertoire à partir duquel Agate servira ses fichiers.

**cert** (par défaut : *#f*)

Le chemin vers le certificat TLS en PEM à utiliser pour chiffrer les connexions. Il doit être rempli avec une valeur venant de l'utilisateur.

**key** (par défaut : *#f*)

Le chemin de fichier vers la clef privée PKCS8 à utiliser pour les connexions chiffrées. Elle doit être remplie avec une valeur de l'utilisateur.

- addr** (par défaut : `'("0.0.0.0:1965" "[:]:1965")`)  
Une liste des adresses sur lesquelles écouter.
- hostname** (par défaut : `#f`)  
Le nom de domaine de ce serveur Gemini. Facultatif.
- lang** (par défaut : `#f`)  
Codes de langues RFC 4646 pour les documents text/gemini. Facultatif.
- silent?** (par défaut : `#f`)  
Indiquez `#t` pour désactiver la journalisation de la sortie.
- serve-secret?** (par défaut : `#f`)  
Indiquez `#t` pour servir des fichiers secrets (des fichiers et répertoires commençant par un point).
- log-ip?** (par défaut : `#t`)  
Indique si l'adresse IP de sortie doit être journalisée.
- user** (par défaut : `"agate"`)  
Propriétaire du processus `agate`.
- group** (par défaut : `"agate"`)  
Groupe du propriétaire du processus `agate`.
- log-file** (par défaut : `"/var/log/agate.log"`)  
Le fichier qui devrait contenir la sortie de journal d'Agate.

### 12.9.20 Services de certificats

Le module (`gnu services certbot`) fournit un service qui récupère automatiquement un certificat TLS valide de l'autorité de certification Let's Encrypt. Ces certificats peuvent ensuite être utilisés pour servir du contenu de manière sécurisée sur HTTPS et d'autres protocoles basés sur TLS, en sachant que le client sera capable de vérifier l'authenticité du serveur.

Let's Encrypt (<https://letsencrypt.org/>) fournit l'outil `certbot` pour automatiser le processus de certification. Cet outil génère d'abord un clef sur le serveur de manière sécurisée. Ensuite il demande à l'autorité de certification Let's Encrypt de signer la clef. La CA vérifie que la requête provient de l'hôte en question en utilisant un protocole de défi-réponse, ce qui requiert que le serveur fournisse sa réponse par HTTP. Si ce protocole se passe sans encombre, la CA signe la clef et on obtient un certificat. Ce certificat est valide pour une durée limitée et donc, pour continuer à fournir des services en TLS, le serveur doit régulièrement demander à la CA de renouveler sa signature.

Le service `certbot` automatise ce processus : la génération initiale de la clef, la demande de certification initiale au service Let's Encrypt, l'intégration du protocole de défi/réponse dans le serveur web, l'écriture du certificat sur le disque, les renouvellements périodiques et les tâches de déploiement avec le renouvellement (p. ex. recharger les services, copier les clefs avec d'autres permissions).

Certbot est lancé deux fois par jour, à une minute aléatoire dans l'heure. Il ne fera rien sauf si vos certificats doivent être renouvelés ou sont révoqués, mais le lancer régulièrement permettra à vos services de rester en ligne si Let's Encrypt décide de révoquer votre certificat.



En utilisant ce service, vous acceptez le document « ACME Subscriber Agreement », qu'on peut trouver ici : <https://acme-v01.api.letsencrypt.org/directory>.

**certbot-service-type** [Variable Scheme]

Un type de service pour le client Let's Encrypt `certbot`. Sa valeur doit être un enregistrement `certbot-configuration` comme dans cet exemple :

```
(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 "toto@example.net")
    (certificates
      (list
        (certificate-configuration
          (domains '("example.net" "www.example.net"))
          (deploy-hook %nginx-deploy-hook))
        (certificate-configuration
          (domains '("titi.example.net"))))))))
```

Voir plus bas pour des détails sur `certbot-configuration`.

**certbot-configuration** [Type de données]

Type données représentant la configuration du service `certbot`. Ce type a les paramètres suivants :

**package** (par défaut : `certbot`)

Le paquet `certbot` à utiliser.

**webroot** (par défaut : `/var/www`)

Le répertoire depuis lequel servir les fichiers du défi/réponse de Let's Encrypt.

**certificates** (par défaut : `()`)

Une liste de `certificates-configuration` pour lesquels générer des certificats et demander des signatures. Chaque certificat a un `name` et plusieurs `domains`.

**email** (par défaut : `#f`)

Adresse de courriel facultative pour la création de compte et le contact de récupération de compte. Il est recommandé de l'indiquer car cela vous permet de recevoir des notifications importantes à propos de votre compte et des certificats créés.

**server** (par défaut : `#f`)

URL facultative d'un serveur ACME. Indiquer ce paramètre remplace la valeur de `certbot` par défaut, qui est le serveur de Let's Encrypt.

**rsa-key-size** (par défaut : 2048)  
Taille de la clef RSA.

**default-location** (par défaut : *voir plus bas*)

Le **nginx-location-configuration** par défaut. Comme **certbot** doit pouvoir servir les défis et les réponses, il doit être capable de lancer un serveur web. Cela se fait en étendant le service web **nginx** avec un **nginx-server-configuration** qui écoute sur les *domains* sur le port 80 et qui a un **nginx-location-configuration** pour le chemin `/.well-known/` utilisé par Let's Encrypt. Voir Section 12.9.19 [Services web], page 441, pour plus d'information sur les types de données de la configuration de **nginx**.

Les requêtes vers d'autres URL correspondra à **default-location**, qui, s'il est présent, sera ajout é à tous les **nginx-server-configuration**.

Par défaut, le **default-location** sera une redirection de `http://domain/...` vers `https://domain/...`, en vous laissant définir ce que vous voulez servir sur votre site en **https**.

Passez **#f** pour ne pas utiliser de location par défaut.

**certificate-configuration** [Type de données]

Type de données représentant la configuration d'un certificat. Ce type a les paramètres suivants :

**name** (par défaut : *voir plus bas*)

Ce nom est utilisé par **Certbot** pour ses tâches quotidiennes et dans les chemins de fichiers ; il n'affecte pas le contenu des certificats eux-mêmes. Pour voir les noms des certificats, lancez **certbot certificates**.

Sa valeur par défaut est le premier domaine spécifié.

**domains** (par défaut : ())

Le premier domaine spécifié sera le CN du sujet du certificat, et tous les domaines seront les noms alternatifs du sujet dans le certificat.

**challenge** (par défaut : **#f**)

Le type de défi à utiliser avec **certbot**. Si **#f** est spécifié, le défi HTTP par défaut sera utilisé. Si une valeur est spécifiée, le greffon manuel sera utilisé par défaut (voir **authentication-hook**, **cleanup-hook** et la documentation sur <https://certbot.eff.org/docs/using.html#hooks>), et donne à Let's Encrypt la permission d'enregistrer l'adresse IP publique de la machine qui a fait la demande.

**csr** (par défaut : **#f**)

Nom de fichier d'une Demande de Signature de Certificat (CSR) au format DER ou PEM. Si **#f** est spécifié, cet argument ne sera pas passé à **certbot**. Si une valeur est spécifiée, **certbot** l'utilisera pour obtenir un certificat, au lieu d'utiliser une CSR auto-signée. Le(s) nom(s) de domaine mentionné(s) dans **domains** doivent être en cohérence avec ceux mentionné(s) dans le fichier CSR.

**authentication-hook** (par défaut : #t)

Commande à lancer dans un shell une fois par défi de certificat auquel répondre. Pour cette commande, la variable shell `$CERTBOT_DOMAIN` contiendra le domaine à authentifier, `$CERTBOT_VALIDATION` contiendra la chaîne de validation et `$CERTBOT_TOKEN` contiendra le nom de fichier de la ressource demandée pour le défi HTTP-01.

**cleanup-hook** (par défaut : #f)

Commande à lancer dans un shell une fois par défi de certificat auquel `auth-hook` a répondu. Pour cette commande, les variables shell disponibles dans le script `auth-hook` sont toujours disponibles, et en plus `$CERTBOT_AUTH_OUTPUT` contiendra la sortie standard du script `auth-hook`.

**deploy-hook** (par défaut : #f)

Commande à lancer dans un shell une fois par certificat récupéré avec succès. Pour cette commande, la variable `$RENEWED_LINEAGE` pointera sur le sous-répertoire live (par exemple, `"/etc/letsencrypt/live/example.com"`) contenant le nouveau certificat et la clef ; la variable `$RENEWED_DOMAINS` contiendra les noms de domaines séparés par des espaces (par exemple `"example.com www.example.com"`).

Pour chaque `certificate-configuration`, le certificat est sauvegardé dans `/etc/letsencrypt/live/name/fullchain.pem` et la clef est sauvegardée dans `/etc/letsencrypt/live/name/privkey.pem`.

## 12.9.21 Services DNS

Le module (`gnu services dns`) fournit des services liés au *système de noms de domaines* (DNS). Il fournit un service de serveur pour héberger un serveur DNS *faisant autorité* pour plusieurs zones, en esclave ou en maître. Ce service utilise Knot DNS (<https://www.knot-dns.cz/>). Il fournit aussi un service de cache et de renvoie DNS pour le LAN, qui utilise `dnsmasq` (<http://www.thekelleys.org.uk/dnsmasq/doc.html>).

### Service Knot

Voici un exemple de configuration pour un serveur faisant autorité sur deux zone, un maître et un esclave :

```
(define-zone-entries example.org.zone
;; Name TTL Class Type Data
  ("@" "" "IN" "A" "127.0.0.1")
  ("@" "" "IN" "NS" "ns")
  ("ns" "" "IN" "A" "127.0.0.1"))

(define master-zone
  (knot-zone-configuration
    (domain "example.org")
    (zone (zone-file
```

```

        (origin "example.org")
        (entries example.org.zone))))))

(define slave-zone
  (knot-zone-configuration
    (domain "plop.org")
    (dnssec-policy "default")
    (master (list "plop-master"))))

(define plop-master
  (knot-remote-configuration
    (id "plop-master")
    (address (list "208.76.58.171"))))

(operating-system
  ;; ...
  (services (cons* (service knot-service-type
                          (knot-configuration
                           (remotes (list plop-master))
                           (zones (list master-zone slave-zone))))
                  ;; ...
                  %base-services)))

```

**knot-service-type** [Variable Scheme]

C'est le type pour le serveur DNS Knot.

Knot DNS est un serveur DNS faisant autorité, ce qui signifie qu'il peut servir plusieurs zones, c'est-à-dire des noms de domaines que vous achetez à un registrar. Ce serveur n'est pas un résolveur, ce qui signifie qu'il ne peut pas résoudre les noms pour lesquels il ne fait pas autorité. Ce serveur peut être configuré pour servir des zones comme un serveur maître ou comme un serveur esclave, en fonction des zones. Les zones esclaves récupèrent leurs données des maîtres, et seront servies comme faisant autorité. Du point de vue d'un résolveur, il n'y a pas de différence entre un maître et un esclave<sup>7</sup>.

Les types de données suivants sont utilisés pour configurer le serveur DNS Knot :

**knot-key-configuration** [Type de données]

Type de données représentant une clef. Ce type a les paramètres suivants :

**id** (par défaut : "")

Un identifiant pour d'autres champs de configuration qui se réfèrent à cette clef. Les ID doivent être uniques et non vides.

**algorithm** (par défaut : #f)

L'algorithme à utiliser. Choisissez entre #f, 'hmac-md5, 'hmac-sha1, 'hmac-sha224, 'hmac-sha256, 'hmac-sha384 et 'hmac-sha512.

<sup>7</sup> NdT : Voir la conférence en Français de Stéphane Bortzmeyer pour en apprendre plus sur le DNS : <https://iletaitunefoisinternet.fr/dns-bortzmeyer/index.html>

**secret** (par défaut : "")  
La clef secrète elle-même.

**knot-acl-configuration** [Type de données]  
Type de données représentant une configuration de liste de contrôle d'accès (ACL).  
Ce type a les paramètres suivants :

**id** (par défaut : "")  
Un identifiant pour d'autres champs de configuration qui se réfèrent à cette clef. Les ID doivent être uniques et non vides.

**address** (par défaut : '() )  
Une liste ordonnée d'adresses IP, de sous-réseaux ou d'intervalles de réseaux représentés par des chaînes de caractères. La requête doit correspondre à l'une d'entre elles. La valeur vide signifie que l'adresse n'a pas besoin de correspondre.

**key** (par défaut : '() )  
Une liste ordonnées de références à des clefs représentés par des chaînes. La chaîne doit correspondre à un ID définie dans un **knot-key-configuration**. Aucune clef signifie qu'une clef n'est pas nécessaire pour correspondre à l'ACL.

**action** (par défaut : '() )  
Une liste ordonnée d'actions permises ou interdites par cet ACL. Les valeurs possibles sont une liste de zéro ou plus d'éléments entre '**transfer**, '**notify** et '**update**.

**deny?** (par défaut : #f)  
Lorsque la valeur est vraie, l'ACL définie des restrictions. Les actions listées sont interdites. Lorsque la valeur est fausse, les actions listées sont autorisées.

**zone-entry** [Type de données]  
Type de données représentant une entrée dans un fichier de zone. Ce type a les paramètres suivants :

**name** (par défaut : "@")  
Le nom de l'enregistrement. "@" se réfère à l'origine de la zone. Les noms sont relatifs à l'origine de la zone. Par exemple, dans la zone **example.org**, "**ns.example.org**" se réfère en fait à **ns.example.org.example.org**. Les noms qui finissent par un point sont absolus, ce qui signifie que "**ns.example.org.**" se réfère bien à **ns.example.org**.

**ttd** (par défaut : "")  
La durée de vie (TTL) de cet enregistrement. S'il n'est pas indiqué, le TTL par défaut est utilisé.

**class** (par défaut : "IN")  
La classe de l'enregistrement. Knot ne supporte actuellement que "IN" et partiellement "CH".

**type** (par défaut : "A")

Le type d'enregistrement. Les types usuels sont A (une adresse IPv4), NS (serveur de nom) et MX (serveur de courriel). Bien d'autres types sont définis.

**data** (par défaut : "")

Les données contenues dans l'enregistrement. Par exemple une adresse IP associée à un enregistrement A, ou un nom de domaine associé à un enregistrement NS. Rappelez-vous que les noms de domaines sont relatifs à l'origine à moins qu'ils ne finissent par un point.

**zone-file** [Type de données]

Type données représentant le contenu d'un fichier de zone. Ce type a les paramètres suivants :

**entries** (par défaut : '()')

La liste des entrées. On s'occupe de l'enregistrement SOA, donc vous n'avez pas besoin de l'ajouter dans la liste des entrées. Cette liste devrait contenir une entrée pour votre serveur DNS primaire faisant autorité. En plus d'utiliser une liste des entrées directement, vous pouvez utiliser **define-zone-entries** pour définir un objet contenant la liste des entrées plus facilement, que vous pouvez ensuite passer au champ **entries** de **zone-file**.

**origin** (par défaut : "")

Le nom de votre zone. Ce paramètre ne peut pas être vide.

**ns** (par défaut : "ns")

Le domaine de votre serveur DNS primaire faisant autorité. Le nom est relatif à l'origine, à moins qu'il finisse par un point. Il est nécessaire que ce serveur DNS primaire corresponde à un enregistrement NS dans la zone et qu'il soit associé à une adresse IP dans la liste des entrées.

**mail** (par défaut : "hostmaster")

Une adresse de courriel pour vous contacter en tant que propriétaire de la zone. Cela se transforme en <mail>@<origin>.

**serial** (par défaut : 1)

Le numéro de série de la zone. Comme c'est utilisé pour vérifier les changements à la fois par les esclaves et par les résolveurs, il est nécessaire qu'il ne décroisse *jamais*. Incrémentez-le toujours quand vous faites un changement sur votre zone.

**refresh** (par défaut : (\* 2 24 3600))

La fréquence à laquelle les esclaves demanderont un transfert de zone. Cette valeur est un nombre de secondes. On peut le calculer avec des multiplications ou avec (**string->duration**).

**retry** (par défaut : (\* 15 60))

La période après laquelle un esclave essaiera de contacter son maître lorsqu'il échoue à le faire la première fois.

**expiry** (par défaut : (\* 14 24 3600))

TTL par défaut des enregistrements. Les enregistrements existants sont considérés corrects pour au moins cette durée. Après cette période, les résolveurs invalideront leur cache et vérifieront de nouveau qu'ils existent toujours.

**nx** (par défaut : 3600)

TTL par défaut des enregistrement inexistants. Ce TTL est habituellement court parce que vous voulez que vous nouveaux domaines soient disponibles pour tout le monde le plus rapidement possible.

**knot-remote-configuration** [Type de données]

Type de données représentant une configuration de serveurs distants. Ce type a les paramètres suivants :

**id** (par défaut : "")

Un identifiant pour que les autres champs de configuration se réfèrent à ce serveur distant. les ID doivent être uniques et non vides.

**address** (par défaut : '() )

Une liste ordonnée d'adresses IP de destination. Ces adresses sont essayées en séquence. Un port facultatif peut être donné avec le séparateur @. Par exemple (`list "1.2.3.4" "2.3.4.5@53"`). Le port par défaut est le 53.

**via** (par défaut : '() )

Une liste ordonnée d'adresses IP sources. Une liste vide fera choisir une IP source appropriée à Knot. Un port facultatif peut être donné avec le séparateur @. La valeur par défaut est de choisir aléatoirement.

**key** (par défaut : #f)

Une référence à une clef, c'est-à-dire une chaîne contenant l'identifiant d'une clef définie dans un champ `knot-key-configuration`.

**knot-keystore-configuration** [Type de données]

Type de données représentant une base de clefs pour garder les clefs dnssec. Ce type a les paramètres suivants :

**id** (par défaut : "")

L'id de cette base de clefs. Il ne doit pas être vide.

**backend** (par défaut : 'pem)

Le moteur de stockage des clefs. Cela peut être 'pem ou 'pkcs11.

**config** (par défaut : "/var/lib/knot/keys/keys")

La chaîne de configuration du moteur. Voici un exemple pour PKCS#11 : `"pkcs11:token=knot;pin-value=1234/gnu/store/.../lib/pkcs11/libsoftsm2.so"`. Pour le moteur pem, la chaîne représente un chemin dans le système de fichiers.

**knot-policy-configuration** [Type de données]

Type de données représentant une politique dnssec. Knot DNS est capable de signer automatiquement vos zones. Il peut soit générer et gérer vos clefs automatiquement ou utiliser des clefs que vous générez.

Dnssec est habituellement implémenté avec deux clefs : une KSK (key signing key) qui est utilisé pour signer une seconde, la ZSK (zone signing key) qui est utilisée pour signer la zone. Pour pouvoir être de confiance, la KSK doit être présente dans la zone parente (normalement un domaine de haut niveau). Si votre registrar supporte dnssec, vous devrez leur envoyer le hash de votre KSK pour qu'il puisse ajouter un enregistrement DS dans la zone parente. Ce n'est pas automatique et vous devrez le faire à chaque fois que vous changerez votre KSK.

La politique définit aussi la durée de vie des clefs. Habituellement, la ZSK peut être changée facilement et utilise des fonctions cryptographiques plus faibles (avec un paramètre plus faible) pour signer les enregistrements rapidement, donc elles sont changées très régulièrement. La KSK en revanche requiert une interaction manuelle avec le registrar, donc elle change moins souvent et utilise des paramètres plus robustes puisqu'elle ne signe qu'un seul enregistrement.

Ce type a les paramètres suivants :

`id` (par défaut : "")

L'id de la politique. Il ne doit pas être vide.

`keystore` (par défaut : "default")

Une référence à une base de clefs, c'est-à-dire une chaîne contenant l'identifiant d'une base de clefs définie dans un champ `knot-keystore-configuration`. L'identifiant "default" signifie la base par défaut (une base de données kasp initialisée par ce service).

`manual?` (par défaut : #f)

Indique si la clef est gérée manuellement ou automatiquement.

`single-type-signing?` (par défaut : #f)

Lorsque la valeur est #t, utilise le schéma de signature Single-Type.

`algorithm` (par défaut : "ecdsap256sha256")

Un algorithme de clef de signature et de signatures.

`sk-size` (par défaut : 256)

La longueur de la KSK. Remarquez que cette valeur est correcte pour l'algorithme par défaut, mais ne serait pas sécurisée pour d'autres algorithmes.

`zsk-size` (par défaut : 256)

La longueur de la ZSK. Remarquez que cette valeur est correcte pour l'algorithme par défaut, mais ne serait pas sécurisée pour d'autres algorithmes.

`dnskey-ttl` (par défaut : 'default')

La valeur du TTL pour les enregistrements DNSKEY ajoutés au sommet de la zone. La valeur spéciale 'default' signifie la même valeur que le TTL du SOA de la zone.

`zsk-lifetime` (par défaut : (\* 30 24 3600))

La période entre la publication d'une ZSK et l'initialisation d'un nouveau changement.



**propagation-delay** (par défaut : (\* 24 3600))  
Un délai supplémentaire pour chaque étape du changement. Cette valeur devrait être assez grande pour couvrir le temps de propagation des données entre le serveur primaire et tous les secondaires.

**rrsig-lifetime** (par défaut : (\* 14 24 3600))  
Une période de validité des nouvelles signatures.

**rrsig-refresh** (par défaut : (\* 7 24 3600))  
Une période qui indique combien de temps avant l'expiration d'une signature elle sera rafraîchie.

**nsec3?** (par défaut : #f)  
Lorsque la valeur est #t, on utilisera NSEC3 au lieu de NSEC.

**nsec3-iterations** (par défaut : 5)  
Le nombre de fois supplémentaires que le hash est effectué.

**nsec3-salt-length** (par défaut : 8)  
La longueur du champ de sel en octets, ajouté au nom du propriétaire avant de hasher.

**nsec3-salt-lifetime** (par défaut : (\* 30 24 3600))  
La période de validité des nouveaux champs sel.

**knot-zone-configuration** [Type de données]  
Type de données représentant la zone servie par Knot. ce type a les paramètres suivants :

**domain** (par défaut : "")  
Le domaine servi par cette configuration. Il ne doit pas être vide.

**file** (par défaut : "")  
Le fichier où la zone est sauvegardée. Ce paramètre est ignoré pour les zones maîtres. La valeur vide signifie l'emplacement par défaut qui dépend du nom de domaine.

**zone** (par défaut : (zone-file))  
Le contenu du fichier de zone. Ce paramètre est ignoré par les zones esclaves. Il doit contenir un enregistrement zone-file.

**master** (par défaut : '())  
Une liste des serveurs distants maîtres. Lorsque la liste est vide, cette zone est un maître. Lorsque la valeur est indiquée, cette zone est un esclave. C'est la liste des identifiants des serveurs distants.

**ddns-master** (par défaut : #f)  
Le maître principal. Lorsque la valeur est vide, la valeur par défaut est le premier maître de la liste des maîtres.

**notify** (par défaut : '())  
Une liste d'identifiants de groupe de serveurs esclaves.

**acl** (par défaut : '())  
Une liste d'identifiants d'ACL.

**semantic-checks?** (par défaut : **#f**)  
Lorsque la valeur est indiquée, cela ajoute plus de vérifications sémantiques à la zone.

**zonefile-sync** (par défaut : **0**)  
Le délai entre une modification en mémoire et sur le disque. **0** signifie une synchronisation immédiate.

**zonefile-load** (par défaut : **#f**)  
La manière dont le contenu du fichier de zone influe sur le chargement de la zone. Les valeurs possibles sont :

- **#f** pour utilise la valeur par défaut de Knot,
- **'none** pour ne pas utiliser le fichier de zone du tout,
- **'difference** pour calculer les différences entre le contenu déjà disponible et le contenu du fichier de zone et les appliquer au contenu actuel de la zone,
- **'difference-no-serial** pour la même chose que **'difference**, mais en ignorant le serial du SOA du fichier de zone, pour que le serveur s'en charge automatiquement.
- **'whole** pour charger le contenu de la zone depuis le fichier de zone.

**journal-content** (par défaut : **#f**)  
La manière dont le journal est utilisé pour stocker la zone et ses changements. Les valeurs possibles sont **'none** pour ne pas l'utiliser du tout, **'changes** pour stocker les changements et **'all** pour stocker le contenu. **#f** ne met pas en place cette option et la valeur par défaut de Knot est utilisée.

**max-journal-usage** (par défaut : **#f**)  
La taille maximale du journal sur le disque. **#f** ne met pas en place cette option et la valeur par défaut de Knot est utilisée.

**max-journal-depth** (par défaut : **#f**)  
La taille maximale de l'historique. **#f** ne met pas en place cette option et la valeur par défaut de Knot est utilisée.

**max-zone-size** (par défaut : **#f**)  
La taille maximale du fichier de zone. Cette limite est prise en charge pour les transferts entrants et les mises à jour. **#f** ne met pas en place cette option et la valeur par défaut de Knot est utilisée.

**dnssec-policy** (par défaut : **#f**)  
Une référence à un enregistrement **knot-policy-configuration**, ou le nom spécial **"default"**. Si la valeur est **#f**, cette zone n'est pas signée.

**serial-policy** (par défaut : **'increment**)  
Une politique entre **'increment** et **'unixtime**.

**knot-configuration** [Type de données]  
Type de données représentant la configuration de Knot. Ce type a les paramètres suivants :

- knot** (par défaut : `knot`)  
Le paquet Knot.
- run-directory** (par défaut : `"/var/run/knot"`)  
Le répertoire de travail. Ce répertoire sera utilisé pour le fichier pid et les sockets.
- includes** (par défaut : `'()`)  
Une liste plate de chaînes ou d'objets simili-fichiers qui seront inclus en haut du fichier de configuration.  
Cela peut être utile pour gérer des secrets hors-bande. Par exemple, on peut stocker des clefs secrètes dans un fichier hors-bande qui n'est pas géré par Guix, et donc pas visible dans `/gnu/store` — p. ex. vous pouvez stocker la configuration des clefs secrètes dans `/etc/knot/secrets.conf` et ajouter ce fichier à la liste `includes`.  
On peut générer une clé secrète `tsig` (pour `nsupdate` et les transferts de zones) avec la commande `keymgr` du paquet `knot`. Remarquez que le paquet n'est pas automatiquement installé par le service. L'exemple suivant montre comment générer une nouvelle clé `tsig` :
- ```
keymgr -t mysecret > /etc/knot/secrets.conf
chmod 600 /etc/knot/secrets.conf
```
- Remarquez aussi que la clé générée sera nommée `mysecret`, donc c'est le nom qui est requis dans le champ `key` de l'enregistrement `knot-acl-configuration` et aux autres endroits qui se réfèrent à cette clé.  
Cela peut aussi être utilisé pour ajouter des configurations qui ne sont pas prises en charge par cette interface.
- listen-v4** (par défaut : `"0.0.0.0"`)  
Une adresse IP sur laquelle écouter.
- listen-v6** (par défaut : `:::"`)  
Une adresse IP sur laquelle écouter.
- listen-port** (par défaut : `53`)  
Un port sur lequel écouter.
- keys** (par défaut : `'()`)  
La liste des `knot-key-configuration` utilisés par cette configuration.
- acls** (par défaut : `'()`)  
La liste des `knot-acl-configuration` utilisés par cette configuration.
- remotes** (par défaut : `'()`)  
La liste des `knot-remote-configuration` utilisés par cette configuration.
- zones** (par défaut : `'()`)  
La liste des `knot-zone-configuration` utilisés par cette configuration.

## Service de résolveur Knot

- knot-resolver-service-type** [Variable Scheme]  
C'est le type du service de résolution de `knot`, dont la valeur devrait être un objet `knot-resolver-configuration` comme dans cet exemple :

```
(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
""))))
```

Pour plus d'information, regardez [manual \(https://knot-resolver.readthedocs.org/en/stable/daemon.html#configuration\)](https://knot-resolver.readthedocs.org/en/stable/daemon.html#configuration).

**knot-resolver-configuration** [Type de données]

Type de données représentant la configuration de knot-resolver.

**package** (par défaut : *knot-resolver*)

Objet de paquet du résolveur DNS knot.

**kresd-config-file** (par défaut : %kresd.conf)

Objet simili-fichier du fichier de configuration de kresd à utiliser. Par défaut il écouterait sur 127.0.0.1 et ::1.

**garbage-collection-interval** (par défaut : 1000)

Nombre de millisecondes correspondant à la périodicité avec laquelle *kres-cache-gc* nettoie le cache.

## Services Dnsmasq

**dnsmasq-service-type** [Variable Scheme]

C'est le type du service dnsmasq, dont la valeur devrait être un objet *dnsmasq-configuration* comme dans cet exemple :

```
(service dnsmasq-service-type
  (dnsmasq-configuration
    (no-resolv? #t)
    (servers '("192.168.1.1"))))
```

**dnsmasq-configuration** [Type de données]

Type de données qui représente la configuration de dnsmasq.

**package** (par défaut : *dnsmasq*)

L'objet de paquet du serveur dnsmasq.

**no-hosts?** (par défaut : #f)

Lorsque la valeur est vraie, ne pas lire les noms d'hôte dans */etc/hosts*.

**port** (par défaut : 53)

Le port sur lequel écouter. Le mettre à zéro désactive complètement les réponses DNS, ce qui ne laisse que les fonctions DHCP et TFTP.

**local-service?** (par défaut : #t)

Accepte les requêtes DNS seulement des hôtes dont les adresses sont sur le sous-réseau local, c.-à-d. sur un sous-réseau pour lequel une interface existe sur le serveur.

- listen-addresses** (par défaut : '() )  
Écoute sur les adresses IP données.
- resolv-file** (par défaut : "/etc/resolv.conf")  
Le fichier où lire l'adresse IP des serveurs de noms en amont.
- no-resolv?** (par défaut : #f)  
Lorsque la valeur est vraie, ne pas lire *resolv-file*.
- forward-private-reverse-lookup?** (par défaut : #t)  
Lorsque la valeur est fautive, toutes les requêtes inverses pour les intervalles d'IP privées reçoivent la réponse « le domaine n'existe pas » plutôt qu'être renvoyé en amont.
- query-servers-in-order?** (par défaut : #f)  
Lorsque la valeur est vraie, dnsmasq demande aux serveurs dans le même ordre qu'ils apparaissent dans *servers*.
- servers** (par défaut : '() )  
Spécifiez l'adresse IP des serveurs en amont directement.
- addresses** (par défaut : '() )  
Pour chaque entrée, spécifie une adresse IP à renvoyer pour les hôtes des domaines donnés. Les requêtes dans les domaines ne sont jamais envoyés et la réponse sera toujours l'adresse IP spécifiée.  
C'est utile pour rediriger des hôtes localement, comme dans cet exemple :
- ```
(service dnsmasq-service-type
  (dnsmasq-configuration
    (addresses
      '(; Redirige vers un serveur web local.
        "/example.org/127.0.0.1"
        ; Redirige un sous-domaine vers une adresse IP spécifique
        "/subdomain.example.org/192.168.1.42"))))
```
- Remarquez que les règles dans */etc/hosts* prennent le pas sur celles-ci.
- cache-size** (par défaut : 150)  
Indique la taille du cache de dnsmasq. Indiquer 0 désactive le cache.
- negative-cache?** (par défaut : #t)  
Lorsque la valeur est fautive, désactive le cache des réponses négatives.
- cpe-id** (par défaut : #f)  
Si une valeur est indiquée, ajoute un identifiant CPE (Customer-Premises Equipment) aux requêtes DNS qui sont renvoyées en amont.
- tftp-enable?** (par défaut : #f)  
Indique s'il faut activer le serveur TFTP inclus.
- tftp-no-fail?** (par défaut : #f)  
Si la valeur est vraie, dnsmasq n'échoue pas si le serveur TFTP ne peut pas démarrer.

- tftp-single-port?** (par défaut : #f)  
Indique s'il faut utiliser un seul port pour TFTP.
- tftp-secure?** (par défaut : #f)  
Si la valeur est vraie, seuls les fichiers appartenant à l'utilisateur sous lequel tourne dnsmasq sont accessibles.  
Si dnsmasq est lancé en root, des règles différentes s'appliquent : **tftp-secure?** n'a pas d'effet, mais seuls les fichiers qui ont le bit de lecture pour tout le monde sont accessibles.
- tftp-max** (par défaut : #f)  
Si une valeur est indiquée, indique le nombre maximal de connexions permises en parallèle.
- tftp-mtu** (par défaut : #f)  
Si une valeur est indiquée, indique le MTU pour les paquets TFTP.
- tftp-no-blocksize?** (par défaut : #f)  
Si la valeur est vraie, empêche que le serveur TFTP ne négocie la taille de bloc avec un client.
- tftp-lowercase?** (par défaut : #f)  
Indique s'il faut convertir tous les noms de fichiers dans les requêtes TFTP en minuscule.
- tftp-port-range** (par défaut : #f)  
Si une valeur est indiquée, fixe les ports dynamiques (un par client) à la zone données ("**<début>**,**<fin>**").
- tftp-root** (par défaut : /var/empty,lo)  
Cherche les fichiers à transférer avec TFTP relativement au répertoire donné. Lorsque la valeur est indiquée, les chemins TFTP qui contiennent '.' sont rejetés, pour éviter que les clients n'arrivent en dehors de la racine donnée. Les chemins absolus (commençant par '/') sont permis, mais ils doivent être à l'intérieur de la racine de TFTP. Si un argument d'interface facultatif est donné, le répertoire n'est utilisé que pour les requêtes TFTP via cette interface.
- tftp-unique-root** (par défaut : #f)  
Si une valeur est indiquée, ajoute d'IP ou l'adresse matérielle du client TFTP en tant que composant du chemin à la fin de la racine TFTP. Valide uniquement si une racine TFTP est indiquée et que le répertoire existe. La valeur par défaut ajoute l'adresse IP (au format à trois points standard).  
Par exemple, si **--tftp-root** est '/tftp' et que le client '1.2.3.4' demande le fichier **monfichier** alors le chemin effectif sera **/tftp/1.2.3.4/monfichier** si **/tftp/1.2.3.4** existe, ou **/tftp/monfichier** sinon. Avec '=mac', c'est l'adresse MAC qui sera ajoutée à la place, en utilisant des nombres en minuscule à taille fixe et séparés par des tirets, p. ex. : '01-02-03-04-aa-bb'. Remarquez que la résolution d'une adresse MAC n'est possible que si le client est sur le réseau local ou a obtenu un bail DHCP de dnsmasq.

## service ddclient

Le service ddclient décrit plus bas lance le démon ddclient, qui prend en charge la mise à jour automatique des entrées DNS pour les fournisseurs de service comme Dyn (<https://dyn.com/dns/>).

L'exemple suivant montre comment instantier le service avec sa configuration par défaut :

```
(service ddclient-service-type)
```

Remarquez que ddclient a besoin d'accéder à des identifiants stockés dans un *fichier de secrets*, par défaut `/etc/ddclient/secrets` (voir `secret-file` plus bas). On s'attend à ce que vous créiez ce fichier manuellement, de manière externe à guix (vous *pourriez* ajouter ce fichier dans une partie de votre configuration, par exemple avec `plain-file`, mais il serait lisible pour tout le monde via `/gnu/store`). Voir les exemples dans le répertoire `share/ddclient` du paquet `ddclient`.

Les champs de `ddclient-configuration` disponibles sont :

`package ddclient` [paramètre de `ddclient-configuration`]  
Le paquet ddclient.

`integer daemon` [paramètre de `ddclient-configuration`]  
La période après laquelle ddclient réessaiera de vérifier l'IP et le nom de domaine.  
La valeur par défaut est '300'.

`boolean syslog` [paramètre de `ddclient-configuration`]  
Utiliser syslog pour la sortie.  
La valeur par défaut est '#t'.

`string mail` [paramètre de `ddclient-configuration`]  
Courriel de l'utilisateur.  
La valeur par défaut est "root".

`string mail-failure` [paramètre de `ddclient-configuration`]  
Courriel de l'utilisateur pour les échecs.  
La valeur par défaut est "root".

`string pid` [paramètre de `ddclient-configuration`]  
Le fichier de PID de ddclient.  
La valeur par défaut est `"/var/run/ddclient/ddclient.pid"`.

`boolean ssl` [paramètre de `ddclient-configuration`]  
Activer le support de SSL.  
La valeur par défaut est '#t'.

`string user` [paramètre de `ddclient-configuration`]  
Spécifie le nm d'utilisateur ou l'ID qui est utilisé pour lancer le programme ddclient.  
La valeur par défaut est "ddclient".

**string group** [paramètre de `ddclient`-configuration]  
 Groupe de l'utilisateur qui lancera le programme `ddclient`.  
 La valeur par défaut est `"ddclient"`.

**string secret-file** [paramètre de `ddclient`-configuration]  
 Fichier de secrets qui sera ajouté au fichier `ddclient.conf`. Ce fichier contient les paramètres d'authentification utilisés par `ddclient`. On s'attend à ce que vous le créiez manuellement.  
 La valeur par défaut est `"/etc/ddclient/secrets.conf"`.

**list extra-options** [paramètre de `ddclient`-configuration]  
 Options supplémentaires qui seront ajoutées au fichier `ddclient.conf`.  
 La valeur par défaut est `()`.

### 12.9.22 Services VNC

le module (`gnu services vnc`) fournit les services liés à *l'informatique virtuelle en réseau* (VNC), qui rend possible l'utilisation locale d'applications Xorg graphiques qui tournent sur une machine distante. Avec un gestionnaire graphique qui prend en charge le *protocole de contrôle de gestion de l'affichage X*, comme GDM (voir [gdm], page 325) ou LightDM (voir [lightdm], page 330), il est possible de rendre un bureau à distance complet pour un environnement multi-utilisateur.

#### Xvnc

Xvnc est un serveur VNC qui démarre son propre serveur X. Cela signifie qu'il peut lancer des serveurs sans affichage. Les implémentations de Xvnc fournies par `tigervnc-server` et `turbovnc` se veulent rapides et efficaces.

**Variable Scheme `xvnc-service-type`** [Variable]  
 Le service `xvnc-server-type` peut être configuré via l'enregistrement `xvnc-configuration`, documenté ci-dessous. Un second affichage virtuel peut être créé sur une machine à distance via la configuration suivante :

```
(service xvnc-service-type
  (xvnc-configuration (display-number 10)))
```

Pour l'exemple, la commande `xclock` peut ensuite être démarrée sur la machine à distance sur l'affichage numéro 10, et elle peut être affichée localement via la commande `vncviewer` :

```
# Démarre xclock sur la machine distante.
ssh -L5910:localhost:5910 -- guix shell xclock -- env DISPLAY=:10 xclock
# On y accède via VNC.
guix shell tigervnc-client -- vncviewer localhost:5910
```

La configuration suivante combine XDMCP et Inetd pour permettre à plusieurs utilisateurs ou utilisatrices d'utiliser le système à distance et de se connecter via le gestionnaire d'affichage GDM en même temps :

```
(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))))))
```

Vous pouvez ensuite vous connecter à distance en utilisant la commande `vncviewer` ou un client VNC compatible et démarrer une session de bureau de votre choix :

```
vncviewer hôte-distant:5905
```

**Attention:** À moins que votre machine ne se trouve dans un environnement contrôlé, pour des raisons de sécurité, la configuration `localhost?` de l'enregistrement `xvnc-configuration` devrait garder sa valeur par défaut `#t` et être exposé par un moyen sécurisé comme la redirection de port SSH. Le port XDMCP, UDP 177 devrait aussi être bloqué de l'extérieur par un pare-feu, car ce n'est pas un protocole sécurisé et peut exposer les identifiants en clair.

`xvnc-configuration` [Type de données]

Les champs de `xvnc-configuration` disponibles sont :

`xvnc` (par défaut : `tigervnc-server`) (type : simili-fichier)

Le paquet qui fournit le binaire `Xvnc`.

`display-number` (par défaut : 0) (type : nombre)

Le numéro d'affichage utilisé par `Xvnc`. Vous devriez le paramétrer à un numéro non utilisé par un serveur `Xorg`.

`geometry` (par défaut : "1024x768") (type : chaîne)

La taille du bureau à créer.

`depth` (par défaut : 24) (type : color-depth)

La profondeur de pixel en bits du bureau à créer . Les valeurs possibles sont 16, 24 et 32.

`port` (type : peut-être-port)

Le port sur lequel écouter les connexions entrantes des visualiseurs. Si la valeur n'est pas spécifiée, écoute sur le port 5900 plus le numéro d'affichage par défaut.

`ipv4?` (par défaut : `#t`) (type : booléen)

Utiliser IPv4 pour les connexions entrantes et sortantes.

`ipv6?` (par défaut : `#t`) (type : booléen)

Utiliser IPv6 pour les connexions entrantes et sortantes.

- password-file** (type : peut-être-chaine)  
Le mot de passe à utiliser, si vous en voulez un. Consultez `vncpasswd(1)` pour apprendre à générer un tel fichier.
- xdmcp?** (par défaut : #f) (type : booléen)  
Demande une session au serveur XDMCP. Cela permet aux utilisateurs et aux utilisatrices de se connecter à une session de bureau depuis l'écran du gestionnaire de connexion. Pour un scénario à plusieurs utilisateurs et utilisatrices, vous devrez également activer l'option `inetd?` pour que chaque connexion au serveur VNC soit gérée séparément, plutôt qu'elles soient partagées.
- inetd?** (par défaut : #f) (type : booléen)  
Utiliser un service de type `Inetd`, qui lance le serveur `Xvnc` à la demande.
- frame-rate** (par défaut : 60) (type : entier)  
Le nombre maximum de mise à jour par seconde envoyées à chaque client.
- security-types** (par défaut : ("None")) (type : types-de-sécurité)  
Les schémas de sécurité autorisés à utiliser pour les connexions entrantes. La valeur par défaut est "None", ce qui est sûr étant donné que `Xvnc` est configuré pour authentifier les personnes via le gestionnaire d'affichage et seulement pour les connexions locales. Les valeurs acceptées sont les suivantes : ("None" "VncAuth" "Plain" "TLSTNone" "TLSTVnc" "TLSTPlain" "X509None" "X509Vnc")
- localhost?** (par défaut : #t) (type : booléen)  
Ne permettre que les connexions de la même machine. La valeur est `#true` par défaut par sécurité, ce qui signifie que SSH ou d'autres manières sécurisées doivent être utilisés pour exposer le port distant.
- log-level** (par défaut : 30) (type : niveau-de-journalisation)  
Le niveau de journalisation, un nombre entre 0 et 100, 100 étant la sortie la plus verbeuse. Les messages de journalisation sont envoyés au `syslog`.
- extra-options** (par défaut : ()) (type : chaines)  
On peut utiliser cette option pour fournir des options `Xvnc` supplémentaires qui ne sont pas proposées par cet enregistrement `<xvnc-configuration>`.

### 12.9.23 Services VPN

Le module (`gnu services vpn`) fournit des services liés aux *réseaux privés virtuels* (VPN).

#### Bitmask

**bitmask-service-type** [Variable Scheme]  
Un type de service pour le client VPN Bitmask (<https://bitmask.net>). Il rend le client disponible dans le système et charge sa politique Polkit. Remarquez que le client attend un `polkit-agent` actif, lancé soit par votre environnement de bureau soit par vous manuellement.

## OpenVPN

Il fournit un service *client* pour que votre machine se connecte à un VPN et un service *serveur* pour que votre machine héberge un VPN.

**openvpn-client-service** [*#:config*] [Procédure Scheme]  
(*openvpn-client-configuration*)

Renvoie un service qui lance **openvpn**, un démon VPN, en tant que client.

**openvpn-server-service** [*#:config*] [Procédure Scheme]  
(*openvpn-server-configuration*)

Renvoie un service qui lance **openvpn**, un démon VPN, en tant que serveur.

Les deux services peuvent être lancés en même temps.

**openvpn-client-configuration** [Type de données]

Les champs de **openvpn-client-configuration** disponibles sont :

**openvpn** (par défaut : **openvpn**) (type : simili-fichier)  
Le paquet OpenVPN.

**pid-file** (par défaut : `"/var/run/openvpn/openvpn.pid"`) (type : chaîne)  
Le fichier de PID d'OpenVPN.

**proto** (par défaut : **udp**) (type : proto)  
Le protocole (UDP ou TCP) utilisé pour ouvrir un canal entre les clients et les serveurs.

**dev** (par défaut : **tun**) (type : dev)  
Le périphérique utilisé pour représenter la connexion VPN.

**ca** (par défaut : `"/etc/openvpn/ca.crt"`) (type : peut-être-chaîne)  
L'autorité de certification qui sert à vérifier les connexions.

**cert** (par défaut : `"/etc/openvpn/client.crt"`) (type : peut-être-chaîne)  
Le certificat de la machine sur laquelle tourne le démon. Il devrait être signé par l'autorité indiquée dans **ca**.

**key** (par défaut : `"/etc/openvpn/client.key"`) (type : peut-être-chaîne)  
La clef de la machine sur laquelle tourne le démon. Elle doit être la clef dont le certificat est donné dans **cert**.

**comp-lzo?** (par défaut : **#t**) (type : booléen)  
Indique s'il faut utiliser l'algorithme de compression lzo.

**persist-key?** (par défaut : **#t**) (type : booléen)  
Ne pas relire les fichiers de clefs entre les SIGUSR1 et les `-ping-restart`.

**persist-tun?** (par défaut : **#t**) (type : booléen)  
Ne pas fermer et rouvrir les périphériques TUN/TAP ou lancer de scripts de démarrage/d'arrêt entre les SIGUSR1 et les `-ping-restart`.

**fast-io?** (par défaut : **#f**) (type : booléen)  
(Expérimental) Optimise les écritures TUN/TAP/UDP en évitant d'appeler poll/epoll/select avant l'opération d'écriture.

- verbosity** (par défaut : 3) (type : entier)  
Niveau de verbosité.
- tls-auth** (par défaut : #f) (type : tls-auth-client)  
Ajoute une couche d'authentification HMAC supplémentaire au dessus du canal de contrôle TLS pour se protéger contre les attaques DoS.
- auth-user-pass** (type : peut-être-chaine)  
S'authentifie avec le serveur en utilisant le nom d'utilisateur et le mot de passe. L'option est un fichier contenant le nom d'utilisateur et le mot de passe sur deux lignes. N'utilisez pas un objet simili-fichier car il serait ajouté au dépôt et serait lisible pour n'importe quel utilisateur.
- verify-key-usage?** (par défaut : #t) (type : key-usage)  
Indique s'il faut vérifier que le certificat du serveur a l'extension d'utilisation.
- bind?** (par défaut : #f) (type : bind)  
Se lier à un port spécifique.
- resolv-retry?** (par défaut : #t) (type : resolv-retry)  
Réessayer de résoudre l'adresse du serveur.
- remote** (par défaut : ()) (type : liste-de-openvpn-remote)  
Une liste de serveurs distants sur lesquels se connecter.
- openvpn-remote-configuration** [Type de données]  
Les champs de **openvpn-remote-configuration** disponibles sont :
- name** (par défaut : "my-server") (type : chaine)  
Nom du serveur.
- port** (par défaut : 1194) (type : nombre)  
Numéro de port sur lequel écoute le serveur.
- openvpn-server-configuration** [Type de données]  
Les champs de **openvpn-server-configuration** disponibles sont :
- openvpn** (par défaut : openvpn) (type : simili-fichier)  
Le paquet OpenVPN.
- pid-file** (par défaut : "/var/run/openvpn/openvpn.pid") (type : chaine)  
Le fichier de PID d'OpenVPN.
- proto** (par défaut : udp) (type : proto)  
Le protocole (UDP ou TCP) utilisé pour ouvrir un canal entre les clients et les serveurs.
- dev** (par défaut : tun) (type : dev)  
Le périphérique utilisé pour représenter la connexion VPN.
- ca** (par défaut : "/etc/openvpn/ca.crt") (type : peut-être-chaine)  
L'autorité de certification qui sert à vérifier les connexions.
- certe** (par défaut : "/etc/openvpn/client.crt") (type : peut-être-chaine)  
Le certificat de la machine sur laquelle tourne le démon. Il devrait être signé par l'autorité indiquée dans **ca**.

- key** (par défaut : `"/etc/openvpn/client.key"`) (type : peut-être-chaine)  
La clef de la machine sur laquelle tourne le démon. Elle doit être la clef dont le certificat est donné dans `cert`.
- comp-lzo?** (par défaut : `#t`) (type : booléen)  
Indique s'il faut utiliser l'algorithme de compression lzo.
- persist-key?** (par défaut : `#t`) (type : booléen)  
Ne pas relire les fichiers de clefs entre les SIGUSR1 et les `-ping-restart`.
- persist-tun?** (par défaut : `#t`) (type : booléen)  
Ne pas fermer et rouvrir les périphériques TUN/TAP ou lancer de scripts de démarrage/d'arrêt entre les SIGUSR1 et les `-ping-restart`.
- fast-io?** (par défaut : `#f`) (type : booléen)  
(Expérimental) Optimise les écritures TUN/TAP/UDP en évitant d'appeler poll/epoll/select avant l'opération d'écriture.
- verbosity** (par défaut : `3`) (type : entier)  
Niveau de verbosité.
- tls-auth** (par défaut : `#f`) (type : tls-auth-server)  
Ajoute une couche d'authentification HMAC supplémentaire au dessus du canal de contrôle TLS pour se protéger contre les attaques DoS.
- port** (par défaut : `1194`) (type : nombre)  
Spécifie le numéro de port sur lequel les serveurs écoutent.
- server** (par défaut : `"10.8.0.0 255.255.255.0"`) (type : ip-mask)  
Une ip et un masque de sous-réseau spécifiant le sous-réseau dans le réseau virtuel.
- server-ipv6** (par défaut : `#f`) (type : cidr6)  
Une notation CIDR pour spécifier le sous-réseau IPv6 dans le réseau virtuel.
- dh** (par défaut : `"/etc/openvpn/dh2048.pem"`) (type : chaîne)  
Le fichier de paramètres Diffie-Hellman.
- ifconfig-pool-persist** (par défaut : `"/etc/openvpn/ipp.txt"`) (type : chaîne)  
Le fichier qui enregistre les IP des clients.
- redirect-gateway?** (par défaut : `#f`) (type : gateway)  
Lorsque la valeur est vraie, le serveur agira comme une passerelle pour ses clients.
- client-to-client?** (par défaut : `#f`) (type : booléen)  
Lorsque la valeur est vraie, les clients sont autorisés à se parler entre eux dans le VPN.
- keepalive** (par défaut : `(10 120)`) (type : keepalive)  
Fait que des messages de ping sont envoyés régulièrement dans les deux sens pour que chaque côté sache quand l'autre n'est plus disponible. `keepalive` a besoin d'une paire. Le premier élément est la période d'envoi du ping, et le second élément est le délai d'attente avant de considéré que l'autre côté n'est plus disponible.

- max-clients** (par défaut : 100) (type : nombre)  
Le nombre maximum de clients.
- status** (par défaut : `"/var/run/openvpn/status"`) (type : chaîne)  
Le fichier de statut. Ce fichier montre un court rapport sur les connexions actuelles. Il est tronqué et réécrit toutes les minutes.
- client-config-dir** (par défaut : `()`) (type : liste-de-openvpn-ccd)  
La liste des configuration pour certains clients.

## strongSwan

Pour l'instant, le service **strongSwan** fournit seulement une configuration héritée des version précédentes avec des fichiers `ipsec.conf` et `ipsec.secrets`.

**strongswan-service-type** [Variable Scheme]  
Un type de service pour configurer strongSwan pour un VPN (réseau privé virtuel) IPsec. Sa valeur doit être un enregistrement **strongswan-configuration** comme dans cet exemple :

```
(service strongswan-service-type
  (strongswan-configuration
    (ipsec-conf "/etc/ipsec.conf")
    (ipsec-secrets "/etc/ipsec.secrets"))))
```

**strongswan-configuration** [Type de données]  
Le type de données représentant la configuration du service StrongSwan.

- strongswan**  
Le paquet strongSwan à utiliser pour ce service.
- ipsec-conf** (par défaut : `#f`)  
Le nom de fichier de votre `ipsec.conf`. Si différent de `#f`, alors cette valeur et `ipsec-secrets` doivent toutes deux être des chaînes de caractères.
- ipsec-secrets** (par défaut : `#f`)  
Le nom de fichier de votre `ipsec.secrets`. Si différent de `#f`, alors cette valeur et `ipsec-conf` doivent toutes deux être des chaînes de caractères.

## Wireguard

**wireguard-service-type** [Variable Scheme]  
Un type de service pour une interface de tunnel Wireguard. Sa valeur doit être un enregistrement **wireguard-configuration** comme dans cet exemple :

```
(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** [Type de données]

Le type de données représentant la configuration du service Wireguard.

**wireguard**

Le paquet wireguard à utiliser pour ce service.

**interface** (par défaut : "wg0")

Le nom d'interface pour le VPN.

**addresses** (par défaut : "10.0.01/32")

Les adresses IP à assigner à l'interface ci-dessus.

**port** (par défaut : 51820)

Le port sur lequel écouter les connexions entrantes.

**dns** (par défaut : #f)

Les serveurs DNS à annoncer aux clients VPN via DHCP.

**private-key** (par défaut : "/etc/wireguard/private.key")

Le fichier de clé privée pour l'interface. Il est automatiquement généré si le fichier n'existe pas.

**peers** (par défaut : '()')

Les pairs autorisés sur cette interface. C'est une liste d'enregistrements *wireguard-peer*.

**pre-up** (par défaut : '()')

Les commandes de script à lancer avant de configurer l'interface.

**post-up** (par défaut : '()')

Les commandes de script à lancer après la configuration de l'interface.

**pre-down** (par défaut : '()')

Les commandes de script à lancer avec d'arrêter l'interface.

**post-down** (par défaut : '()')

Les commandes de script à lancer après l'arrêt de l'interface.

**table** (par défaut : "auto")

La table de routage à laquelle les routes sont ajoutées, en tant que chaîne. Il y a deux valeurs spéciales : "off" qui désactive la création de route complètement, et "auto" (par défaut) qui ajoute les routes à la table par défaut et active la gestion spéciale des routes par défaut.

**wireguard-peer** [Type de données]

Type de données représentant un pair Wireguard attaché à une interface donnée.

**name** Le nom du pair.

**endpoint** (par défaut : #f)

Le point d'entrée facultatif du pair, comme "demo.wireguard.com:51820". ■

**public-key**

La clé publique du pair, représentée par une chaîne en base64.

**allowed-ips**

Une liste d'adresses IP à partir desquelles le trafic entrant pour ce pair est autorisé et vers lesquelles le trafic à destination de ce pair est dirigé.

**keep-alive** (par défaut : #f)

Une durée en seconde facultative. Un paquet sera régulièrement envoyé au serveur après cette durée. Cela aide à recevoir les connexions entrantes de ce pair si vous êtes derrière un NAT ou un pare-feu.

### 12.9.24 Système de fichiers en réseau

Le module (`gnu services nfs`) fournit les services suivants, qui sont tous utilisés pour monter et exporter des arborescences de répertoires en *network file systems* (NFS).

Bien qu'il soit possible d'utiliser des composants individuels qui forment ensemble un service de système de fichiers en réseau, nous recommandons de configurer un serveur NFS avec `nfs-service-type`.

## Services NFS

Le service NFS s'occupe de paramétrer tous les services composant NFS, les systèmes de fichiers de configuration du noyau et installe les fichiers de configuration aux endroits attendus par NFS.

**nfs-service-type** [Variable Scheme]

Un type de service pour un serveur NFS complet.

**nfs-configuration** [Type de données]

Ce type de données représente la configuration d'un service NFS et de tous ses sous-systèmes.

Il prend les paramètres suivants :

**nfs-utils** (par défaut : `nfs-utils`)

Le paquet `nfs-utils` à utiliser.

**nfs-versions** (par défaut : `'("4.2" "4.1" "4.0")`)

Si une liste de chaînes est fournie, le démon `rpc.nfsd` sera limité à la prise en charge des versions données du protocole NFS.

**exports** (par défaut : `'()`)

C'est une liste de répertoires que le serveur NFS devrait exporter. Chaque entrée est une liste consistant en deux éléments : un nom de répertoire et une chaîne de caractères contenant toutes les options. Voici un exemple dans lequel le répertoire `/export` est servi à tous les clients en lecture-seule :

```
(nfs-configuration
 (exports
  '(("/export"
     "(ro,insecure,no_subtree_check,crossmnt,fsid=0)"))))■
```

**rpcmountd-port** (par défaut : #f)

Le port réseau que le démon `rpc.mountd` devrait utiliser.



- rpcstatd-port** (par défaut : **#f**)  
Le port réseau que le démon `rpc.statd` devrait utiliser.
- rpcbind** (par défaut : `rpcbind`)  
Le paquet `rpcbind` à utiliser.
- idmap-domain** (par défaut : `"localdomain"`)  
Le nom de domaine NFSv4 local.
- nfsd-port** (par défaut : `2049`)  
Le port réseau que le démon `nfsd` devrait utiliser.
- nfsd-threads** (par défaut : `8`)  
Le nombre de thread à utiliser par le démon `nfsd`.
- nfsd-tcp?** (par défaut : **#t**)  
Indique si le démon `nfsd` devrait écouter sur une socket TCP.
- nfsd-udp?** (par défaut : **#f**)  
Indique si le démon `nfsd` devrait écouter sur une socket UDP.
- pipefs-directory** (par défaut : `"/var/lib/nfs/rpc_pipefs"`)  
Le répertoire où le système de fichier `pipefs` doit être monté.
- debug** (par défaut : `'()'`)  
Une liste des sous-systèmes pour lesquels la sortie de débogage est activée. C'est une liste de symboles. Ces symboles sont valides : `nfsd`, `nfs`, `rpc`, `idmap`, `statd` et `mountd`.

Si vous n'avez pas besoin d'un service NFS complet ou préférez le construire vous-même vous pouvez utiliser les composants individuels décrit plus bas.

## Service RPC Bind

Le service RPC Bind fournit un dispositif pour faire correspondre les numéros de programmes à des adresses universelles. De nombreux services liés à NFS utilisent ce dispositif. Donc il est automatiquement démarré lorsqu'un service qui en dépend est démarré.

**rpcbind-service-type** [Variable Scheme]  
Un type de service pour le démon RPC portmapper.

**rpcbind-configuration** [Type de données]  
Type données représentant la configuration du service RPC Bind. Ce type a les paramètres suivants :

**rpcbind** (par défaut : `rpcbind`)  
Le paquet `rpcbind` à utiliser.

**warm-start?** (par défaut : **#t**)  
Si ce paramètre est **#t**, alors le démon lira un fichier d'état au démarrage ce qui lui fait recharger les informations d'états sauvegardés par une instance précédente.

## Pseudo-système de fichiers Pipefs

Le système de fichiers pipefs est utilisé pour transférer des données liées à NFS entre le noyau et les programmes en espace utilisateur.

**pipefs-service-type** [Variable Scheme]

Un type de service pour le pseudo-système de fichiers pipefs.

**pipefs-configuration** [Type de données]

Type de données représentant la configuration du service du pseudo-système de fichiers pipefs. Ce type a les paramètres suivants :

**mount-point** (par défaut : `"/var/lib/nfs/rpc_pipefs"`)

Le répertoire dans lequel le système de fichiers est attaché.

## Service de démon GSS

Le démon du *système de sécurité global* (GSS) fournit une sécurité forte pour les protocoles basés sur des RPC. Avant d'échanger des requêtes RPC, un client RPC doit établir un contexte sécurisé. Typiquement cela se fait avec la commande Kerberos `kinit` ou automatiquement à la connexion avec les services PAM (voir Section 12.9.17 [Services Kerberos], page 433).

**gss-service-type** [Variable Scheme]

Un type de service pour le démon du système de sécurité global (GSS).

**gss-configuration** [Type de données]

Type de données représentant la configuration du service du démon GSS. Ce type a les paramètres suivants :

**nfs-utils** (par défaut : `nfs-utils`)

Le paquet dans lequel la commande `rpc.gssd` se trouve.

**pipefs-directory** (par défaut : `"/var/lib/nfs/rpc_pipefs"`)

Le répertoire où le système de fichier pipefs doit être monté.

## Service de démon IDMAP

Le service du démon `idmap` fournit une correspondance entre les ID utilisateur et les noms d'utilisateurs. Typiquement, cela est requis pour accéder aux systèmes de fichiers montés via NFSv4.

**idmap-service-type** [Variable Scheme]

Un type de service pour le démon de correspondance d'identité (IDMAP).

**idmap-configuration** [Type de données]

Type de données représentant la configuration du service du démon IDMAP. Ce type a les paramètres suivants :

**nfs-utils** (par défaut : `nfs-utils`)

Le paquet dans lequel se trouve la commande `rpc.idmapd`.

**pipefs-directory** (par défaut : `"/var/lib/nfs/rpc_pipefs"`)

Le répertoire où le système de fichier pipefs doit être monté.

`domain` (par défaut : `#f`)

Le nom de domaine NFSv4 local. Il faut que ce soit une chaîne de caractères ou `#f`. Si la valeur est `#f` le démon utilisera le nom de domaine pleinement qualifié de l'hôte.

`verbosity` (par défaut : 0)

Le niveau de verbosité du démon.

### 12.9.25 Services Samba

Le module (`gnu services samba`) fournit des définitions de services pour les services Samba ainsi que d'autres services utilitaires. Actuellement il fournit les services suivants.

#### Samba

Samba (<https://www.samba.org>) fournit des partages réseau pour les répertoires et les imprimantes qui utilisent le protocole SMB/CIFS souvent utilisé sous Windows. Il peut aussi se comporter comme un contrôleur de domaine Active Directory (AD DC) pour les autres hôtes dans un réseau hétérogène avec plusieurs types d'ordinateurs.

Variable Scheme `samba-service-type`

[Variable]

Le type de service pour activer les services samba `samba`, `nmbd`, `smbd` et `winbindd`. Par défaut ce type de service ne lance aucun démon Samba. Ils doivent être activés individuellement.

voici un exemple de base qui configure un partage de fichiers Samba simple et anonyme (sans authentification) qui expose le répertoire `/public`.

**Astuce:** Le répertoire `/public` et sont contenu doivent être disponible en lecture-écriture pour le monde entier, donc vous devrez utiliser `'chmod -R 777 /public'` dessus.

**Attention:** Une telle configuration Samba ne devrait être utilisée que dans des environnements contrôlés et vous ne devriez pas partager des fichiers privés avec, car n'importe qui se connectant à votre réseau y aurait accès.

```
(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`

[Type de données]

Enregistrement de configuration pour la suite Samba.

- package** (par défaut : **samba**)  
Le paquet samba à utiliser.
- config-file** (par défaut : **#f**)  
Le fichier de configuration à utiliser. Pour comprendre sa syntaxe, lancez `'man smb.conf'`.
- enable-samba?** (par défaut : **#f**)  
Active le démon **samba**.
- enable-smbd?** (par défaut : **#f**)  
Active le démon **smbd**.
- enable-nmbd?** (par défaut : **#f**)  
Active le démon **nmbd**.
- enable-winbindd?** (par défaut : **#f**)  
Active le démon **winbindd**.

## Démon de découverte des services web

Le WSDD (démon de découverte des services web) implémente le protocole de découverte dynamique des services web (<http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html>) qui permet des découvrir des hôtes sur le DNS multicast, de la même manière que Avahi. C'est un remplaçant pour les hôtes SMB qui ont désactivé leur SMBv1 pour des raisons de sécurité.

- wsdd-service-type** [Variable Scheme]  
Type pour le démon hôte WSD. La valeur de ce type de service est un enregistrement **wssd-configuration**. Les détails du type d'enregistrement **wsdd-configuration** sont fournis plus bas.
- wsdd-configuration** [Type de données]  
Ce type de données représente la configuration du service wsdd.
- package** (par défaut : **wsdd**)  
Le paquet wsdd à utiliser.
- ipv4only?** (par défaut : **#f**)  
N'écoute que sur les adresses IPv4.
- ipv6only** (par défaut : **#f**)  
N'écoute que sur les adresses IPv6. Remarquez : activer les deux options n'est pas possible, car il n'y aurait aucune version d'IP sur laquelle écouter.
- chroot** (par défaut : **#f**)  
Entre dans un chroot dans un répertoire séparé pour empêcher l'accès à d'autres répertoires. Cela permet d'améliorer la sécurité en cas de vulnérabilité dans **wsdd**.
- hop-limit** (par défaut : **1**)  
Limite le niveau de sauts pour les paquets multicast. La valeur par défaut est **1**, ce qui devrait éviter aux paquets de quitter le réseau local.

- interface** (par défaut : '() )  
Limite l'écoute aux interfaces listées. Par défaut `wadd` écoutera sur toutes les interfaces. Sauf l'interface de rebouclage qui n'est jamais utilisée.
- uuid-device** (par défaut : #f)  
Le protocole WSD nécessite qu'un périphérique ait un UUID. Indiquez ce paramètre pour assigner manuellement un UUID au service.
- domain** (par défaut : #f)  
Notifie que cet hôte est membre d'un Active Directory.
- host-name** (par défaut : #f)  
Indique le nom d'hôte manuellement au lieu de laisser `wadd` hériter du nom de cet hôte. Seul la partie du nom d'hôte d'un éventuel nom de domaine pleinement qualifié sera utilisé dans le cas par défaut.
- preserve-case?** (par défaut : #f)  
Par défaut `wadd` convertira le nom d'hôte dans le groupe de travail en majuscules. L'inverse est vrai pour les noms d'hôtes dans les domaines. Indiquer ce paramètre préservera la casse.
- workgroup** (par défaut : "WORKGROUP")  
Change le nom du groupe de travail. Par défaut `wadd` rapport cet hôte comme étant membre d'un groupe de travail.

### 12.9.26 Intégration continue

Cuirass (<https://guix.gnu.org/fr/cuirass/>) est un outil d'intégration continue pour Guix. On peut l'utiliser aussi bien pour le développement que pour fournir des substituts à d'autres (voir Section 6.3 [Substituts], page 54).

Le module (`gnu services cuirass`) fournit le service suivant.

**cuirass-service-type** [Procédure Scheme]  
Le type du service Cuirass. Sa valeur doit être un objet `cuirass-configuration`, décrit ci-dessous.

Pour ajouter des travaux de construction, vous devez indiquer le champ `specifications` de la configuration. Par exemple, voici un exemple qui construira tous les paquets fournis par le canal `my-channel`.

```
(define %cuirass-specs
  #~(list (specification
          (name "my-channel")
          (build '(channels my-channel))
          (channels
            (cons (channel
                  (name 'my-channel)
                  (url "https://my-channel.git"))
                  %default-channels))))))

(service cuirass-service-type
  (cuirass-configuration
```

```
(specifications %cuirass-specs)))
```

Pour construire le paquet `linux-libre` défini par le canal Guix par défaut, on peut utiliser la configuration suivante.

```
(define %cuirass-specs
  #~(list (specification
          (name "my-linux")
          (build '(packages "linux-libre")))))

(service cuirass-service-type
  (cuirass-configuration
   (specifications %cuirass-specs)))
```

Les autres possibilités de configuration, ainsi que l’enregistrement de spécification lui-même sont décrits dans le manuel de Cuirass (voir Section “Specifications” dans *Cuirass*).

Tandis que les informations liés aux travaux de construction sont directement dans les spécifications, les paramètres globaux pour le processus `cuirass` sont accessibles dans les autres champs de `cuirass-configuration`.

**cuirass-configuration** [Type de données]

Type de données représentant la configuration de Cuirass.

`cuirass` (par défaut : `cuirass`)

Le paquet Cuirass à utiliser.

`log-file` (par défaut : `"/var/log/cuirass.log"`)

Emplacement du fichier de journal.

`web-log-file` (par défaut : `"/var/log/cuirass-web.log"`)

Emplacement du fichier journal utilisé par l’interface web.

`cache-directory` (par défaut : `"/var/cache/cuirass"`)

Emplacement du cache du dépôt.

`user` (par défaut : `"cuirass"`)

Propriétaire du processus `cuirass`.

`group` (par défaut : `"cuirass"`)

Groupe du propriétaire du processus `cuirass`.

`interval` (par défaut : `60`)

Nombre de secondes entre les mises à jour du dépôt suivis des travaux de Cuirass.

`parameters` (par défaut : `#f`)

Lit les paramètres dans le fichier `parameters`. Les paramètres pris en charges sont décrits ici (voir Section “Parameters” dans *Cuirass*).

`remote-server` (par défaut : `#f`)

Un enregistrement `cuirass-remote-server-configuration` pour utiliser le mécanisme de construction distante ou `#f` pour utiliser le mécanisme de construction par défaut.

- database** (par défaut : "dbname=cuirass host=/var/run/postgresql")  
Utiliser *database* comme base de données contenant les travaux et les résultats des constructions passées. Comme Cuirass utilise PostgreSQL comme moteur de base de données, *database* doit être une chaîne de la même forme que "dbname=cuirass host=localhost".
- port** (par défaut : 8081)  
Numéro de port utilisé pour le serveur HTTP.
- host** (par défaut : "localhost")  
Écoute sur l'interface réseau de *host*. La valeur par défaut est d'accepter les connexions depuis localhost.
- specifications** (par défaut : #~'())  
Une gexp (voir Section 9.12 [G-Expressions], page 166) qui s'évalue en une liste d'enregistrement de spécifications. L'enregistrement de spécification est décrit dans le manuel de Cuirass (voir Section "Specifications" dans *Cuirass*).
- use-substitutes?** (par défaut : #f)  
Cela permet d'utiliser des substituts pour éviter de construire toutes les dépendances d'un travail depuis les sources.
- one-shot?** (par défaut : #f)  
N'évaluer les spécification et construire les dérivations qu'une seule fois.
- fallback?** (par défaut : #f)  
Lorsque la substitution d'un binaire pré-construit échoue, revenir à la construction locale du paquet.
- extra-options** (par défaut : '())  
Options supplémentaires à passer au lancement des processus de Cuirass.

## Cuirass, construction distante

Cuirass prend en charge deux mécanisme pour construire les dérivations.

- Par le démon Guix local. C'est le mécanisme de construction par défaut. Une fois les constructions évaluées, elles sont envoyées au démon Guix local. Cuirass écoute ensuite la sortie du démon Guix pour détecter les divers événements de construction.
- Par le mécanisme de construction distante. Les constructions ne sont pas soumises au démon Guix local. À la place, un serveur distant distribue les requêtes de construction aux serveurs connectés, en fonction des priorités de construction.

Pour activer ce mode de construction, vous devez passer un enregistrement `cuirass-remote-server-configuration` au champ `remote-server` de l'enregistrement `cuirass-configuration`. L'enregistrement `cuirass-remote-server-configuration` est décrit plus bas.

Ce mode de construction passe mieux à l'échelle que celui par défaut. C'est le mode de construction utilisé sur la ferme de construction de GNU Guix <https://ci.guix.gnu.org>. Vous devriez l'utiliser si vous utilisez Cuirass pour construire un grand nombre de paquets.

**cuirass-remote-server-configuration** [Type de données]

Le type de données représentant la configuration des serveurs distants de Cuirass.

**backend-port** (par défaut : 5555)

Le port TCP pour communiquer avec les processus **remote-worker** via ZMQ. Sa valeur par défaut est 5555.

**log-port** (par défaut : 5556)

Le port TCP du serveur de journalisation. Sa valeur par défaut est 5556.

**publish-port** (par défaut : 5557)

Le port TCP du serveur de publication. Sa valeur par défaut est 5557.

**log-file** (par défaut : "/var/log/cuirass-remote-server.log")

Emplacement du fichier de journal.

**cache** (par défaut : "/var/cache/cuirass/remote")

Utiliser le répertoire *cache* pour mettre les journaux en cache.

**trigger-url** (par défaut : #f)

Lorsqu'un substitut a été correctement récupéré, cause la préparation d'un substitut sur *trigger-url*.

**publish?** (par défaut : #t)

Si la valeur est fausse, ne démarre pas de serveur de publication et ignore l'argument **publish-port**. Ça peut être utile s'il y a déjà un serveur de publication indépendant qui se trouve à côté du serveur distant.

**public-key**

**private-key**

Utilise les *fichiers* spécifiques comme pair de clefs utilisées pour signer les éléments avant de les publier.

Au moins un serveur distant doit aussi être démarré sur une machine du réseau local pour effectivement effectuer les constructions et rapporter leur état.

**cuirass-remote-worker-configuration** [Type de données]

Type de données représentant la configuration des serveurs distants de Cuirass.

**cuirass** (par défaut : cuirass)

Le paquet Cuirass à utiliser.

**workers** (par défaut : 1)

Démarre *workers* travailleurs en parallèle.

**server** (par défaut : #f)

Ne pas utiliser Avahi pour découvrir des serveurs et se connecter à l'IP **server** donnée à la place.

**systems** (par défaut : (list (%current-system)))

Ne demander des constructions que pour les systèmes *systems* donnés.

**log-file** (par défaut : "/var/log/cuirass-remote-worker.log")

Emplacement du fichier de journal.



`publish-port` (par défaut : 5558)

Le port TCP du serveur de publication. La valeur par défaut est 5558.

`substitute-urls` (par défaut : `%default-substitute-urls`)

La liste des URL où trouver des substituts par défaut.

`public-key`

`private-key`

Utilise les *fichiers* spécifiques comme pair de clefs utilisées pour signer les éléments avant de les publier.

## Laminar

Laminar (<https://laminar.ohwg.net/>) est service d'intégration continue léger et modulaire. Il n'a pas d'interface de configuration web et utilise plutôt des fichiers de configuration et des scripts sous contrôle de version.

Laminar encourage l'utilisation d'outils existants comme bash et cron au lieu de les réinventer.

`laminar-service-type`

[Procédure Scheme]

Le type du service Laminar. Sa valeur doit être un objet `laminar-configuration`, décrit ci-dessous.

Toutes les configurations ont des valeurs par défaut, une configuration minimale pour lancer Laminar se trouve plus bas. Par défaut, l'interface web est disponible sur le port 8080.

(`service laminar-service-type`)

`laminar-configuration`

[Type de données]

Type données qui représente la configuration de Laminar.

`laminar` (par défaut : `laminar`)

Le paquet Laminar à utiliser.

`home-directory` (par défaut : `"/var/lib/laminar"`)

Le répertoire de la configuration des travaux et des répertoires d'exécution.

`bind-http` (par défaut : `"*:8080"`)

L'interface et le port ou la socket unix sur laquelle laminard écoutera les connexions entrantes sur l'interface web.

`bind-rpc` (par défaut : `"unix-abstract:laminar"`)

L'interface et le port ou la socket unix sur laquelle laminard écoutera les commandes entrantes comme les requêtes de construction.

`title` (par défaut : `"Laminar"`)

Le titre de la page à afficher sur l'interface web.

`keep-rundirs` (par défaut : 0)

Indiquez un entier indiquant le nombre de répertoire d'exécution garder par tâche. Les répertoires avec les numéros les plus petits seront supprimés. La valeur par défaut est 0, ce qui signifie que tous les répertoires d'exécution seront immédiatement supprimés.

`archive-url` (par défaut : `#f`)

L'interface web servi par laminard utilisera cette URL pour former les liens vers les artefacts des travaux archivés.

`base-url` (par défaut : `#f`)

URL de base à utiliser pour les liens vers laminar.

## 12.9.27 Services de gestion de l'énergie

### Démon TLP

Le module (`gnu services pm`) fournit une définition de service Guix pour l'outil de gestion d'énergie Linux TLP.

TLP active plusieurs modes un espace utilisateur et dans le noyau. Contrairement à `upower-service`, ce n'est pas un outil passif de surveillance, puisqu'il applique des paramètres personnalisés à chaque fois qu'il détecte une nouvelle source d'énergie. Vous pouvez trouver plus d'informations sur la page d'accueil de TLP (<https://linrunner.de/en/tlp/tlp.html>).

`tlp-service-type`

[Variable Scheme]

Le type de service pour l'outil TLP. Les paramètres par défaut sont optimisés pour la durée de la batterie sur la plupart des systèmes, mais vous pouvez les modifier comme vous voulez en ajoutant un `tlp-configuration` valide :

```
(service tlp-service-type
  (tlp-configuration
    (cpu-scaling-governor-on-ac (list "performance"))
    (sched-powersave-on-bat? #t)))
```

Chaque définition de paramètre est précédée par son type ; par exemple, 'booléen toto' indique que le paramètre `toto` doit être spécifié comme un booléen. Les types qui commencent par `peut-être-` dénotent des paramètres qui n'apparaîtront pas dans la configuration de TLP lorsque leur valeur est non spécifiée, ou est explicitement initialisée à la valeur `%unset-value`.

Les champs de `tlp-configuration` disponibles sont :

`package tlp` [paramètre de `tlp-configuration`]

Le paquet TLP.

`boolean tlp-enable?` [paramètre de `tlp-configuration`]

Indiquez vrai si vous souhaitez activer TLP.

La valeur par défaut est '`#t`'.

`string tlp-default-mode` [paramètre de `tlp-configuration`]

Mode par défaut lorsqu'aucune source d'énergie ne peut être détectée. Les possibilités sont AC et BAT.

La valeur par défaut est '`"AC"`'.

- entier-non-négatif** [paramètre de tlp-configuration]  
**disk-idle-secs-on-ac**  
 Nombre de secondes que le noyau Linux doit attendre après que les disques s'arrêtent pour se synchroniser quand il est sur secteur.  
 La valeur par défaut est '0'.
- entier-non-négatif** [paramètre de tlp-configuration]  
**disk-idle-secs-on-bat**  
 Comme `disk-idle-ac` mais en mode batterie.  
 La valeur par défaut est '2'.
- entier-non-négatif** [paramètre de tlp-configuration]  
**max-lost-work-secs-on-ac**  
 Périodicité du nettoyage des pages invalidées, en secondes.  
 La valeur par défaut est '15'.
- entier-non-négatif** [paramètre de tlp-configuration]  
**max-lost-work-secs-on-bat**  
 Comme `max-lost-work-secs-on-ac` mais en mode batterie.  
 La valeur par défaut est '60'.
- peut-être-liste-de-chaines-séparées-par-des-espaces** [paramètre de tlp-configuration]  
**cpu-scaling-governor-on-ac**  
 Gouverneur de fréquence d'horloge sur secteur. Avec le pilote `intel_pstate`, les possibilités sont `powersave` et `performance`. Avec le pilote `acpi-cpufreq`, les possibilités sont `ondemand`, `powersave`, `performance` et `conservative`.  
 La valeur par défaut est 'disabled'.
- peut-être-liste-de-chaines-séparées-par-des-espaces** [paramètre de tlp-configuration]  
**cpu-scaling-governor-on-bat**  
 Comme `cpu-scaling-governor-on-ac` mais en mode batterie.  
 La valeur par défaut est 'disabled'.
- peut-être-entier-non-négatif** [paramètre de tlp-configuration]  
**cpu-scaling-min-freq-on-ac**  
 Indique la fréquence d'horloge minimale pour le gouverneur sur secteur.  
 La valeur par défaut est 'disabled'.
- peut-être-entier-non-négatif** [paramètre de tlp-configuration]  
**cpu-scaling-max-freq-on-ac**  
 Indique la fréquence d'horloge maximale pour le gouverneur sur secteur.  
 La valeur par défaut est 'disabled'.
- peut-être-entier-non-négatif** [paramètre de tlp-configuration]  
**cpu-scaling-min-freq-on-bat**  
 Indique la fréquence d'horloge minimale pour le gouverneur sur batterie.  
 La valeur par défaut est 'disabled'.

- `peut-être-entier-non-négatif` [paramètre de `tlp-configuration`]  
`cpu-scaling-max-freq-on-bat`  
Indique la fréquence d'horloge maximale pour le gouverneur sur batterie.  
La valeur par défaut est `'disabled'`.
- `peut-être-entier-non-négatif` [paramètre de `tlp-configuration`]  
`cpu-min-perf-on-ac`  
Limite le P-état minimum pour contrôler la dissipation de puissance dans le CPU, sur secteur. Les valeurs sont indiqués comme un pourcentage des performances disponibles.  
La valeur par défaut est `'disabled'`.
- `peut-être-entier-non-négatif` [paramètre de `tlp-configuration`]  
`cpu-max-perf-on-ac`  
Limite le P-état maximum pour contrôler la dissipation de puissance dans le CPU, sur secteur. Les valeurs sont indiqués comme un pourcentage des performances disponibles.  
La valeur par défaut est `'disabled'`.
- `peut-être-entier-non-négatif` [paramètre de `tlp-configuration`]  
`cpu-min-perf-on-bat`  
Comme `cpu-min-perf-on-ac` mais en mode batterie.  
La valeur par défaut est `'disabled'`.
- `peut-être-entier-non-négatif` [paramètre de `tlp-configuration`]  
`cpu-max-perf-on-bat`  
Comme `cpu-max-perf-on-ac` mais en mode batterie.  
La valeur par défaut est `'disabled'`.
- `peut-être-booléen` `cpu-boost-on-ac?` [paramètre de `tlp-configuration`]  
Active la fonctionnalité turbo boost du CPU sur secteur.  
La valeur par défaut est `'disabled'`.
- `peut-être-booléen` `cpu-boost-on-bat?` [paramètre de `tlp-configuration`]  
Comme `cpu-boost-on-ac?` mais en mode batterie.  
La valeur par défaut est `'disabled'`.
- `boolean` `sched-powersave-on-ac?` [paramètre de `tlp-configuration`]  
Permet au noyau Linux de minimiser le nombre de cœurs/hyper-threads CPU utilisés lorsque la charge est faible.  
La valeur par défaut est `'#f'`.
- `boolean` `sched-powersave-on-bat?` [paramètre de `tlp-configuration`]  
Comme `sched-powersave-on-ac?` mais en mode batterie.  
La valeur par défaut est `'#t'`.
- `boolean` `nmi-watchdog?` [paramètre de `tlp-configuration`]  
Active le chien de garde NMI du noyau Linux.  
La valeur par défaut est `'#f'`.

- peut-être-chaine phc-controls** [paramètre de tlp-configuration]  
 Pour les noyaux Linux avec le correctif PHC, change le voltage du CPU. Une valeur serait par exemple "F:V F:V F:V F:V".  
 La valeur par défaut est 'disabled'.
- string energy-perf-policy-on-ac** [paramètre de tlp-configuration]  
 Indique le niveau de performance du CPU par rapport à la politique de gestion de l'énergie sur secteur. Les possibilités sont performance, normal et powersave.  
 La valeur par défaut est "performance".
- string energy-perf-policy-on-bat** [paramètre de tlp-configuration]  
 Comme energy-perf-policy-ac mais en mode batterie.  
 La valeur par défaut est "powersave".
- space-separated-string-list disks-devices** [paramètre de tlp-configuration]  
 Périphériques de disque dur.
- space-separated-string-list disk-apm-level-on-ac** [paramètre de tlp-configuration]  
 Niveau de gestion de l'énergie avancé des disques durs.
- space-separated-string-list disk-apm-level-on-bat** [paramètre de tlp-configuration]  
 Comme disk-apm-bat mais en mode batterie.
- peut-être-liste-de-chaines-séparées-par-des-espaces disk-spindown-timeout-on-ac** [paramètre de tlp-configuration]  
 Délai d'attente pour arrêter de faire tourner les disques. Une valeur doit être spécifiée pour chaque disque dur déclaré.  
 La valeur par défaut est 'disabled'.
- peut-être-liste-de-chaines-séparées-par-des-espaces disk-spindown-timeout-on-bat** [paramètre de tlp-configuration]  
 Comme disk-spindown-timeout-on-ac mais en mode batterie.  
 La valeur par défaut est 'disabled'.
- peut-être-liste-de-chaines-séparées-par-des-espaces disk-iosched** [paramètre de tlp-configuration]  
 Sélectionne l'ordonnanceur d'entrées-sorties pour le disque. Une valeur doit être spécifiée pour chaque disque déclaré. Les possibilités sont par exemple cfq, deadline et noop.  
 La valeur par défaut est 'disabled'.
- string sata-linkpwr-on-ac** [paramètre de tlp-configuration]  
 Niveau de gestion de l'énergie des lien SATA aggressive (ALPM). Les possibilités sont min\_power, medium\_power et max\_performance.  
 La valeur par défaut est "max\_performance".

- string sata-linkpwr-on-bat** [paramètre de tlp-configuration]  
Comme `sata-linkpwr-ac` mais en mode batterie.  
La valeur par défaut est `"min_power"`.
- peut-être-chaîne** [paramètre de tlp-configuration]  
**sata-linkpwr-blacklist**  
Exclu les périphériques SATA spécifiés de la gestion de l'énergie des liens.  
La valeur par défaut est `'disabled'`.
- peut-être-booléen-on-off** [paramètre de tlp-configuration]  
**ahci-runtime-pm-on-ac?**  
Active la gestion de l'énergie à l'exécution pour les contrôleurs AHCI et les disques, sur secteur.  
La valeur par défaut est `'disabled'`.
- peut-être-booléen-on-off** [paramètre de tlp-configuration]  
**ahci-runtime-pm-on-bat?**  
Comme `ahci-runtime-pm-on-ac` mais en mode batterie.  
La valeur par défaut est `'disabled'`.
- entier-non-négatif** [paramètre de tlp-configuration]  
**ahci-runtime-pm-timeout**  
Secondes d'inactivités avant de suspendre les disques.  
La valeur par défaut est `'15'`.
- string pcie-aspm-on-ac** [paramètre de tlp-configuration]  
Niveau de gestion de l'énergie des états actifs de PCI Express. Les possibilités sont `default`, `performance` et `powersave`.  
La valeur par défaut est `"performance"`.
- string pcie-aspm-on-bat** [paramètre de tlp-configuration]  
Comme `pcie-aspm-ac` mais en mode batterie.  
La valeur par défaut est `"powersave"`.
- peut-être-entier-non-négatif** [paramètre de tlp-configuration]  
**start-charge-thresh-bat0**  
Pourcentage à partir duquel la batterie 0 doit commencer à charger. Seulement pris en charge sur les ordinateurs portables.  
La valeur par défaut est `'disabled'`.
- peut-être-entier-non-négatif** [paramètre de tlp-configuration]  
**stop-charge-thresh-bat0**  
Pourcentage à partir duquel la batterie 0 devrait arrêter de charger. Seulement pris en charge sur les ordinateurs portables.  
La valeur par défaut est `'disabled'`.

- peut-être-entier-non-négatif** [paramètre de tlp-configuration]  
**start-charge-thresh-bat1**  
Pourcentage à partir duquel la batterie 1 devrait commencer à charger. Seulement pris en charge sur les ordinateurs portables.  
La valeur par défaut est 'disabled'.
- peut-être-entier-non-négatif** [paramètre de tlp-configuration]  
**stop-charge-thresh-bat1**  
Pourcentage à partir duquel la batterie 1 devrait arrêter de charger. Seulement pris en charge sur les ordinateurs portables.  
La valeur par défaut est 'disabled'.
- string radeon-power-profile-on-ac** [paramètre de tlp-configuration]  
Niveau de vitesse de l'horloge des cartes graphiques Radeon. Les possibilités sont low, mid, high, auto et default.  
La valeur par défaut est "high".
- string radeon-power-profile-on-bat** [paramètre de tlp-configuration]  
Comme radeon-power-ac mais en mode batterie.  
La valeur par défaut est "low".
- string radeon-dpm-state-on-ac** [paramètre de tlp-configuration]  
Méthode de gestion de l'énergie dynamique de Radeon (DPM). Les possibilités sont battery et performance.  
La valeur par défaut est "performance".
- string radeon-dpm-state-on-bat** [paramètre de tlp-configuration]  
Comme radeon-dpm-state-ac mais en mode batterie.  
La valeur par défaut est "battery".
- string radeon-dpm-perf-level-on-ac** [paramètre de tlp-configuration]  
Niveau de performance de DPM. Les possibilités sont auto, low et high.  
La valeur par défaut est "auto".
- string radeon-dpm-perf-level-on-bat** [paramètre de tlp-configuration]  
Comme radeon-dpm-perf-ac mais en mode batterie.  
La valeur par défaut est "auto".
- on-off-boolean wifi-pwr-on-ac?** [paramètre de tlp-configuration]  
Mode de gestion de l'énergie wifi.  
La valeur par défaut est '#f'.
- on-off-boolean wifi-pwr-on-bat?** [paramètre de tlp-configuration]  
Comme wifi-power-ac? mais en mode batterie.  
La valeur par défaut est '#t'.
- y-n-boolean wol-disable?** [paramètre de tlp-configuration]  
Désactive wake on LAN.  
La valeur par défaut est '#t'.

- entier-non-négatif** [paramètre de tlp-configuration]  
**sound-power-save-on-ac**  
 Durée d'attente en secondes avant d'activer la gestion de l'énergie audio sur les périphériques Intel HDA et AC97. La valeur 0 désactive la gestion de l'énergie.  
 La valeur par défaut est '0'.
- entier-non-négatif** [paramètre de tlp-configuration]  
**sound-power-save-on-bat**  
 Comme **sound-powersave-ac** mais en mode batterie.  
 La valeur par défaut est '1'.
- y-n-boolean** [paramètre de tlp-configuration]  
**sound-power-save-controller?**  
 Désactive le contrôleur en mode de gestion de l'énergie sur les périphériques Intel HDA.  
 La valeur par défaut est '#t'.
- boolean bay-poweroff-on-bat?** [paramètre de tlp-configuration]  
 Active le périphérique optique AltraBay/MediaBay en mode batterie. Le périphérique peut être de nouveau alimenté en lâchant (et en réinsérant) le levier d'éjection ou en appuyant sur le bouton d'éjection sur les modèles plus récents.  
 La valeur par défaut est '#f'.
- string bay-device** [paramètre de tlp-configuration]  
 Nom du périphérique optique à éteindre.  
 La valeur par défaut est "sr0".
- string runtime-pm-on-ac** [paramètre de tlp-configuration]  
 Gestion de l'énergie à l'exécution sur les bus PCI(e). Les possibilités sont on et auto.  
 La valeur par défaut est "on".
- string runtime-pm-on-bat** [paramètre de tlp-configuration]  
 Comme **runtime-pm-ac** mais en mode batterie.  
 La valeur par défaut est "auto".
- boolean runtime-pm-all?** [paramètre de tlp-configuration]  
 Gestion de l'énergie à l'exécution pour tous les bus PCI(e), sauf ceux en liste noire.  
 La valeur par défaut est '#t'.
- peut-être-liste-de-chaines-séparées-par-des-espaces** [paramètre de tlp-configuration]  
**runtime-pm-blacklist**  
 Exclue les adresses des périphériques PCI(e) spécifiés de la gestion de l'énergie à l'exécution.  
 La valeur par défaut est 'disabled'.
- space-separated-string-list** [paramètre de tlp-configuration]  
**runtime-pm-driver-blacklist**  
 Exclue les périphériques PCI(e) assignés aux pilotes spécifiés de la gestion de l'énergie à l'exécution.



- boolean** `usb-autosuspend?` [paramètre de `tlp-configuration`]  
Active la fonctionnalité de mise en veille automatique de l'USB.  
La valeur par défaut est '#t'.
- peut-être-chaine** `usb-blacklist` [paramètre de `tlp-configuration`]  
Exclue les périphériques spécifiés de la mise en veille automatique de l'USB.  
La valeur par défaut est 'disabled'.
- boolean** `usb-blacklist-wwan?` [paramètre de `tlp-configuration`]  
Exclue les périphériques WWAN de la mise en veille automatique de l'USB.  
La valeur par défaut est '#t'.
- peut-être-chaine** `usb-whitelist` [paramètre de `tlp-configuration`]  
Inclue les périphériques spécifiés dans la mise en veille automatique de l'USB, même s'ils sont déjà exclus par le pilote ou via `usb-blacklist-wwan?`.  
La valeur par défaut est 'disabled'.
- peut-être-booléen** `usb-autosuspend-disable-on-shutdown?` [paramètre de `tlp-configuration`]  
Active la mise en veille de l'USB avant l'arrêt.  
La valeur par défaut est 'disabled'.
- boolean** `restore-device-state-on-startup?` [paramètre de `tlp-configuration`]  
Restaure l'état des périphériques radio (bluetooth, wifi, wwan) du dernier arrêt au démarrage du système.  
La valeur par défaut est '#f'.

## Démon Thermald

Le module (`gnu services pm`) fournit une interface pour `thermald`, un service de gestion de l'horloge CPU qui aide à éviter la surchauffe.

- thermald-service-type** [Variable Scheme]  
C'est le type de service pour `thermald` (<https://01.org/linux-thermal-daemon/>), le démon de température de Linux, responsable du contrôle de l'état thermique des processeurs et d'éviter la surchauffe.
- thermald-configuration** [Type de données]  
Type de données représentant la configuration de `thermald-service-type`.
- `adptative?` (par défaut : #f)  
Utilise les tables adaptatives DPTF (Dynamic Power and Thermal Framework) si elles sont présentes.
- `ignore-cpuid-check?` (par défaut : #f)  
Ignore la vérification des modèles CPU supportés avec `cpuid`.
- `thermald` (par défaut : `thermald`)  
Objet du paquet de `thermald`.

### 12.9.28 Services audio

Le module (`gnu services audio`) fournit un service qui lance MPD (le démon de lecture de musique).

#### Music Player Daemon

Le démon de lecture de musique (MPD) est un service qui joue de la musique tout en étant contrôlé depuis la machine locale ou à travers le réseau par divers clients.

L'exemple suivant montre comment on peut lancer `mpd` en tant qu'utilisateur "`bob`" sur le port `6666`. Il utilise `pulseaudio` pour la sortie audio.

```
(service mpd-service-type
  (mpd-configuration
    (user "bob")
    (port "6666")))
```

`mpd-service-type` [Variable Scheme]  
Le type de service pour `mpd`

`mpd-configuration` [Type de données]  
Type de données représentant la configuration de `mpd`.

`user` (par défaut : "`mpd`")  
L'utilisateur qui lance `mpd`.

`music-dir` (par défaut : "`~/Music`")  
Le répertoire à scanner pour trouver les fichiers de musique.

`playlist-dir` (par défaut : "`~/mpd/playlists`")  
Le répertoire où stocker les playlists.

`db-file` (par défaut : "`~/mpd/tag_cache`")  
Emplacement de la base de données de musiques.

`state-file` (par défaut : "`~/mpd/state`")  
Emplacement du fichier qui stocke l'état actuel de MPD.

`sticker-file` (par défaut : "`~/mpd/sticker.sql`")  
Emplacement de la base de données de stickers.

`port` (par défaut : "`6600`")  
Le port sur lequel lancer `mpd`.

`address` (par défaut : "`any`")  
L'adresse sur laquelle se lie `mpd`. Pour utiliser un socket Unix domain, un chemin absolu peut être spécifié ici.

`outputs` (par défaut : "`(list (mpd-output))`")  
Les sorties audio que MPD peut utiliser. Par défaut c'est une seule sortie audio utilisant `pulseaudio`.

`mpd-output` [Type de données]  
Type de données représentant une sortie audio de `mpd`.

`name` (par défaut : "`MPD`")  
Le nom de la sortie audio.

- type** (par défaut : "pulse")  
Le type de sortie audio.
- enabled?** (par défaut : #t)  
Spécifie si cette sortie audio est activée au démarrage de MPD. Par défaut, toutes les sorties audio sont activées. C'est le paramètre par défaut s'il n'y a pas de fichier d'état ; avec un fichier d'état, l'état précédent est restauré.
- tags?** (par défaut : #t)  
Si la valeur est #f, MPD n'enverra pas les tags à cette sortie. C'est utile uniquement pour les greffons de sortie qui peuvent recevoir les tags, comme le greffon de sortie `httpd`.
- always-on?** (par défaut : #f)  
Si la valeur est #t, alors MPD essaye de garder cette sortie audio toujours ouverte. Cela peut être utile pour les serveurs de streaming, lorsque vous ne voulez pas déconnecter tous les clients même si la lecture est accidentellement interrompue.
- mixer-type**  
Ce champ accepte un symbole qui spécifie quel mixeur utiliser pour cette sortie audio : le mixeur `hardware`, le mixeur `software`, le mixeur `null` (permet de gérer le volume, mais sans effet ; peut être utilisé pour implémenter un mixeur externe) ou aucun mixeur (`none`).
- extra-options** (par défaut : '() )  
Une liste d'association de symboles d'options à des valeurs à ajouter à la configuration de la sortie audio.

L'exemple suivant montre comment configurer `mpd` pour fournir une sortie de streaming audio HTTP.

```
(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 Services de virtualisation

Le module (`gnu services virtualization`) fournit des services pour les démons `libvirt` et `virtlog`, ainsi que d'autres services liés à la virtualisation.

## Démon libvirt

`libvirtd` est le démon côté serveur du système de gestion de virtualisation libvirt. Ce démon tourne sur des serveurs hôtes et effectue les tâches de gestion requises pour les clients virtualisés.

`libvirt-service-type` [Variable Scheme]

C'est le type du démon libvirt (<https://libvirt.org>). Sa valeur doit être un `libvirt-configuration`.

```
(service libvirt-service-type
  (libvirt-configuration
    (unix-sock-group "libvirt")
    (tls-port "16555")))
```

Les champs de `libvirt-configuration` disponibles sont :

`package libvirt` [paramètre de `libvirt-configuration`]

Paquet libvirt.

`boolean listen-tls?` [paramètre de `libvirt-configuration`]

Indique s'il faut écouter des connexions TLS sécurisées sur le port TCP/IP public. Vous devez remplir le champ `listen` pour que cela ait un effet.

Il est nécessaire de mettre en place une CA et de créer un certificat serveur avant d'utiliser cette fonctionnalité.

La valeur par défaut est `'#t'`.

`boolean listen-tcp?` [paramètre de `libvirt-configuration`]

Écoute des connexions non-chiffrées sur le port TCP/IP public. Vous devez remplir le champ `listen` pour que cela ait un effet.

L'utilisation des sockets TCP requiert une authentification SASL par défaut. Seuls les mécanismes SASL qui prennent en charge le chiffrement des données sont permis. Il s'agit de `DIGEST_MD5` et `GSSAPI` (Kerberos5).

La valeur par défaut est `'#f'`.

`string tls-port` [paramètre de `libvirt-configuration`]

Port pour accepter les connexions TLS sécurisées. Il peut s'agir d'un numéro de port ou d'un nom de service.

La valeur par défaut est `"16514"`.

`string tcp-port` [paramètre de `libvirt-configuration`]

Port sur lequel accepter les connexions TCP non sécurisées. Cela peut être un numéro de port ou un nom de service.

La valeur par défaut est `"16509"`.

`string listen-addr` [paramètre de `libvirt-configuration`]

Adresse IP ou nom d'hôte utilisé pour les connexions des clients.

La valeur par défaut est `"0.0.0.0"`.

- boolean** `mdns-adv?` [paramètre de `libvirt-configuration`]  
Indique s'il faut annoncer le service `libvirt` en mDNS.  
Autrement, vous pouvez désactiver cela pour tous les services en stoppant le démon `Avahi`.  
La valeur par défaut est `'#f'`.
- string** `mdns-name` [paramètre de `libvirt-configuration`]  
Nom annoncé par défaut sur mDNS. Cela doit être unique sur le réseau local.  
La valeur par défaut est `"Virtualization Host <hostname>"`.
- string** `unix-sock-group` [paramètre de `libvirt-configuration`]  
Groupe propriétaire du socket Unix domain. Cela peut être utilisé pour permettre à un ensemble d'utilisateurs « de confiance » de gérer les fonctionnalités sans devenir `root`.  
La valeur par défaut est `"root"`.
- string** `unix-sock-ro-perms` [paramètre de `libvirt-configuration`]  
Permission Unix pour le socket en lecture seule. Il est utilisé pour surveiller le statut des VM uniquement.  
La valeur par défaut est `"0777"`.
- string** `unix-sock-rw-perms` [paramètre de `libvirt-configuration`]  
Permission Unix pour le socket en lecture-écriture. La valeur par défaut n'autorise que `root`. Si `PolicyKit` est activé sur le socket, la valeur par défaut change et permet tout le monde (c.-à-d. `0777`)  
La valeur par défaut est `"0770"`.
- string** `unix-sock-admin-perms` [paramètre de `libvirt-configuration`]  
Permissions Unix pour le socket d'administration. La valeur par défaut ne permet que le propriétaire (`root`), ne la changez pas à moins que vous ne soyez sûr de savoir à qui vous exposez cet accès.  
La valeur par défaut est `"0777"`.
- string** `unix-sock-dir` [paramètre de `libvirt-configuration`]  
Le répertoire dans lequel les sockets sont créés.  
La valeur par défaut est `"/var/run/libvirt"`.
- string** `auth-unix-ro` [paramètre de `libvirt-configuration`]  
Schéma d'authentification pour les socket Unix en lecture-seule. Par défaut les permissions des socket permettent à n'importe qui de se connecter  
La valeur par défaut est `"polkit"`.
- string** `auth-unix-rw` [paramètre de `libvirt-configuration`]  
Schéma d'authentification pour les socket UNIX en lecture-écriture. Par défaut les permissions du socket ne permettent que `root`. Si le support de `PolicyKit` a été compilé dans `libvirt`, la valeur par défaut utilise l'authentification « `polkit` ».  
La valeur par défaut est `"polkit"`.

- string auth-tcp** [paramètre de `libvirt-configuration`]  
Schéma d'authentification pour les sockets TCP. Si vous n'avez pas activé SASL, alors tout le trafic TCP est en clair. Ne le faites pas en dehors de scénario de développement ou de test.  
La valeur par défaut est `"sas1"`.
- string auth-tls** [paramètre de `libvirt-configuration`]  
Schéma d'authentification pour les sockets TLS. Les sockets TLS sont déjà chiffrés par la couche TLS, et une authentification limitée est effectuée avec les certificats.  
Il est possible d'utiliser de n'importe quel mécanisme d'authentification SASL en utilisant « `sasl` » pour cette option  
La valeur par défaut est `"none"`.
- optional-list access-drivers** [paramètre de `libvirt-configuration`]  
Schéma de contrôle d'accès à l'API.  
Par défaut un utilisateur authentifié peut accéder à toutes les API. Les pilotes d'accès peuvent placer des restrictions là-dessus.  
La valeur par défaut est `('')`.
- string key-file** [paramètre de `libvirt-configuration`]  
Chemin de fichier de la clef du serveur. Si la valeur est une chaîne vide, aucune clef privée n'est chargée.  
La valeur par défaut est `""`.
- string cert-file** [paramètre de `libvirt-configuration`]  
Chemin de fichier de la clef du serveur. Si la chaîne est vide, aucun certificat n'est chargé.  
La valeur par défaut est `""`.
- string ca-file** [paramètre de `libvirt-configuration`]  
Chemin de fichier de la clef du serveur. Si la chaîne est vide, aucun certificat de CA n'est chargé.  
La valeur par défaut est `""`.
- string crl-file** [paramètre de `libvirt-configuration`]  
Chemin de la liste de révocation des certificats. Si la chaîne est vide, aucun CRL n'est chargé.  
La valeur par défaut est `""`.
- boolean tls-no-sanity-cert** [paramètre de `libvirt-configuration`]  
Désactive la vérification de nos propres certificats serveurs.  
Lorsque `libvirtd` démarre il effectue des vérifications de routine sur ses propres certificats.  
La valeur par défaut est `#f`.

- boolean** `tls-no-verify-cert` [paramètre de `libvirt-configuration`]  
Désactive la vérification des certificats clients.  
La vérification des certificats clients est le mécanisme d'authentification principal.  
Tout client qui ne présent pas de certificat signé par la CA sera rejeté.  
La valeur par défaut est `'#f'`.
- optional-list** `tls-allowed-dn-list` [paramètre de `libvirt-configuration`]  
Liste blanche des Distinguished Name x509 autorisés.  
La valeur par défaut est `'()''`.
- optional-list** `sasl-allowed-usernames` [paramètre de `libvirt-configuration`]  
Liste blanche des noms d'utilisateur SASL permis. Le format des noms d'utilisateurs dépend du mécanisme d'authentification SASL.  
La valeur par défaut est `'()''`.
- string** `tls-priority` [paramètre de `libvirt-configuration`]  
Modifie la chaîne de priorité TLS par défaut fixée à la compilation. La valeur par défaut est typiquement `'NORMAL'` à moins qu'elle n'ait été modifiée à la compilation. Ne l'indiquez que si vous voulez que libvirt agisse différemment des paramètres par défaut globaux.  
La valeur par défaut est `"NORMAL"`.
- integer** `max-clients` [paramètre de `libvirt-configuration`]  
Nombre maximum de connexions clientes en même temps sur tous les sockets.  
La valeur par défaut est `'5000'`.
- integer** `max-queued-clients` [paramètre de `libvirt-configuration`]  
Longueur maximum de la queue de connexions en attente d'acceptation du démon. Remarquez que certains protocoles supportant la retransmission peuvent obéir à ce paramètre pour qu'une connexion ultérieure réussisse.  
La valeur par défaut est `'1000'`.
- integer** `max-anonymous-clients` [paramètre de `libvirt-configuration`]  
Longueur maximum de la queue des clients acceptés mais pas authentifiés. Indiquez zéro pour désactiver ce paramètre  
La valeur par défaut est `'20'`.
- integer** `min-workers` [paramètre de `libvirt-configuration`]  
Nombre de processus de travail démarrés initialement.  
La valeur par défaut est `'5'`.
- integer** `max-workers` [paramètre de `libvirt-configuration`]  
Nombre maximum de threads de travail.  
Si le nombre de clients actifs dépasse `min-workers`, plus de threads seront démarrés, jusqu'à la limite de `max-workers`. Typiquement vous voulez que `max-workers` soit égal au nombre maximum de clients permis.  
La valeur par défaut est `'20'`.

- integer prio-workers** [paramètre de libvirt-configuration]  
Nombre de travailleurs prioritaires. Si tous les threads de travail du groupe ci-dessus sont bloqués, certains appels marqués comme prioritaires (notamment `domainDestroy`) peuvent être exécutés par ce groupe.  
La valeur par défaut est '5'.
- integer max-requests** [paramètre de libvirt-configuration]  
Limite globale totale sur les appels RPC concurrents.  
La valeur par défaut est '20'.
- integer max-client-requests** [paramètre de libvirt-configuration]  
Limite de requêtes concurrentes depuis une connexion cliente unique. Pour éviter qu'un client ne monopolise le serveur, vous devriez indiquer une petite partie des paramètres `global max-requests` et `max-workers`.  
La valeur par défaut est '5'.
- integer admin-min-workers** [paramètre de libvirt-configuration]  
Comme `min-workers` mais pour l'interface d'administration.  
La valeur par défaut est '1'.
- integer admin-max-workers** [paramètre de libvirt-configuration]  
Comme `max-workers` mais pour l'interface d'administration.  
La valeur par défaut est '5'.
- integer admin-max-clients** [paramètre de libvirt-configuration]  
Comme `max-clients` mais pour l'interface d'administration.  
La valeur par défaut est '5'.
- integer admin-max-queued-clients** [paramètre de libvirt-configuration]  
Comme `max-queued-clients` mais pour l'interface d'administration.  
La valeur par défaut est '5'.
- integer admin-max-client-requests** [paramètre de libvirt-configuration]  
Comme `max-client-requests` mais pour l'interface d'administration.  
La valeur par défaut est '5'.
- integer log-level** [paramètre de libvirt-configuration]  
Niveau de journalisation. 4 : erreurs, 3 : avertissements, 2 : information, 1 : débogage.  
La valeur par défaut est '3'.
- string log-filters** [paramètre de libvirt-configuration]  
Filtres de journalisation.  
Un filtre qui permet de sélectionner un niveau de journalisation différent pour une catégorie donnée. Le format d'un filtre est :
- x:nom
  - x:+nom



où `nom` est une chaîne de caractères qui correspond à la catégorie donnée dans `VIR_LOG_INIT()` au début de chaque fichier source de `libvirt`, p. ex. `remote`, `qemu` ou `util.json` (le nom dans le filtre peut être une sous-chaîne du nom complet de la catégorie, pour pouvoir correspondre à plusieurs catégories similaires), le préfixe facultatif `+` dit à `libvirt` d'enregistrer les traces de piles pour chaque message qui correspond au nom, et `x` est le niveau minimal des messages qui devraient être enregistrés :

- 1 : DEBUG
- 2 : INFO
- 3 : WARNING
- 4 : ERROR

On peut définir plusieurs filtres dans une seule déclaration de filtres, ils doivent juste être séparés par des espaces.

La valeur par défaut est `"3:remote 4:event"`.

**string** `log-outputs` [paramètre de `libvirt-configuration`]

Sorties de débogage.

Une sortie est l'un des endroits où les journaux sont enregistrés. Le format d'une sortie peut être :

`x:stderr` la sortie va vers `stderr`

`x:syslog:nom`  
utilise `syslog` comme sortie et utilise le nom donné comme identifiant

`x:file:chemin_fichier`  
la sortie va vers un fichier, avec le chemin donné

`x:journald`  
la sortie va vers le système de journalisation `journald`

Dans tous les cas, le préfixe `x` est le niveau minimal, qui agit comme un filtre

- 1 : DEBUG
- 2 : INFO
- 3 : WARNING
- 4 : ERROR

Plusieurs sorties peuvent être définies, elles doivent juste être séparées par des espaces.

La valeur par défaut est `"3:stderr"`.

**integer** `audit-level` [paramètre de `libvirt-configuration`]

Permet de modifier l'utilisation du sous-système d'audit

- 0 : désactive tout audit
- 1 : active l'audit, seulement s'il est activé sur l'hôte
- 2 : active l'audit, et quitte s'il est désactivé sur l'hôte.

La valeur par défaut est `'1'`.

**boolean audit-logging** [paramètre de libvirt-configuration]  
Envoie les messages d'audit via l'infrastructure de journalisation de libvirt.  
La valeur par défaut est '#f'.

**optional-string host-uuid** [paramètre de libvirt-configuration]  
UUID de l'hôte. L'UUID ne doit pas avoir tous ses nombres identiques.  
La valeur par défaut est "".

**string host-uuid-source** [paramètre de libvirt-configuration]  
Source où lire l'UUID de l'hôte.

- **smbios** : récupère l'UUID à partir de `dmidecode -s system-uuid`
- **machine-id** : récupère l'UUID à partir de `/etc/machine-id`

Si `dmidecode` ne fournit pas un UUID valide, un UUID temporaire sera généré.  
La valeur par défaut est "smbios".

**integer keepalive-interval** [paramètre de libvirt-configuration]  
Un message keepalive est envoyé au client après `keepalive_interval` secondes d'inactivité pour vérifier si le client répond toujours. Si la valeur est -1, libvirtd n'enverra jamais de requête keepalive ; cependant les clients peuvent toujours en envoyer et le démon y répondra.  
La valeur par défaut est '5'.

**integer keepalive-count** [paramètre de libvirt-configuration]  
Nombre maximum de messages keepalive qui peuvent être envoyés au client sans réponse avant que la connexion ne soit considérée comme cassée.  
En d'autres termes, la connexion est approximativement fermée après `keepalive_interval * (keepalive_count + 1)` secondes après le dernier message reçu de la part du client. Lorsque `keepalive-count` est à 0, les connexions seront automatiquement fermées après `keepalive-interval` secondes d'inactivité sans envoyer le moindre message keepalive.  
La valeur par défaut est '5'.

**integer admin-keepalive-interval** [paramètre de libvirt-configuration]  
Comme précédemment, mais pour l'interface d'administration.  
La valeur par défaut est '5'.

**integer admin-keepalive-count** [paramètre de libvirt-configuration]  
Comme précédemment, mais pour l'interface d'administration.  
La valeur par défaut est '5'.

**integer ovs-timeout** [paramètre de libvirt-configuration]  
Délai d'attente pour les appels Open vSwitch.  
L'utilitaire `ovs-vsctl` est utilisé pour la configuration et son option de délai d'attente est à 5 secondes pour éviter qu'une attente infinie ne bloque libvirt.  
La valeur par défaut est '5'.

## Démon Virrlog

Le service `virtlogd` est un démon côté serveur qui fait partie de `libvirt`, utilisé pour gérer les journaux des consoles des machines virtuelles.

Ce démon n'est pas utilisé directement par les clients `libvirt`, mais il est appelé pour eux par `libvirtd`. En maintenant les journaux dans un démon séparé, le démon `libvirtd` principal peut être redémarré sans risque de perte de journaux. Le démon `virtlogd` a la possibilité de ré-exécuter `exec()` sur lui-même quand il reçoit `SIGUSR1`, pour permettre des mises à jour à chaud sans temps mort.

**virtlog-service-type** [Variable Scheme]  
Le type de service pour le démon `virtlogd`. Sa valeur doit être un `virtlog-configuration`.

```
(service virtlog-service-type
  (virtlog-configuration
    (max-clients 1000)))
```

**paramètre de libvirt package** *libvirt* [Variable]  
Paquet `libvirt`.

**integer log-level** [paramètre de `virtlog-configuration`]  
Niveau de journalisation. 4 : erreurs, 3 : avertissements, 2 : information, 1 : débogage.  
La valeur par défaut est '3'.

**string log-filters** [paramètre de `virtlog-configuration`]  
Filtres de journalisation.

Un filtre qui permet de sélectionner plusieurs niveaux de journalisation pour une catégorie donnée. Le format d'un filtre est :

- `x:nom`
- `x:+nom`

où `nom` est une chaîne de caractères qui correspond à la catégorie donnée dans `VIR_LOG_INIT()` au début de chaque fichier source de `libvirt`, p. ex. « `remote` », « `qemu` » ou « `util.json` » (le nom dans le filtre peut être une sous-chaîne du nom complet de la catégorie, pour pouvoir correspondre à plusieurs catégories similaires), le préfixe facultatif « `+` » dit à `libvirt` d'enregistrer les traces de piles pour chaque message qui correspond au nom, et `x` est le niveau minimal des messages qui devraient être enregistrés :

- 1 : DEBUG
- 2 : INFO
- 3 : WARNING
- 4 : ERROR

On peut définir plusieurs filtres dans une seule déclaration de filtres, ils doivent juste être séparés par des espaces.

La valeur par défaut est "`3:remote 4:event`".

**string log-outputs** [paramètre de virtlog-configuration]

Sorties de débogage.

Une sortie est l'un des endroits où les journaux sont enregistrés. Le format d'une sortie peut être :

**x:stderr** la sortie va vers stderr

**x:syslog:nom**

utilise syslog comme sortie et utilise le nom donné comme identifiant

**x:file:chemin\_fichier**

la sortie va vers un fichier, avec le chemin donné

**x:journald**

la sortie va vers le système de journalisation journald

Dans tous les cas, le préfixe x est le niveau minimal, qui agit comme un filtre

- 1 : DEBUG
- 2 : INFO
- 3 : WARNING
- 4 : ERROR

Plusieurs sorties peuvent être définies, elles doivent juste être séparées par des espaces.

La valeur par défaut est `"3:stderr"`.

**integer max-clients** [paramètre de virtlog-configuration]

Nombre maximum de connexions clientes en même temps sur tous les sockets.

La valeur par défaut est `'1024'`.

**integer max-size** [paramètre de virtlog-configuration]

Taille de fichier maximale avant roulement.

La valeur par défaut est `'2MB'`

**integer max-backups** [paramètre de virtlog-configuration]

Nombre maximal de fichiers de sauvegardes à garder.

La valeur par défaut est `'3'`

## Émulation transparente avec QEMU

`qemu-binfmt-service-type` fournit le support de l'émulation transparente de binaires construits pour des architectures différentes — p. ex. il permet d'exécuter de manière transparente des programmes ARMv7 sur une machine x86\_64. Cela se fait en combinant l'émulateur QEMU (<https://www.qemu.org>) et la fonctionnalité `binfmt_misc` du noyau Linux. Cette fonctionnalité ne vous permet que d'émuler GNU/Linux sur une architecture différente, mais regardez plus bas pour GNU/Hurd.

**qemu-binfmt-service-type** [Variable Scheme]

Le type du service QEMU/binfmt pour l'émulation transparente. Sa valeur doit être un objet `qemu-binfmt-configuration`, qui spécifie le paquet QEMU à utiliser ainsi que l'architecture que vous voulez émuler :

```
(service qemu-binfmt-service-type
```

```
(qemu-binfmt-configuration
  (platforms (lookup-qemu-platforms "arm" "aarch64"))))
```

Dans cet exemple, on active l'émulation transparente pour les plateformes ARM et aarch64. Lancer `herd stop qemu-binfmt` l'éteint et lancer `herd start qemu-binfmt` le rallume (voir Section "Invoking herd" dans *The GNU Shepherd Manual*).

**qemu-binfmt-configuration** [Type de données]

La configuration du service `qemu-binfmt`.

**platforms** (par défaut : '() )

La liste des plates-formes émulées par QEMU. Chaque élément doit être un objet *platform object* tel que renvoyé par `lookup-qemu-platforms` (voir plus bas).

Par exemple, supposons que vous soyez sur une machine x86\_64 et que vous avez ce services :

```
(service qemu-binfmt-service-type
  (qemu-binfmt-configuration
    (platforms (lookup-qemu-platforms "arm"))))
```

Vous pouvez lancer :

```
guix build -s armhf-linux inkscape
```

et cela construira Inkscape pour ARMv7 *comme s'il s'agissait d'une construction native*, de manière transparente avec QEMU pour émuler un CPU ARMv7. Plutôt pratique si vous voulez tester un paquet construit pour une architecture à laquelle vous n'avez pas accès !

**qemu** (par défaut : `qemu`)

Le paquet QEMU à utiliser.

**lookup-qemu-platforms** *platforms...* [Procédure Scheme]

Renvoie la liste des objets de plates-formes QEMU correspondant à *platforms...* *platforms* doit être une liste de chaînes de caractères correspondant aux noms de plates-formes, comme "arm", "sparc", "mips64el" etc.

**qemu-platform?** *obj* [Procédure Scheme]

Renvoie vrai si *obj* est un objet de plate-forme.

**qemu-platform-name** *platform* [Procédure Scheme]

Renvoie le nom de *platform* — une chaîne comme "arm".

## Agent invité de QEMU

L'agent invité de QEMU permet de contrôler le système émulé à partir de l'hôte. Le service `qemu-guest-agent` lance l'agent sur les invités Guix. Pour contrôler l'agent à partir de l'hôte, ouvrez un socket en invoquant QEMU avec les arguments suivants :

```
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 \
...
```

Cela crée un socket dans `/tmp/qga.sock` sur l'hôte. Une fois l'agent invité lancé vous pouvez exécuter des commandes avec `socat` :

```
$ guix shell socat -- socat unix-connect:/tmp/qga.sock stdio
{"execute": "guest-get-host-name"}
{"return": {"host-name": "guix"}}
```

Voir la documentation de l'agent invité de QEMU (<https://wiki.qemu.org/Features/GuestAgent>) pour les options et les commandes supplémentaires.

`qemu-guest-agent-service-type` [Variable Scheme]

Type de service pour l'agent invité de QEMU.

`qemu-guest-agent-configuration` [Type de données]

La configuration du service `qemu-guest-agent`.

`qemu` (par défaut : `qemu-minimal`)

Le paquet QEMU à utiliser.

`device` (par défaut : `""`)

Le nom de périphérique ou de socket que l'agent utilise pour communiquer avec l'hôte. Si le nom est vide, QEMU utilisera un nom de fichier par défaut.

## Exécuter le Hurd dans une machine virtuelle

Le service `hurd-vm` permet de lancer GNU/Hurd dans une machine virtuelle (VM), un *childhurd*. Ce service est conçu pour être utilisé sur GNU/Linux et la configuration du système GNU/Hurd donné est compilée de manière croisée. La machine virtuelle est un service Shepherd qui a les noms `hurd-vm` et `childhurd` et qui peut être contrôlé avec les commandes suivantes :

```
herd start hurd-vm
herd stop childhurd
```

Lorsque le service est lancé, vous pouvez voir sa console en vous y connectant avec un client VNC, par exemple avec :

```
guix shell tigervnc-client -- vncviewer localhost:5900
```

La configuration par défaut (voir `hurd-vm-configuration` ci-dessous) crée un serveur secure shell (SSH) dans votre système GNU/Hurd, que QEMU (l'émulateur de machine virtuelle) redirige sur le port 10222 de l'hôte. Ainsi, vous pouvez vous connecter en SSH au `childhurd` avec :

```
ssh root@localhost -p 10022
```

Le `childhurd` est volatile et sans état . il démarre avec un nouveau système de fichier à chaque fois que vous le redémarrez. Par défaut cependant, tous les fichiers sous `/etc/childhurd` sur l'hôte sont copiés tels quels sur le système de fichiers racine du `childhurd` à son démarrage. Cela vous permet d'initialiser des « secrets » à l'intérieur de la VM : les clés hôtes SSH, les clés de substituts autorisés, etc — voir l'explication de `secret-root` ci-dessous.

**hurdm-vm-service-type** [Variable Scheme]

C'est le type du service du Hurd dans une machine virtuelle. Sa valeur doit être un objet `hurdm-vm-configuration`, qui spécifie le système d'exploitation (voir Section 12.2 [référence de operating-system], page 246) et la taille de disque pour la machine virtuelle du Hurd, le paquet QEMU à utiliser ainsi que les options pour le lancer.

Par exemple :

```
(service hurdm-vm-service-type
  (hurdm-vm-configuration
    (disk-size (* 5000 (expt 2 20))) ;5G
    (memory-size 1024)) ;1024MiB
```

créerait une image disque assez grande pour construire GNU Hello, avec une peu de place en plus.

**hurdm-vm-configuration** [Type de données]

Type de données représentant la configuration de `hurdm-vm-service-type`.

**os** (par défaut : `%hurdm-vm-operating-system`)

Le système d'exploitation à instancier. La valeur par défaut est le système minimal avec un démon OpenSSH permissif en écoute sur le port 2222 (voir Section 12.9.5 [Services réseau], page 299).

**qemu** (par défaut : `qemu-minimal`)

Le paquet QEMU à utiliser.

**image** (par défaut : `hurdm-vm-disk-image`)

La procédure utilisée pour construire l'image disque construite à partir de cette configuration.

**disk-size** (par défaut : `'guess`)

La taille de l'image disque.

**memory-size** (par défaut : `512`)

La taille de mémoire de la machine virtuelle en mébioctets.

**options** (par défaut : `'("--snapshot")`)

Options supplémentaires pour lancer QEMU.

**id** (par défaut : `#f`)

Si l'option est indiquée, c'est un entier strictement positif utilisé créer plusieurs instances de `childhurd`. Il est ajouté au nom du service, p. ex. `childhurd1`.

**net-options** (par défaut : `hurdm-vm-net-options`)

La procédure utilisée pour produire une liste d'options réseau pour QEMU.

Par défaut, elle produit

```
'("--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")
```

avec les ports renvoyés :

```
secrets-port: (+ 11004 (* 1000 ID))
ssh-port: (+ 10022 (* 1000 ID))
vnc-port: (+ 15900 (* 1000 ID))
```

`secret-root` (par défaut : `/etc/childhurd`)

Le répertoire racine avec des secrets externes à installer dans le `childhurd` une fois lancé. Les `childhurds` sont volatile, ce qui signifie qu'à chaque démarrage, les secrets comme les clés hôtes SSH et la clé de signature de Guix sont recréés.

Si le répertoire `/etc/childhurd` n'existe pas, le `secret-service` qui tourne dans le `Childhurd` recevra une liste vide de secrets.

Par défaut, le service remplit automatiquement `/etc/childhurd` avec les secrets non-volatiles suivants, à moins qu'ils existent déjà :

```
/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
```

Ces fichiers sont automatiquement envoyés à la VM Hurd invitée au démarrage, avec les permissions.

Avec ces fichiers en place, il ne manque plus grand chose pour pouvoir permettre à l'hôte de télécharger les constructions `i586-gnu` au `childhurd` :

1. Autoriser la clé du `childhurd` sur l'hôte pour que l'hôte puisse accepter les constructions venant du `childhurd`, ce qui peut se faire de cette manière :

```
guix archive --authorize < \
  /etc/childhurd/etc/guix/signing-key.pub
```

2. Ajouter le `childhurd` à `/etc/guix/machines.scm` (voir Section 2.4.2 [Réglages du téléchargement du démon], page 12).

Nous travaillons à rendre cela automatique — contactez nous sur `guix-devel@gnu.org` pour en discuter !

Remarquez que par défaut l'image de la VM est volatile, c.-à-d. qu'une fois arrêtée le contenu est perdu. Si vous voulez une image avec état à la place, remplacez les champs `image` et `options` pour enlever le drapeau `--snapshot` avec quelque chose de ce style-là :

```
(service hurd-vm-service-type
  (hurd-vm-configuration
    (image (const "/out/of/store/writable/hurd.img")))
    (options '())))
```



## Ganeti

**Remarque:** Ce service est considéré comme étant expérimental. Les options de configuration peuvent changer de manière non compatible, et tous les paramètres n'ont pas été testés. Si vous utilisez ce service, nous vous encourageons à partager votre expérience avec [guix-devel@gnu.org](mailto:guix-devel@gnu.org).

Ganeti est un système de gestion de machines virtuelles. Il est conçu pour faire tourner les machines virtuelles en continue sur une grappe de serveurs même dans le cas d'une erreur matérielle, et pour rendre les tâches de maintenance et de récupération faciles. Il consiste en plusieurs services qui sont décrits plus loin dans cette section. En plus du service Ganeti, vous aurez besoin du service OpenSSH (voir Section 12.9.5 [Services réseau], page 299), et de mettre à jour le fichier `/etc/hosts` (voir Section 12.2 [référence de operating-system], page 246) avec le nom de la grappe et l'adresse (ou utiliser un serveur DNS).

Tous les nœuds participant dans une grappe Ganeti devraient avoir la même configuration de Ganeti et `/etc/hosts`. voici un exemple de configuration pour un nœud d'une grappe Ganeti qui prend en charge plusieurs moteurs de stockage et installe les *fournisseurs de systèmes debootstrap* et *guix* :

```
(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
"))))

;; Installe QEMU pour pouvoir utiliser les instances KVM, ainsi que LVM, DRBD et Cep
;; pour utiliser les moteurs de stockage « plain », « drbd » et « rbd ».
(packages (append (map specification->package
  ("qemu" "lvm2" "drbd-utils" "ceph"
  ;; Add the debootstrap and guix OS providers.
  "ganeti-instance-guix" "ganeti-instance-debootstrap"))
  %base-packages))

(services
  (append (list (service static-networking-service-type
    (list (static-networking
      (addresses
        (list (network-address
          (device "eth0")
          (value "192.168.1.201/24")))))
      (routes
        (list (network-route
```

```

                (destination "default")
                (gateway "192.168.1.254"))))
    (name-servers '("192.168.1.252"
                  "192.168.1.253"))))

;; Ganeti utilise SSH pour communiquer entre les nœuds.
(service openssh-service-type
  (openssh-configuration
    (permit-root-login 'prohibit-password)))

(service ganeti-service-type
  (ganeti-configuration
    ;; cette liste spécifie les chemins du système de fichiers
    ;; pour le stockage des images de machines virtuelles.
    (file-storage-paths '("/srv/ganeti/file-storage"))
    ;; Cette variable configure une « variante » pour
    ;; Debootstrap et Guix qui fonctionne avec KVM.
    (os %default-ganeti-os)))
  %base-services)))

```

Nous vous encourageons à lire le guide d'administration de Ganeti (<https://docs.ganeti.org/docs/ganeti/3.0/html/admin.html>) pour apprendre les diverses options de grappes et les opérations de base. Il y a aussi un billet de blog (<https://guix.gnu.org/blog/2020/running-a-ganeti-cluster-on-guix/>) décrivant comment configurer et initialiser une petite grappe.

**ganeti-service-type** [Variable Scheme]

C'est le type de service qui inclut tous les services dont les nœuds Ganeti ont besoin.

Sa valeur est un objet `ganeti-configuration` qui définit le paquet pour utiliser les opérations en ligne de commande, ainsi que pour les divers démons. Les chemins de stockage autorisés et les systèmes d'exploitation invités disponibles sont aussi configurés à travers ce type de données.

**ganeti-configuration** [Type de données]

Le service `ganeti` prend les options de configuration suivante :

**ganeti** (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser. Il sera installé sur le profil du système et rendra `gnt-cluster`, `gnt-instance`, etc disponibles. Remarquez que la valeur spécifiée ici n'affecte pas les autres services auxquels se réfère le paquet `ganeti` (voir plus bas).

```

noded-configuration (par défaut : (ganeti-noded-configuration))
confd-configuration (par défaut : (ganeti-confd-configuration))
wconfd-configuration (par défaut : (ganeti-wconfd-configuration))
luxid-configuration (par défaut : (ganeti-luxid-configuration))
rapi-configuration (par défaut : (ganeti-rapi-configuration))
kvmd-configuration (par défaut : (ganeti-kvmd-configuration))
mond-configuration (par défaut : (ganeti-mond-configuration))
metad-configuration (par défaut : (ganeti-metad-configuration))
watcher-configuration (par défaut : (ganeti-watcher-configuration))
cleaner-configuration (par défaut : (ganeti-cleaner-configuration))

```

Ces options contrôlent les divers démons et tâches cron distribués avec Ganeti. Les valeurs possibles sont détaillées plus bas. Pour changer un paramètre, vous devez utiliser le type de configuration pour ce service :

```

(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 (par défaut : '())
```

Liste des répertoire autorisés pour le moteur de stockage de fichiers.

```
os (par défaut : %default-ganeti-os)
```

Liste des enregistrements <ganeti-os>.

En résumé `ganeti-service-type` est un raccourci pour la déclaration de chaque service individuel :

```

(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)

```

Plus une extension de service pour `etc-service-type` qui configure le moteur de stockage de fichiers et les variantes de systèmes.

**ganeti-os** [Type de données]

Ce type de données peut être passé au paramètre `os` de `ganeti-configuration`. Il prend les paramètres suivants :

**name** Le nom du fournisseur de système. Il est seulement utilisé pour spécifier où la configuration se trouve. Indiquer « `debootstrap` » créera `/etc/ganeti/intance-debootstrap`.

**extension** (par défaut : **#f**)

L'extension de fichier pour les variantes de ce type de système. Par exemple `.conf` ou `.scm`. Elle sera ajoutée à la fin du nom de fichier si elle est indiquée.

**variants** (par défaut : `'()`)

Il doit s'agir soit d'une liste d'objets `ganeti-os-variant` pour cet OS ou un objet « simili-fichier » (voir Section 9.12 [G-Expressions], page 166) représentant le répertoire des variantes.

Pour utiliser le fournisseur Guix OS avec les définitions des variantes dans un répertoire local au lieu de déclarer des variantes individuelles (voir *guix-variants* ci-dessous), vous pouvez utiliser :

```
(ganeti-os
 (name "guix")
 (variants (local-file "ganeti-guix-variants"
                    #:recursive? #true)))
```

Remarquez que vous devrez maintenir le fichier `variants.list` (voir `ganeti-os-interface(7)` (<https://docs.ganeti.org/docs/ganeti/3.0/man/ganeti-os-interface.html>)) manuellement dans ce cas.

**ganeti-os-variant** [Type de données]

Type de données représentant une variante de système Ganeti. Il prend les paramètres suivants :

**name** Le nom de cette variante.

**configuration** Un fichier de configuration pour cette variante.

**%default-debootstrap-hooks** [Variable Scheme]

Cette variable contient les crochets pour configurer le réseau et le chargeur d'amorçage GRUB.

**%default-debootstrap-extra-pkgs** [variable Scheme]

Cette variable contient une liste de paquets requis pour un invité complètement virtualisé.

**debootstrap-configuration** [Type de données]

Ce type de données crée des fichiers de configurations pour le fournisseur de système debootstrap.

**hooks** (par défaut : `%default-debootstrap-hooks`)

Lorsque la valeur n'est pas `#f`, cela doit être une G-expression qui spécifie un répertoire avec les scripts qui seront lancés à l'installation du système. Elle peut aussi être une liste de paires de `(nom, simili-fichier)`. Par exemple :

```
`((99-hello-world . ,(plain-file "#!/bin/sh\nnecho Hello, World")))
```

Cela va créer un répertoire avec un exécutable nommé `99-hello-world` et le lancera à chaque fois que cette variante est installée. Si la valeur

est **#f**, les crochets dans `/etc/ganeti/instance-debootstrap/hooks` seront utilisés, s'ils existent.

**proxy** (par défaut : **#f**)

Serveur mandataire HTTP facultatif à utiliser.

**mirror** (par défaut : **#f**)

Le miroir Debian. Habituellement quelque chose comme `http://ftp.no.debian.org/debian`. La valeur par défaut dépend de la distribution.

**arch** (par défaut : **#f**)

L'architecture pour dpkg. Indiquez **armhf** pour debootstrap pour une instnace ARMv7 sur un hôte AArch64. La valeur par défaut est l'architecture système actuelle.

**suite** (par défaut : **"stable"**)

Lorsqu'il est indiqué, ce paramètre doit être une distribution Debian comme **buster** ou **focal**. Si la valeur est **#f**, la valeur par défaut du fournisseur de système est utilisée.

**extra-pkgs** (par défaut : **%default-debootstrap-extra-pkgs**)

Liste des paquets supplémentaires qui seront installés par dpkg en plus du système minimal.

**components** (par défaut : **#f**)

Lorsque la valeur est indiquée, doit être une liste de « composants » de répertoires Debian. Par exemple **'("main" "contrib")**.

**generate-cache?** (par défaut : **#t**)

Indique s'il faut automatiquement mettre en cache l'archive debootstrap générée.

**clean-cache** (par défaut : **14**)

Supprime le cache après ce nombre de jours. Utilisez **#f** pour ne jamais vider le cache.

**partition-style** (par défaut : **'msdos'**)

Le type de partition à créer. Lorsqu'il est indiqué, ce paramètre doit être **'msdos'**, **'none'** ou une chaîne de caractères.

**partition-alignment** (par défaut : **2048**)

Alignement des partitions en nombre de secteur.

**debootstrap-variant** *nom configuration* [Procédure Scheme]

C'est une procédure auxiliaire qui crée un enregistrement **ganeti-os-variant**. Il prend deux paramètres : un nom et un objet **debootstrap-configuration**.

**debootstrap-os variants...** [Procédure Scheme]

C'est une procédure auxiliaire qui crée un enregistrement **ganeti-os**. Elle prend une liste de variantes créé avec **debootstrap-variant**.

`guix-variant name configuration` [Procédure Scheme]

C'est une procédure auxiliaire qui crée un enregistrement `ganeti-os-variant` à utiliser avec le fournisseur de système Guix. Il prend un nom et une G-expression qui renvoie un objet simili-fichier (voir Section 9.12 [G-Expressions], page 166) contenant une configuration Guix System.

`guix-os variants...` [Procédure Scheme]

C'est une procédure auxiliaire qui crée un enregistrement `ganeti-os`. Elle prend une liste de variantes produites par `guix-variant`.

`%default-debootstrap-variants` [Variable Scheme]

C'est une variable pratique pour que le fournisseur debootstrap fonctionne directement sans avoir à déclarer des variantes manuellement. Elle contient une seule variante debootstrap avec la configuration par défaut :

```
(list (debootstrap-variant
      "default"
      (debootstrap-configuration)))
```

`%default-guix-variants` [Variable Scheme]

C'est une variable pratique pour que le fournisseur Guix fonctionne directement sans configuration supplémentaire. Elle crée une machine virtuelle qui a un serveur SSH, une console série et autorise les clés SSH des hôtes Ganeti.

```
(list (guix-variant
      "default"
      (file-append ganeti-instance-guix
                   "/share/doc/ganeti-instance-guix/examples/dynamic.scm")))
```

Les utilisateur-rices peuvent implémenter la prise en charge des fournisseurs de systèmes inconnus de Guix en étendant les enregistrements `ganeti-os` et `ganeti-os-variant` comme il faut. Par exemple :

```
(ganeti-os
 (name "custom")
 (extension ".conf")
 (variants
  (list (ganeti-os-variant
        (name "toto")
        (configuration (plain-file "titi" "this is fine"))))))
```

Cela crée `/etc/ganeti/instance-custom/variants/toto.conf` qui pointe vers un fichier dans le dépôt qui contient `this is fine`. Cela crée aussi `/etc/ganeti/instance-custom/variants/variants.list` qui contient `toto`.

Évidemment cela ne fonctionnera pas avec tous les fournisseurs d'OS disponibles. Si vous trouvez que cette interface est trop limitée, contactez-nous sur `guix-devel@gnu.org`.

Le reste de cette section documente les divers services inclus par `ganeti-service-type`.

`ganeti-noded-service-type` [Variable Scheme]

`ganeti-noded` est le démon responsable des fonctions spécifiques au nœud dans le système Ganeti. La valeur de ce service doit être un objet `ganeti-noded-configuration`.

**ganeti-noded-configuration** [Type de données]

La configuration du service `ganeti-noded`.

`ganeti` (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser pour ce service.

`port` (par défaut : `1811`)

Port TCP sur lequel le démon de nœud écoute les requêtes réseaux.

`address` (par défaut : `"0.0.0.0"`)

Adresse réseau sur laquelle le démon se lie. L'adresse par défaut signifie de se lier à toutes les adresse disponibles.

`interface` (par défaut : `#f`)

Si une valeur est indiquée, elle doit être une interface réseau spécifique (p. ex. `eth0`) à laquelle le démon se liera.

`max-clients` (par défaut : `20`)

Cela indique une limite du nombre de connexions clientes simultanées que le démon pourra prendre en charge. Les connexions au delà de ce nombre sont acceptées, mais aucune réponse ne sera envoyée avant que suffisamment de connexions ne soient fermées.

`ssl?` (par défaut : `#t`)

Indique s'il faut utiliser SSL/TLS pour chiffrer les communications réseaux. Le certification est automatiquement intégré par la grappe et peut être modifié avec `gnt-cluster renew-crypto`.

`ssl-key` (par défaut : `"/var/lib/ganeti/server.pem"`)

Cela peut être utilisé pour fournir une clé de chiffrement spécifique pour les communications TLS.

`ssl-cert` (par défaut : `"/var/lib/ganeti/server.pem"`)

Cela peut être utilisé pour fournir un certification spécifique pour les communications TLS.

`debug?` (par défaut : `#f`)

Lorsque la valeur est vraie, le démon effectue plus de journalisation pour le débogage. Remarque que cela laissera fuiter des détails de chiffrement dans les journaux, utilisez cette option avec prudence.

**ganeti-confd-service-type** [Variable Scheme]

`ganeti-confd` répond aux requêtes liées à la configuration de la grappe Ganeti. Le but de ce démon est d'avoir une manière rapide et très disponible de demander les valeurs de configuration de la grappe. Il est automatiquement activé sur tous les *candidats maitres*. La valeur de ce service doit être un objet `ganeti-confd-configuration`.

**ganeti-confd-configuration** [Type de données]

La configuration du service `ganeti-confd`.

`ganeti` (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser pour ce service.

`port` (par défaut : 1814)

Le port UDP sur lequel écouter les requêtes réseaux.

`address` (par défaut : "0.0.0.0")

L'adresse réseau sur laquelle le démon se liera.

`debug?` (par défaut : #f)

Lorsque la valeur est vraie, le démon effectue des actions de journalisation supplémentaires pour le débogage.

`ganeti-wconfd-service-type` [Variable Scheme]

`ganeti-wconfd` est le démon qui fait autorité sur la configuration de la grappe et est la seule entité qui peut y accepter des changements. Tous les travaux qui ont besoin de modifier la configuration le feront en envoyant les requêtes appropriées à ce démon. Il ne tourne que sur le *nœud maitre* et sera automatiquement désactivé sur les autres nœuds.

La valeur de ce service est un objet `ganeti-wconfd-configuration`.

`ganeti-wconfd-configuration` [Type de données]

La configuration du service `ganeti-wconfd`.

`ganeti` (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser pour ce service.

`no-voting?` (par défaut : #f)

Le démon refusera de démarrer si la majorité des nœuds de la grappe ne sont pas d'accord pour dire qu'il est le nœud maitre. Indiquez #t pour le démarre même si le quorum n'a pas été atteint (dangereux, utilisez avec prudence).

`debug?` (par défaut : #f)

Lorsque la valeur est vraie, le démon effectue des actions de journalisation supplémentaires pour le débogage.

`ganeti-luxid-service-type` [Variable Scheme]

`ganeti-luxid` est un démon utilisé pour répondre aux requêtes liées à la configuration et à l'état actuel d'une grappe Ganeti. En plus, c'est le démon qui fait autorité pour la queue de travaux de Ganeti. les travaux peuvent être soumis via ce démon et il les programme et les démarre.

Il prend un objet `ganeti-luxid-configuration`.

`ganeti-luxid-configuration` [Type de données]

La configuration du service `ganeti-rapi`.

`ganeti` (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser pour ce service.

`no-voting?` (par défaut : #f)

Le démon refusera de démarrer s'il ne peut pas vérifier que la majorité des nœuds de la grappe pensent qu'il est le nœud maitre. Indiquez #t pour le démarre malgré tout (cela peut être dangereux).



**debug?** (par défaut : **#f**)

Lorsque la valeur est vraie, le démon effectue des actions de journalisation supplémentaires pour le débogage.

**ganeti-rapi-service-type** [Variable Scheme]

**ganeti-rapi** fournit une API à distance pour les grappes Ganeti. Il est lancé sur le maître et peut être utilisé pour effectuer des actions programmées sur la grappe via un protocole de RPC basé sur JSON.

La plupart des opérations de requêtes sont permises sans authentification (à moins que *require-authentication?* ne soit indiqué), alors que les opérations en écriture requièrent une autorisation explicite via le fichier `/var/lib/ganeti/rapi/users`. Voir la documentation de l'API distante de Ganeti (<https://docs.ganeti.org/docs/ganeti/3.0/html/rapi.html>) pour plus d'informations.

La valeur de ce service doit être un objet **ganeti-rapi-configuration**.

**ganeti-rapi-configuration** [Type de données]

La configuration du service **ganeti-rapi**.

**ganeti** (par défaut : **ganeti**)

Le paquet **ganeti** à utiliser pour ce service.

**require-authentication?** (par défaut : **#f**)

Indique s'il faut demander une authentification même pour les opérations en lecture-seule.

**port** (par défaut : **5080**)

Le port TCP sur lequel écouter les requêtes de l'API.

**address** (par défaut : **"0.0.0.0"**)

L'adresse réseau sur laquelle le service se liera. Par défaut il écoute sur toutes les adresses.

**interface** (par défaut : **#f**)

Si une valeur est indiquée, elle doit spécifier une interface réseau spécifique comme **eth0** sur laquelle le démon se liera.

**max-clients** (par défaut : **20**)

Le nombre maximum de requêtes clientes simultanées à prendre en charge. Les connexions supplémentaires sont permises, mais aucune réponse ne sera envoyée tant qu'il n'y aura pas assez de connexions fermées.

**ssl?** (par défaut : **#t**)

Indique s'il faut utiliser le chiffrement SSL/TLS sur le port RAPI.

**ssl-key** (par défaut : **"/var/lib/ganeti/server.pem"**)

Cela peut être utilisé pour fournir une clé de chiffrement spécifique pour les communications TLS.

**ssl-cert** (par défaut : **"/var/lib/ganeti/server.pem"**)

Cela peut être utilisé pour fournir une certification spécifique pour les communications TLS.

`debug?` (par défaut : `#f`)

Lorsque la valeur est vraie, le démon effectue plus de journalisation pour le débogage. Remarque que cela laissera fuiter des détails de chiffrement dans les journaux, utilisez cette option avec prudence.

`ganeti-kvmd-service-type` [Variable Scheme]

`ganeti-kvmd` doit déterminer si une instance KVM donnée a été éteinte par un administrateur ou un utilisateur. Normalement Ganeti redémarre une instance qui a été arrêtée par Ganeti lui-même. Si l'option de grappe `user_shutdown` est vraie, ce démon vérifie la socket QMP fournie par QEMU et écoute les événements d'extinction, et marque l'instance *USER down* au lieu de *ERROR down* lorsqu'elle s'éteint correctement par elle-même.

Il prend un objet `ganeti-kvmd-configuration`.

`ganeti-kvmd-configuration` [Type de données]

`ganeti` (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser pour ce service.

`debug?` (par défaut : `#f`)

Lorsque la valeur est vraie, le démon effectue des actions de journalisation supplémentaires pour le débogage.

`ganeti-mond-service-type` [Variable Scheme]

`ganeti-mond` est un démon facultatif qui propose des fonctionnalités de surveillance de Ganeti. Il est responsable des collecteurs de données et de la publication des informations récupérées sur une interface HTTP.

Il prend un objet `ganeti-mond-configuration`.

`ganeti-mond-configuration` [Type de données]

`ganeti` (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser pour ce service.

`port` (par défaut : `1815`)

Le port sur lequel le démon écoutera.

`address` (par défaut : `"0.0.0.0"`)

L'adresse réseau sur laquelle le démon se liera. Par défaut, il se lie à toutes les interfaces disponibles.

`debug?` (par défaut : `#f`)

Lorsque la valeur est vraie, le démon effectue des actions de journalisation supplémentaires pour le débogage.

`ganeti-metad-service-type` [Variable Scheme]

`ganeti-metad` est un démon facultatif qui peut être utilisé pour fournir des informations sur la grappe aux instances ou aux scripts d'installation de systèmes.

Il prend un objet `ganeti-metad-configuration`.

`ganeti-metad-configuration` [Type de données]

`ganeti` (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser pour ce service.

`port` (par défaut : 80)  
Le port sur lequel le démon écoutera.

`address` (par défaut : `#f`)  
Si la valeur est indiquée, le démon se liera à cette adresse uniquement. Si la valeur n'est pas indiquée, le comportement dépend de la configuration de la grappe.

`debug?` (par défaut : `#f`)  
Lorsque la valeur est vraie, le démon effectue des actions de journalisation supplémentaires pour le débogage.

`ganeti-watcher-service-type` [Variable Scheme]

`ganeti-watcher` est un script conçu pour se lancer périodiquement et s'assurer de la santé de la grappe. Il redémarrera automatiquement les instances qui sont arrêtées sans le consentement de Ganeti, et réparera les liens DRBD au cas où un nœud a redémarré. Il archive aussi les anciennes tâches de la grappe et redémarre les démons Ganeti qui ne sont pas lancés. Si le paramètre de grappe `ensure_node_health` est indiqué, le gardien éteindra aussi les instances et les périphériques DRBD si le nœud sur lequel il est lancé est déclaré hors-ligne par un candidat maître.

Il peut être mis en pause sur tous les nœuds avec `gnt-cluster watcher-pause`.

Le service prend un objet `ganeti-watcher-configuration`.

`ganeti-watcher-configuration` [Type de données]

`ganeti` (par défaut : `ganeti`)  
Le paquet `ganeti` à utiliser pour ce service.

`schedule` (par défaut : `'(next-second-from (next-minute (range 0 60 5)))`)  
Indique quand lancer le script. Par défaut, toutes les cinq minutes.

`rapi-ip` (par défaut : `#f`)  
Cette option doit être spécifiée seulement si le démon RAPI est configuré pour utiliser une interface ou une adresse particulière. Par défaut l'adresse de grappe est utilisée.

`job-age` (par défaut : `(* 6 3600)`)  
Archive les tâches de grappe plus vieilles que cela, en secondes. Par défaut c'est 6 heures. Cela permet de garder un `gnt-job list` gérable.

`verify-disks?` (par défaut : `#t`)  
Si la valeur est `#f`, le gardien n'essayera pas de réparer les liens DRBD cassés automatiquement. Les administrateur·rices devront utiliser `gnt-cluster verify-disks` manuellement à la place.

`debug?` (par défaut : `#f`)  
Lorsque la valeur est `#t`, le script effectue des actions de journalisation supplémentaires pour le débogage.

`ganeti-cleaner-service-type` [Variable Scheme]

`ganeti-cleaner` est un script conçu pour être lancé périodiquement et supprimer les anciens fichiers de la grappe. Ce type de service contrôle deux tâches *cron* : l'une est

conçue pour le nœud maître et purge de manière permanente les anciennes tâches de la grappe, et l'autre est conçue pour tous les nœuds et supprime les certificats X509, les clés et les informations `ganeti-watcher` périmées. Comme tous les services Ganeti, on peut l'ajouter même sur les nœuds non-maîtres car il se désactive tout seul en cas de besoin.

Il prend un objet `ganeti-cleaner-configuration`.

`ganeti-cleaner-configuration` [Type de données]

`ganeti` (par défaut : `ganeti`)

Le paquet `ganeti` à utiliser pour la commande `gnt-cleaner`.

`master-schedule` (par défaut : `"45 1 * * *"`)

La périodicité à laquelle lancer la tâche de nettoyage maître. Par défaut c'est une fois par jour, à 01:45:00.

`node-schedule` (par défaut : `"45 2 * * *"`)

La périodicité à laquelle lancer la tâche de nettoyage des nœuds. Par défaut une fois par jour, à 02:45:00.

### 12.9.30 Services de contrôle de version

Le module (`gnu services version-control`) fournit un service pour permettre l'accès à distance à des dépôts Git locaux. Il y a trois options : en utilisant `git-daemon-service` qui fournit un accès aux dépôts via le protocole non sécurisé `git://` basé sur TCP, en étendant le serveur web `nginx` pour relayer les requêtes vers `git-http-backend` ou en fournissant une interface web avec `cgit-service-type`.

`git-daemon-service` [`#:config (git-daemon-configuration)`] [Procédure Scheme]

Renvoie un service qui lance `git daemon`, un serveur TCP simple pour exposer des dépôts sur le protocole Git pour des accès anonymes.

L'argument facultatif `config` devrait être un objet `<git-daemon-configuration>`, par défaut il permet l'accès en lecture-seule aux dépôts exportés<sup>8</sup> dans `/srv/git`.

`git-daemon-configuration` [Type de données]

Type de données représentant la configuration de `git-daemon-service`.

`package` (par défaut : `git`)

Objet de paquet du système de contrôle de version distribué Git.

`export-all?` (par défaut : `#f`)

Indique s'il faut permettre l'accès à tous les dépôts Git, même s'ils n'ont pas le fichier `git-daemon-export-ok`.

`base-path` (par défaut : `/srv/git`)

Indique s'il faut traduire toutes les requêtes de chemins relativement au chemin actuel. Si vous lancez `git daemon` avec (`base-path "/srv/git"`) sur `'example.com'`, et que vous essayez ensuite de récupérer `'git://example.com/hello.git'`, le démon git interprètera ce chemin comme étant `/srv/git/hello.git`.

<sup>8</sup> En créant le fichier magique `git-daemon-export-ok` dans le répertoire du dépôt.

- user-path** (par défaut : **#f**)  
Indique s'il faut permettre la notation `~user` dans les requêtes. Lorsque spécifié avec une chaîne vide, les requêtes à `git://host/~alice/toto` sont des requêtes d'accès au dépôt `toto` dans le répertoire personnel de l'utilisateur `alice`. Si (`user-path "chemin"`) est spécifié, la même requête est interprétée comme accédant au répertoire `chemin/foo` dans le répertoire personnel de l'utilisateur `alice`.
- listen** (par défaut : `'()`)  
Indique s'il faut écouter sur des adresses IP ou des noms d'hôtes particuliers, par défaut tous.
- port** (par défaut : **#f**)  
Indique s'il faut écouter sur un port particulier, par défaut le 9418.
- whitelist** (par défaut : `'()`)  
Si la liste n'est pas vide, n'autoriser l'accès qu'aux dossiers spécifiés.
- extra-options** (par défaut : `'()`)  
Options supplémentaires qui seront passées à `git daemon`, lancez `man git-daemon` pour plus d'informations.

Le protocole `git://` ne permet pas l'authentification. Lorsque vous récupérez un dépôt via `git://`, vous ne pouvez pas savoir si les données que vous recevez ont été modifiées ou si elles viennent bien de l'hôte spécifié, et votre connexion pourrait être espionnée. Il est préférable d'utiliser un protocole de transport authentifié et chiffré, comme `https`. Bien que Git vous permette de servir des dépôts avec un serveur web peu sophistiqué basé sur les fichiers, il y a un protocole plus rapide implémenté par le programme `git-http-backend`. Ce programme est le moteur des services web Git corrects. Il est conçu pour se trouver derrière un mandataire FastCGI. Voir Section 12.9.19 [Services web], page 441, pour plus d'informations sur la manière de lancer le démon `fcgiwrap` nécessaire.

Guix a un type de données de configuration séparé pour servir des dépôts Git par HTTP.

**git-http-configuration** [Type de données]

Type de données représentant la configuration d'un futur `git-http-service-type` ; il peut actuellement être utilisé pour configurer Nginx à travers `git-http-nginx-location-configuration`.

**package** (par défaut : `git`)  
Objet de paquet du système de contrôle de version distribué Git.

**git-root** (par défaut : `/srv/git`)  
Répertoire contenant les dépôts Git à exposer au monde.

**export-all?** (par défaut : **#f**)  
Indique s'il faut exposer l'accès de tous les dépôts Git dans `git-root`, même s'ils n'ont pas le fichier `git-daemon-export-ok`.

**uri-path** (par défaut : `"/git/"`)  
Préfixe du chemin pour l'accès Git. Avec le préfixe `"/git/"` par défaut, cela traduira `http://server/git/repo.git` en `/srv/git/repo.git`. Les requêtes dont les chemins d'URI ne commencent pas par ce préfixe ne seront pas passées à cette instance de Git.

`fcgiwrap-socket` (par défaut : 127.0.0.1:9000)

Le socket sur lequel le démon `fcgiwrap` écoute. Voir Section 12.9.19 [Services web], page 441.

Il n'y a pas de `git-http-service-type`, actuellement ; à la place vous pouvez créer un `nginx-location-configuration` à partir d'un `git-http-configuration` puis ajouter cela au serveur web.

`git-http-nginx-location-configuration` [Procédure Scheme]  
`[config=(git-http-configuration)]`

Calcule un `nginx-location-configuration` qui correspond à la configuration http Git donnée. Voici un exemple de définition de service nginx qui sert le répertoire `/srv/git` par défaut en HTTPS :

```
(service nginx-service-type
  (nginx-configuration
    (server-blocks
      (list
        (nginx-server-configuration
          (listen '("443 ssl"))
          (server-name "git.my-host.org")
          (ssl-certificate
            "/etc/letsencrypt/live/git.my-host.org/fullchain.pem")
          (ssl-certificate-key
            "/etc/letsencrypt/live/git.my-host.org/privkey.pem")
          (locations
            (list
              (git-http-nginx-location-configuration
                (git-http-configuration (uri-path "/"))))))))))))
```

Ce exemple suppose que vous utilisez Let's Encrypt pour récupérer votre certificat TLS. Voir Section 12.9.20 [Services de certificats], page 461. Le service `certbot` par défaut redirigera tout le trafic HTTP de `git.my-host.org` en HTTPS. Vous devrez aussi ajouter un mandataire `fcgiwrap` à vos services systèmes. Voir Section 12.9.19 [Services web], page 441.

## Service Cgit

`Cgit` (<https://git.zx2c4.com/cgit/>) est une interface web pour des dépôts Git écrite en C.

L'exemple suivant configurera le service avec les valeurs par défaut. Par défaut, on peut accéder à Cgit sur le port (`http://localhost:80`).

```
(service cgit-service-type)
```

Le type `file-object` désigne soit un objet simili-fichier (voir Section 9.12 [G-Expressions], page 166), soit une chaîne.

Les champs de `cgit-configuration` disponibles sont :

`package package` [paramètre de `cgit-configuration`]  
 Le paquet `cgit`.

- nginx-server-configuration-list nginx** [paramètre de `cgit-configuration`]  
Configuration Nginx.
- file-object about-filter** [paramètre de `cgit-configuration`]  
Spécifie une commande qui doit être invoquée pour formater le contenu des pages « à propos » (au plus haut niveau et pour chaque dépôt).  
La valeur par défaut est `""`.
- string agefile** [paramètre de `cgit-configuration`]  
Spécifie un chemin, relativement à chaque dépôt, qui peut être utilisé pour spécifier la date et l'heure du plus récent commit du dépôt.  
La valeur par défaut est `""`.
- file-object auth-filter** [paramètre de `cgit-configuration`]  
Spécifie une commande qui sera invoquée pour authentifier l'accès au dépôt.  
La valeur par défaut est `""`.
- string branch-sort** [paramètre de `cgit-configuration`]  
Drapeau qui, lorsqu'il vaut `'age'`, active le trie par date dans la liste des branches, et le trie par nom lorsqu'il vaut `'name'`.  
La valeur par défaut est `"name"`.
- string cache-root** [paramètre de `cgit-configuration`]  
Chemin utilisé pour stocker les entrées de cache de `cgit`.  
La valeur par défaut est `"/var/cache/cgit"`.
- integer cache-static-ttl** [paramètre de `cgit-configuration`]  
Nombre qui spécifie le temps de vie, en minute, des versions en cache des pages du dépôt accédées par leur SHA-1.  
La valeur par défaut est `'-1'`.
- integer cache-dynamic-ttl** [paramètre de `cgit-configuration`]  
Nombre qui spécifie le temps de vie, en minutes, des version en cache des pages du dépôt accédées sans leur SHA1.  
La valeur par défaut est `'5'`.
- integer cache-repo-ttl** [paramètre de `cgit-configuration`]  
Nombre qui spécifie le temps de vie, en minute, des version en cache de la page de résumé du dépôt.  
La valeur par défaut est `'5'`.
- integer cache-root-ttl** [paramètre de `cgit-configuration`]  
Nombre qui spécifie le temps de vie, en minutes, de la version en cache de la page d'index du dépôt.  
La valeur par défaut est `'5'`.
- integer cache-scanrc-ttl** [paramètre de `cgit-configuration`]  
Nombre qui spécifie le temps de vie, en minutes, de la version en cache du résultat du scan d'un chemin dans le dépôt Git.  
La valeur par défaut est `'15'`.

- integer** `cache-about-ttl` [paramètre de `cgit-configuration`]  
Nombre qui spécifie le temps de vie, en minutes, de la version en cache de la page « à propos » du dépôt.  
La valeur par défaut est '15'.
- integer** `cache-snapshot-ttl` [paramètre de `cgit-configuration`]  
Nombre qui spécifie le temps de vie, en minutes, de la version en cache des archives.  
La valeur par défaut est '5'.
- integer** `cache-size` [paramètre de `cgit-configuration`]  
Le nombre maximum d'entrées dans le cache de `cgit`. Lorsque la valeur est '0', le cache est désactivé.  
La valeur par défaut est '0'.
- boolean** `case-sensitive-sort?` [paramètre de `cgit-configuration`]  
Indique si le tri des éléments est sensible à la casse.  
La valeur par défaut est '#t'.
- list** `clone-prefix` [paramètre de `cgit-configuration`]  
Liste des préfixes communs qui, lorsqu'ils sont combinés à l'URL du dépôt, génèrent des URL de clone valides pour le dépôt.  
La valeur par défaut est '()'.
- list** `clone-url` [paramètre de `cgit-configuration`]  
Liste des modèles `clone-url`.  
La valeur par défaut est '()'.
- file-object** `commit-filter` [paramètre de `cgit-configuration`]  
Commande qui sera invoquée pour formater les messages de commit.  
La valeur par défaut est "".
- string** `commit-sort` [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut 'date', active le tri par date strict dans le messages de commit, et le tri topologique strict lorsqu'il vaut 'topo'.  
La valeur par défaut est "git log".
- file-object** `css` [paramètre de `cgit-configuration`]  
URL qui spécifie le document `css` à inclure dans les pages `cgit`.  
La valeur par défaut est "/share/cgit/cgit.css".
- file-object** `email-filter` [paramètre de `cgit-configuration`]  
Spécifie une commande qui sera invoquée pour formater les noms et l'adresse de courriel des commiteurs, des auteurs et des tagguez, représentés à plusieurs endroits dans l'interface `cgit`.  
La valeur par défaut est "".



- boolean embedded?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, fera générer un fragment HTML à cgit qu'il sera possible d'inclure dans d'autres pages HTML.  
La valeur par défaut est `#f`.
- boolean enable-commit-graph?** [paramètre de `cgit-configuration`]  
Drapeau qui, lorsqu'il vaut `#t`, fera afficher un historique en ASCII-art à gauche des messages de commit dans la page de log du dépôt.  
La valeur par défaut est `#f`.
- boolean enable-filter-overrides?** [paramètre de `cgit-configuration`]  
Drapeau qui, lorsqu'il vaut `#t`, permet à tous les paramètres de filtrage d'être modifiés dans des fichiers `gitrc` spécifiques au dépôt.  
La valeur par défaut est `#f`.
- boolean enable-follow-links?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, permet aux utilisateurs de suivre un fichier dans la vue « log ».  
La valeur par défaut est `#f`.
- boolean enable-http-clone?** [paramètre de `cgit-configuration`]  
Si la valeur est `#t`, cgit agira comme un point d'accès HTTP idiot pour les clones Git.  
La valeur par défaut est `#t`.
- boolean enable-index-links?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, fera générer des liens « résumé », « commit » et « arborescence » supplémentaires pour chaque dépôt dans l'index des dépôts.  
La valeur par défaut est `#f`.
- boolean enable-index-owner?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, fera afficher le propriétaire de chaque dépôt dans l'index des dépôts.  
La valeur par défaut est `#t`.
- boolean enable-log-filecount?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, fera afficher à cgit le nombre de fichiers modifiés pour chaque commit sur la page de log du dépôt.  
La valeur par défaut est `#f`.
- boolean enable-log-linecount?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, fera afficher à cgit le nombre de lignes ajoutées et enlevées pour chaque commit de la page de log du dépôt.  
La valeur par défaut est `#f`.
- boolean enable-remote-branches?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, fera afficher les branches distantes dans les vues du résumé et des références.  
La valeur par défaut est `#f`.

- boolean enable-subject-links?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut '1', fera utiliser à `cgit` le sujet du commit parent comme texte du lien lors de la génération des liens vers les commits parents dans la vue des commits.  
La valeur par défaut est '#f'.
- boolean enable-html-serving?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut '#t', fera utiliser à `cgit` l esujet du commit parent comme texte du lien lors de la génération des liens vers le commit parent dans la vue des commits.  
La valeur par défaut est '#f'.
- boolean enable-tree-linenumbers?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut '#t', fera générer à `cgit` des liens vers le numéro de ligne pour les blobs en texte brut affichés dans la vue de l'arborescence.  
La valeur par défaut est '#t'.
- boolean enable-git-config?** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut '#t', permettra à `cgit` d'utiliser la configuration Git pour spécifier des paramètres spécifiques au dépôt.  
La valeur par défaut est '#f'.
- file-object favicon** [paramètre de `cgit-configuration`]  
URL utilisée comme lien vers un icône pour `cgit`.  
La valeur par défaut est `"/favicon.ico"`.
- string footer** [paramètre de `cgit-configuration`]  
Le contenu du fichier spécifié avec cette option sera inclus directement au bas de toutes les pages (c.-à-d. qu'il remplace le message « généré par ... » générique).  
La valeur par défaut est `""`.
- string head-include** [paramètre de `cgit-configuration`]  
Le contenu du fichier spécifié dans cette option sera inclus directement dans la section HEAD HTML de toutes les pages.  
La valeur par défaut est `""`.
- string header** [paramètre de `cgit-configuration`]  
Le contenu du fichier spécifié avec cette option sera inclus directement au début de toutes les pages.  
La valeur par défaut est `""`.
- file-object include** [paramètre de `cgit-configuration`]  
Nom d'un fichier de configuration à inclure avant que le reste du fichier de configuration actuel ne soit analysé.  
La valeur par défaut est `""`.
- string index-header** [paramètre de `cgit-configuration`]  
Le contenu du fichier spécifié avec cette option sera inclus directement au dessus de l'index des dépôts.  
La valeur par défaut est `""`.

- string index-info** [paramètre de `cgит-configuration`]  
Le contenu du fichier spécifié avec cette option sera inclus directement en dessous de l'en-tête sur la page d'index du dépôt.  
La valeur par défaut est `""`.
- boolean local-time?** [paramètre de `cgит-configuration`]  
Drapeau qui, s'il vaut `#t`, fera afficher à `cgит` l'heure et la date de commit et de tag dans le fuseau horaire du serveur.  
La valeur par défaut est `#f`.
- file-object logo** [paramètre de `cgит-configuration`]  
URL qui spécifie la source d'une image utilisé comme logo sur toutes les pages `cgит`.  
La valeur par défaut est `"/share/cgit/cgit.png"`.
- string logo-link** [paramètre de `cgит-configuration`]  
URL chargée lors du clic sur l'image du logo de `cgит`.  
La valeur par défaut est `""`.
- file-object owner-filter** [paramètre de `cgит-configuration`]  
Commande qui sera invoquée pour formater la colonne propriétaire sur la page principale.  
La valeur par défaut est `""`.
- integer max-atom-items** [paramètre de `cgит-configuration`]  
Nombre d'éléments à afficher dans la vue des flux atom.  
La valeur par défaut est `'10'`.
- integer max-commit-count** [paramètre de `cgит-configuration`]  
Nombre d'éléments à lister par page dans la vue « log ».  
La valeur par défaut est `'50'`.
- integer max-message-length** [paramètre de `cgит-configuration`]  
Nombre caractères de messages de commit à afficher dans la vue « log ».  
La valeur par défaut est `'80'`.
- integer max-repo-count** [paramètre de `cgит-configuration`]  
Spécifie le nombre d'éléments à lister par page sur la page de l'index des dépôts.  
La valeur par défaut est `'50'`.
- integer max-repodesc-length** [paramètre de `cgит-configuration`]  
Spécifie le nombre maximum de caractères de description de dépôts à afficher sur la page d'index des dépôts.  
La valeur par défaut est `'80'`.
- integer max-blob-size** [paramètre de `cgит-configuration`]  
Spécifie la taille maximale d'un blob pour lequel afficher du HTML en kilo-octets.  
La valeur par défaut est `'0'`.

- string max-stats** [paramètre de `cggit-configuration`]  
Période de statistiques maximale. Les valeurs valides sont ‘week’, ‘month’, ‘quarter’ et ‘year’.  
La valeur par défaut est ‘’.
- mimetype-alist mimetype** [paramètre de `cggit-configuration`]  
Type mime pour l’extension de fichier spécifiée.  
La valeur par défaut est ‘((gif "image/gif") (html "text/html") (jpg "image/jpeg") (jpeg "image/jpeg") (pdf "application/pdf") (png "image/png") (svg "image/svg+xml"))’.
- file-object mimetype-file** [paramètre de `cggit-configuration`]  
Spécifie le fichier à utiliser pour la recherche automatique de type mime.  
La valeur par défaut est ‘’.
- string module-link** [paramètre de `cggit-configuration`]  
Texte qui sera utilisé comme chaîne de formatage pour un lien hypertexte lorsqu’un sous-module est affiché dans la liste du répertoire.  
La valeur par défaut est ‘’.
- boolean nocache?** [paramètre de `cggit-configuration`]  
Si la valeur est ‘#t’, le cache est désactivé.  
La valeur par défaut est ‘#f’.
- boolean noplainemail?** [paramètre de `cggit-configuration`]  
Si la valeur est ‘#t’, l’affichage des adresse de courriel des auteurs sera désactivé.  
La valeur par défaut est ‘#f’.
- boolean noheader?** [paramètre de `cggit-configuration`]  
Drapeau qui, s’il vaut ‘#t’, fera omettre à `cggit` l’en-tête standard sur toutes les pages.  
La valeur par défaut est ‘#f’.
- project-list project-list** [paramètre de `cggit-configuration`]  
UNE liste de sous-répertoires dans `repository-directory`, relativement à lui, qui devrait être chargé comme des dépôts Git. Une liste vide signifie que tous les sous-répertoires seront chargés.  
La valeur par défaut est ‘()’.
- file-object readme** [paramètre de `cggit-configuration`]  
Texte utilisé comme valeur par défaut pour `cggit-repo-readme`.  
La valeur par défaut est ‘’.
- boolean remove-suffix?** [paramètre de `cggit-configuration`]  
Si la valeur est #t et que `repository-directory` est activé, si un dépôt avec un suffixe de `.git` est trouvé, ce suffixe sera supprimé de l’URL et du nom.  
La valeur par défaut est ‘#f’.

- integer** `renamelimit` [paramètre de `cggit-configuration`]  
Nombre maximum de fichiers à considérer lors de la détection des renommages.  
La valeur par défaut est `'-1'`.
- string** `repository-sort` [paramètre de `cggit-configuration`]  
La manière dont les dépôt de chaque section sont rangés.  
La valeur par défaut est `""`.
- robots-list** `robots` [paramètre de `cggit-configuration`]  
Texte utilisé comme contenu du méta-attribut `robots`.  
La valeur par défaut est `'("noindex" "nofollow")'`.
- string** `root-desc` [paramètre de `cggit-configuration`]  
Texte affiché en dessous de l'en-tête de la page d'index des dépôts.  
La valeur par défaut est `"a fast webinterface for the git dscm"`.
- string** `root-readme` [paramètre de `cggit-configuration`]  
Le contenu du fichier spécifié avec cette option sera inclus directement en dessous du lien « à propos » sur la page d'index du dépôt.  
La valeur par défaut est `""`.
- string** `root-title` [paramètre de `cggit-configuration`]  
Texte affiché sur la page d'index des dépôts.  
La valeur par défaut est `""`.
- boolean** `scan-hidden-path` [paramètre de `cggit-configuration`]  
Si la valeur est `'#t'` et que `repository-directory` est activé, `repository-directory` recherchera de manière récursive dans les répertoires dont le nom commence par un point. Sinon, `repository-directory` restera hors de ces répertoires, considérés comme « cachés ». Remarquez que cela ne s'applique pas au répertoire `«.git»` dans le dépôts non bruts.  
La valeur par défaut est `'#f'`.
- list** `snapshots` [paramètre de `cggit-configuration`]  
Texte qui spécifie l'ensemble des formats d'archives par défaut pour lesquelles `cggit` générera un lien.  
La valeur par défaut est `'()'.`
- repository-directory** `repository-directory` [paramètre de `cggit-configuration`]  
Nom du répertoire à scanner pour trouver les dépôts (représente `scan-path`).  
La valeur par défaut est `"/srv/git"`.
- string** `section` [paramètre de `cggit-configuration`]  
Le nom de la section de dépôts actuelle — tous les dépôts définis après ce point hériteront du nom de section actuel.  
La valeur par défaut est `""`.

- string section-sort** [paramètre de `cgit-configuration`]  
Drapeau qui, s'il vaut '1', triera les sections dans la liste des dépôts par nom.  
La valeur par défaut est "".
- integer section-from-path** [paramètre de `cgit-configuration`]  
Un nombre qui, s'il est défini avant `repository-directory`, spécifier combien d'éléments de chemin de chaque chemin de dépôt utiliser comme nom de section par défaut.  
La valeur par défaut est '0'.
- boolean side-by-side-diffs?** [paramètre de `cgit-configuration`]  
Si la valeur est '#t', afficher des diffs côte à côte au lieu des unidiffs par défaut.  
La valeur par défaut est '#f'.
- file-object source-filter** [paramètre de `cgit-configuration`]  
Spécifie une commande qui sera invoquée pour formater les blobs en texte brut dans la vue de l'arborescence.  
La valeur par défaut est "".
- integer summary-branches** [paramètre de `cgit-configuration`]  
Spécifie le nombre de branches à afficher dans la vue « résumé » du dépôt.  
La valeur par défaut est '10'.
- integer summary-log** [paramètre de `cgit-configuration`]  
Spécifie le nombre d'élément du journal à afficher dans la vue « résumé » du dépôt.  
La valeur par défaut est '10'.
- integer summary-tags** [paramètre de `cgit-configuration`]  
Spécifie le nombre de tags à afficher dans la vue « résumé » du dépôt.  
La valeur par défaut est '10'.
- string strict-export** [paramètre de `cgit-configuration`]  
Nom de fichier qui, s'il est spécifié, doit être présent dans le dépôt pour que `cgit` accorde l'accès à ce dépôt.  
La valeur par défaut est "".
- string virtual-root** [paramètre de `cgit-configuration`]  
URL qui, si elle est spécifiée, sera utilisée comme racine pour tous les liens `cgit`.  
La valeur par défaut est "/".
- repository-cgit-configuration-list** [paramètre de `cgit-configuration`]  
**repositories**  
Une liste d'enregistrements *cgit-repo* à utiliser avec `config`.  
La valeur par défaut est '()'.
- Les champs de `repository-cgit-configuration` disponibles sont :
- repo-list snapshots** [paramètre de `repository-cgit-configuration`]  
Un masque de formats d'archives pour ce dépôt pour lesquelles `cgit` générera un lien, restreint par le paramètre `snapshots` global.  
La valeur par défaut est '()'.

- repo-file-object** [paramètre de repository-cgit-configuration]  
**source-filter**  
Modifie le **source-filter** par défaut.  
La valeur par défaut est `""`.
- repo-string url** [paramètre de repository-cgit-configuration]  
URL relative utilisée pour accéder au dépôt.  
La valeur par défaut est `""`.
- repo-file-object** [paramètre de repository-cgit-configuration]  
**about-filter**  
Modifie le paramètre **about-filter** par défaut.  
La valeur par défaut est `""`.
- repo-string branch-sort** [paramètre de repository-cgit-configuration]  
Drapeau qui, s'il vaut `'age'`, active le tri par date dans la liste des branches, et lorsqu'il vaut `'name'`, le tri par nom.  
La valeur par défaut est `""`.
- repo-list clone-url** [paramètre de repository-cgit-configuration]  
Une liste d'URL qui peuvent être utilisées pour cloner ce dépôt.  
La valeur par défaut est `('')`.
- repo-file-object** [paramètre de repository-cgit-configuration]  
**commit-filter**  
Modifie le paramètre **commit-filter** par défaut.  
La valeur par défaut est `""`.
- repo-string commit-sort** [paramètre de repository-cgit-configuration]  
Drapeau qui, s'il vaut `'date'`, active le tri par date strict dans les messages de commit, et le tri topologique strict lorsqu'il vaut `'topo'`.  
La valeur par défaut est `""`.
- repo-string defbranch** [paramètre de repository-cgit-configuration]  
Le nom de la branche par défaut de ce dépôt. Si cette branche n'existe pas dans le dépôt, le premier nom de branche (trié) sera utilisé par défaut. Par défaut la branche pointée par HEAD, ou « master » s'il n'y a pas de HEAD convenable.  
La valeur par défaut est `""`.
- repo-string desc** [paramètre de repository-cgit-configuration]  
La valeur à afficher comme description du dépôt.  
La valeur par défaut est `""`.
- repo-string homepage** [paramètre de repository-cgit-configuration]  
La valeur à afficher comme page d'accueil du dépôt.  
La valeur par défaut est `""`.

- `repo-file-object` [paramètre de `repository-cgit-configuration`]  
`email-filter`  
Modifie le paramètre `email-filter` par défaut.  
La valeur par défaut est `""`.
- `peut-être-booléen-repo` [paramètre de `repository-cgit-configuration`]  
`enable-commit-graph?`  
Un drapeau qui peut être utilisé pour désactiver le paramètre `enable-commit-graph?` global.  
La valeur par défaut est `'disabled'`.
- `peut-être-booléen-repo` [paramètre de `repository-cgit-configuration`]  
`enable-log-filecount?`  
Un drapeau qui peut être utilisé pour désactiver le paramètre `enable-log-filecount?` global.  
La valeur par défaut est `'disabled'`.
- `peut-être-booléen-repo` [paramètre de `repository-cgit-configuration`]  
`enable-log-linecount?`  
Un drapeau qui peut être utilisé pour désactiver le paramètre `enable-log-linecount?` global.  
La valeur par défaut est `'disabled'`.
- `peut-être-booléen-repo` [paramètre de `repository-cgit-configuration`]  
`enable-remote-branches?`  
Drapeau qui, s'il vaut `#t`, fera afficher les branches distantes dans les vues du résumé et des références.  
La valeur par défaut est `'disabled'`.
- `peut-être-booléen-repo` [paramètre de `repository-cgit-configuration`]  
`enable-subject-links?`  
Un drapeau qui peut être utilisé pour modifier le paramètre `enable-subject-links?` global.  
La valeur par défaut est `'disabled'`.
- `peut-être-booléen-repo` [paramètre de `repository-cgit-configuration`]  
`enable-html-serving?`  
Un drapeau qui peut être utilisé pour modifier le paramètre `enable-html-serving?` global.  
La valeur par défaut est `'disabled'`.
- `repo-boolean` `hide?` [paramètre de `repository-cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, cache le dépôt de l'index des dépôts.  
La valeur par défaut est `#f`.
- `repo-boolean` `ignore?` [paramètre de `repository-cgit-configuration`]  
Drapeau qui, s'il vaut `#t`, ignore le dépôt.  
La valeur par défaut est `#f`.



- repo-file-object logo** [paramètre de `repository-cgit-configuration`]  
URL qui spécifie la source d'une image qui sera utilisée comme logo sur les pages de ce dépôt.  
La valeur par défaut est `""`.
- repo-string logo-link** [paramètre de `repository-cgit-configuration`]  
URL chargée lors du clic sur l'image du logo de cgit.  
La valeur par défaut est `""`.
- repo-file-object owner-filter** [paramètre de `repository-cgit-configuration`]  
Modifie le paramètre `owner-filter` par défaut.  
La valeur par défaut est `""`.
- repo-string module-link** [paramètre de `repository-cgit-configuration`]  
Texte qui sera utilisé comme chaîne de formatage pour un lien hypertexte lorsqu'un sous-module est affiché dans une liste de fichiers. Les arguments pour la chaîne de formatage sont le chemin et le SHA1 du commit du sous-module.  
La valeur par défaut est `""`.
- module-link-path module-link-path** [paramètre de `repository-cgit-configuration`]  
Texte qui sera utilisé comme chaîne de formatage lorsqu'un sous-module avec un chemin spécifié sera affiché dans une liste de fichiers.  
La valeur par défaut est `()`.
- repo-string max-stats** [paramètre de `repository-cgit-configuration`]  
Modifie la période de statistique maximale par défaut.  
La valeur par défaut est `""`.
- repo-string name** [paramètre de `repository-cgit-configuration`]  
La valeur à afficher comme nom de dépôt.  
La valeur par défaut est `""`.
- repo-string owner** [paramètre de `repository-cgit-configuration`]  
Une valeur utilisée pour identifier le propriétaire du dépôt.  
La valeur par défaut est `""`.
- repo-string path** [paramètre de `repository-cgit-configuration`]  
Un chemin absolu vers le répertoire du dépôt.  
La valeur par défaut est `""`.
- repo-string readme** [paramètre de `repository-cgit-configuration`]  
Un chemin (relatif au dépôt) qui spécifie un fichier à inclure directement comme page « À propos » pour ce dépôt.  
La valeur par défaut est `""`.

**repo-string section** [paramètre de `repository-cgit-configuration`]  
 Le nom de la section de dépôts actuelle — tous les dépôts définis après ce point hériteront du nom de section actuel.  
 La valeur par défaut est `""`.

**repo-list extra-options** [paramètre de `repository-cgit-configuration`]  
 Options supplémentaires ajoutées à la fin du fichier `cgitr`.  
 La valeur par défaut est `()`.

**list extra-options** [paramètre de `cgit-configuration`]  
 Options supplémentaires ajoutées à la fin du fichier `cgitr`.  
 La valeur par défaut est `()`.

Cependant, vous pourriez vouloir simplement récupérer un `cgitr` et l'utiliser. Dans ce cas, vous pouvez passer un `opaque-cgit-configuration` comme enregistrement à `cgit-service-type`. Comme son nom l'indique, une configuration opaque n'a pas de capacité de réflexion facile.

Les champs de `opaque-cgit-configuration` disponibles sont :

**package cgit** [paramètre de `opaque-cgit-configuration`]  
 Le paquet `cgit`.

**string string** [paramètre de `opaque-cgit-configuration`]  
 Le contenu de `cgitr`, en tant que chaîne de caractère.

Par exemple, si votre `cgitr` est juste la chaîne vide, vous pouvez instancier un service `cgit` ainsi :

```
(service cgit-service-type
  (opaque-cgit-configuration
    (cgitr "")))
```

## Service Gitolite

Gitolite (<https://gitolite.com/gitolite/>) est un outil pour héberger des dépôts Git sur un serveur central.

Gitolite peut gérer plusieurs dépôts et utilisateurs et supporte une configuration flexible des permissions pour les utilisateurs sur ces dépôts.

L'exemple suivant configure Gitolite en utilisant l'utilisateur `git` par défaut et la clef SSH fournie.

```
(service gitolite-service-type
  (gitolite-configuration
    (admin-pubkey (plain-file
      "yourname.pub"
      "ssh-rsa AAAA... guix@example.com"))))
```

Gitolite est configuré via un dépôt d'administration spécial que vous pouvez cloner. Par exemple, si vous hébergez Gitolite sur `example.com`, vous pouvez lancer la commande suivante pour cloner le dépôt d'administration.

```
git clone git@example.com:gitolite-admin
```

Lorsque le service Gitolite est activé, la clef `admin-pubkey` fournie sera insérée dans le répertoire `keydir` du dépôt `gitolite-admin`. Si cela change le dépôt, un commit sera effectué avec le message « `gitolite setup by GNU Guix` ».

**gitolite-configuration** [Type de données]

Type de données représentant la configuration de `gitolite-service-type`.

**package** (par défaut : `gitolite`)

Le paquet Gitolite à utiliser. Il y a des dépendances facultatives à Gitolite qui ne sont pas incluses dans le paquet par défaut comme Redis et `git-annex`. Ces fonctionnalités peuvent devenir disponibles en utilisant la procédure `make-gitolite` dans le module (`gnu packages version-control`) pour produire une variante de Gitolite avec les dépendances supplémentaires souhaitées.

Le code suivant renvoie un paquet dans lequel les programmes Redis et `git-annex` peuvent être invoqués par les scripts de Gitolite :

```
(use-modules (gnu packages databases)
             (gnu packages haskell-apps)
             (gnu packages version-control))
(make-gitolite (list redis git-annex))
```

**user** (par défaut : `git`)

Utilisateur pour utiliser Gitolite. Cela sera l'utilisateur à utiliser pour accéder à Gitolite par SSH.

**group** (par défaut : `git`)

Groupe à utiliser pour Gitolite.

**home-directory** (par défaut : `"/var/lib/gitolite"`)

Répertoire dans lequel stocker la configuration et les dépôts de Gitolite.

**rc-file** (par défaut : `(gitolite-rc-file)`)

Un objet « simili-fichier » (voir Section 9.12 [G-Expressions], page 166) représentant la configuration de Gitolite.

**admin-pubkey** (par défaut : `#f`)

Un objet « simili-fichier » (voir Section 9.12 [G-Expressions], page 166) utilisé pour paramétrer Gitolite. Il sera inséré dans le répertoire `keydir` dans le dépôt `gitolite-admin`.

Pour spécifier la clef SSH comme chaîne de caractère, utilisez la fonction `plain-file`.

```
(plain-file "yourname.pub" "ssh-rsa AAAA... guix@example.com")■
```

**gitolite-rc-file** [Type de données]

Type de données représentant le fichier RC de Gitolite.

**umask** (par défaut : `#o0077`)

Cela contrôle les permissions que Gitolite propose sur les dépôts et leur contenu.

Une valeur comme `#o0027` donnera accès en lecture au groupe utilisé par Gitolite (par défaut : `git`). Cela est nécessaire lorsque vous utilisez Gitolite avec un logiciel comme `git` ou `gitweb`.

`local-code` (par défaut : `"$rc{GL_ADMIN_BASE}/local"`)

Vous permet d'ajouter vos propres programmes non essentiels, voire d'écraser les programmes fournis par défaut avec vos propres programmes.

Fournissez le chemin COMPLET à cette variable. Par défaut, le répertoire appelé « local » dans votre clone gitolite est utilisé, ce qui permet de le versionner et de vous permettre de le modifier sans avoir à vous connecter au serveur.

`unsafe-pattern` (par défaut : `#f`)

Une expression régulière en Perl qui correspond aux configurations non sûres dans le fichier de configuration. Voir la documentation de Gitolite ([https://gitolite.com/gitolite/git-config.html#compensating-for-unsafe\\_patt](https://gitolite.com/gitolite/git-config.html#compensating-for-unsafe_patt)) pour plus d'informations.

Lorsque la valeur n'est pas `#f`, elle devrait être une chaîne contenant une expression régulière en Perl, comme `"[^\~#\$\&()|;<>]"`, qui est la valeur par défaut utilisée par gitolite. Elle rejette les caractères spéciaux dans la configuration qui pourraient être interprétés par un shell, ce qui est utile lorsque vous partagez la charge d'administration avec d'autres personnes qui n'ont pas accès à un shell sur le serveur.

`git-config-keys` (par défaut : `""`)

Gitolite vous permet de modifier les configurations git avec le mot-clef `'config'`. Ce paramètre vous permet de contrôler les clefs de configuration acceptables.

`roles` (par défaut : `'(("READERS" . 1) ("WRITERS" . ))`)

Indique les noms des rôles qui peuvent être utilisés par les utilisateurs avec la commande `perms`.

`enable` (par défaut : `'("help" "desc" "info" "perms" "writable" "ssh-authkeys" "git-config" "daemon" "gitweb")`)

Ce paramètre contrôle les commandes et les fonctionnalités à activer dans Gitolite.

## Service Gitile

Gitile (<https://git.lepillier.eu/gitile>) est une forge Git qui permet de visualiser le contenu de dépôts Git dans votre navigateur web.

Gitile fonctionne mieux en tandem avec Gitolite, et servira les dépôts publics de Gitolite par défaut. Le service devrait écouter uniquement sur un port local, et vous devriez configurer un serveur web pour servir les ressources statiques. Le service gitile fournit une manière pratique d'étendre le service Nginx pour cela (voir [NGINX], page 444).

L'exemple suivant configurera Gitile pour servir les dépôts d'un emplacement personnalisé, avec quelques messages par défaut pour la page d'accueil et les pieds de page.

```
(service gitile-service-type
  (gitile-configuration
    (repositories "/srv/git")
    (base-git-url "https://myweb.site/git")))
```

```
(index-title "My git repositories")
(intro '((p "This is all my public work!"))))
(footer '((p "This is the end"))))
(nginx-server-block
  (nginx-server-configuration
    (ssl-certificate
      "/etc/letsencrypt/live/myweb.site/fullchain.pem")
    (ssl-certificate-key
      "/etc/letsencrypt/live/myweb.site/privkey.pem")
    (listen '("443 ssl http2" "[::]:443 ssl http2"))
    (locations
      (list
        ;; Allow for https anonymous fetch on /git/ urls.
        (git-http-nginx-location-configuration
          (git-http-configuration
            (uri-path "/git/")
            (git-root "/var/lib/gitolite/repositories")))))))))))■
```

En plus de l'enregistrement de configuration, vous devriez configurer vos dépôts git pour contenir certaines informations facultatives. Tout d'abord, vos dépôts publics doivent contenir le fichier magique `git-daemon-export-ok` qui permet à Git d'exporter le dépôt. Gitile utilise la présence de ce fichier pour détecter les dépôts publics qu'il doit rendre accessibles. Pour cela, avec Gitolite par exemple, modifiez votre `conf/gitolite.conf` pour inclure ceci dans les dépôts que vous voulez rendre publics :

```
repo toto
  R = daemon
```

En plus, Gitile peut lire la configuration du dépôt pour afficher des informations supplémentaires sur les page du dépôt. Gitile utilise l'espace de nom `gitweb` pour sa configuration. Par exemple, vous pouvez utiliser cela dans votre `conf/gitolite.conf` :

```
repo toto
  R = daemon
  desc = Une description longue, éventuellement avec du <i>HTML</i>, qui se trouvera
  config gitweb.name = Le projet Toto
  config gitweb.synopsis = Une description courte, affichée sur la page principale d
```

N'oubliez pas de commiter et de pousser les changements quand vous êtes satisfait. Vous pourrez avoir besoin de changer la configuration de gitolite pour permettre aux options précédentes d'être acceptées. Une manière de faire est d'ajouter la définition de service suivante :

```
(service gitolite-service-type
  (gitolite-configuration
    (admin-pubkey (local-file "key.pub"))
    (rc-file
      (gitolite-rc-file
        (umask #o0027)
        ;; Permet d'indiquer n'importe quelle clé de configuration■
        (git-config-keys ".*"))
```

```
;; Admet n'importe quel texte comme valeur de configuration valide
(unsafe-patt "~$"))))
```

**gitile-configuration** [Type de données]

Type de données représentant la configuration de `gitile-service-type`.

**package** (par défaut : `gitile`)

Le paquet Gitile à utiliser.

**host** (par défaut : `"localhost"`)

L'hôte sur lequel gitile écoute.

**port** (par défaut : `8080`)

Le port sur lequel gitile écoute.

**database** (par défaut : `"/var/lib/gitile/gitile-db.sql"`)

L'emplacement de la base de données.

**repositories** (par défaut : `"/var/lib/gitolite/repositories"`)

L'emplacement des dépôts. Notez que seuls les dépôts publics seront affichés par Gitile. Pour rendre un dépôt public, ajouter un fichier vide `git-daemon-export-ok` à la racine de ce dépôt.

**base-git-url**

L'URL git de base qui sera utilisée pour afficher les commandes de clonage.

**index-title** (par défaut : `"Index"`)

La titre de la page pour la page d'index qui répertorie tous les dépôts disponibles.

**intro** (par défaut : `'()`)

Le contenu de l'introduction, sous la forme d'une liste d'expressions `sxml`. C'est ce qui est affiché au-dessus de la liste des dépôts, sur la page d'index.

**footer** (par défaut : `'()`)

Le contenu du pied-de-page, sous la forme d'une liste d'expressions `sxml`. Il apparaît sur chaque page servie par Gitile.

**nginx-server-block**

Un bloc de serveur `nginx` qui sera étendu et utilisé comme serveur mandataire inverse par Gitile pour servir ses pages, et comme serveur web normal pour servir ses ressources statiques.

Vous pouvez utiliser ce bloc pour ajouter d'autres URLs personnalisées à votre domaine, comme une URL `/git/` pour les clonages anonymes, ou pour servir tout autre fichier que vous voudriez servir.

### 12.9.31 Services de jeu

#### Le service de la Bataille pour Wesnoth

La Bataille pour Wesnoth (<https://wesnoth.org>) est un jeu de stratégie en tour par tour dans un univers fantastique, avec plusieurs campagnes solo et des parties multijoueurs (en réseau et en local).

**Variable Scheme** *wesnothd-service-type* [Variable]

Type de service pour le service `wesnothd`. Sa valeur doit être un objet `wesnothd-configuration`. Pour lancer `wesnothd` avec la configuration par défaut, instanciez-le ainsi :

```
(service wesnothd-service-type)
```

**wesnothd-configuration** [Type de données]

Type de données représentant la configuration de `wesnothd`.

**package** (par défaut : `wesnoth-server`)

Le paquet de serveur de `wesnoth` à utiliser.

**port** (par défaut : 15000)

Le port sur lequel lier le serveur.

### 12.9.32 Service PAM de montage

Le module (`gnu services pam-mount`) fournit un service qui permet de monter des volumes à la connexion de l'utilisateur-`rice`. Il peut monter n'importe quel format de volume pris en charge par le système.

**Variable Scheme** *pam-mount-service-type* [Variable]

Type de service pour la prise en charge de PAM Mount.

**pam-mount-configuration** [Type de données]

Type de données représentant la configuration de PAM Mount.

Il prend les paramètres suivants :

**rules** Les règles de configuration utilisées pour générer `/etc/security/pam-mount.conf.xml`.

Les règles de configuration sont des éléments SXML (voir Section "SXML" dans *GNU Guile Reference Manual*), et les valeurs par défaut ne montent rien pour personne à la connexion :

```
`((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"))))
```

Certains éléments de volume doivent être ajoutés pour automatiquement monter des volumes à la connexion. voici un exemple qui permet

à l'utilisatrice `alice` de monter son répertoire `HOME` chiffré et permet à l'utilisateur `bob` de monter la partition où il stocke ses données :

```
(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)))
```

La liste complète des options disponibles se trouve sur la page de manuel de `pam_mount.conf` ([http://pam-mount.sourceforge.net/pam\\_mount.conf.5.html](http://pam-mount.sourceforge.net/pam_mount.conf.5.html)).

### 12.9.33 Services Guix

#### Guix Build Coordinator

Le Guix Build Coordinator (<https://git.cbaines.net/guix/build-coordinator/>) aide à distribuer la construction de dérivations entre machines sur lesquelles tourne un *agent*. Le démon de construction est toujours utilisé pour construire les dérivations, mais Guix Build Coordinator gère l'allocation des constructions et des résultats.

Le coordinateur des constructions Guix consiste en un *coordinateur*, et un ou plus processus *agents* connectés. Le coordinateur gère les clients qui soumettent des constructions, et alloue les constructions aux agents. Les processus agents parlent au démon de construction pour effectuer les constructions, puis envoie les résultats au coordinateur.



Il y a un script pour lancer le composant du coordinateur de Guix Build Coordinator, mais le service Guix utilise un script Guile personnalisé à la place, pour fournir une meilleure intégration avec les G-expressions utilisées dans la configuration.

**Variable Scheme** *guix-build-coordinator-service-type* [Variable]  
Le type de service pour Guix Build Coordinator. Sa valeur doit être un `guix-build-coordinator-configuration`.

`guix-build-coordinator-configuration` [Type de données]  
Le type de données représentant la configuration de Guix Build Coordinator.

`package` (par défaut : `guix-build-coordinator`)  
Le paquet Guix Build Coordinator à utiliser.

`user` (par défaut : `"guix-build-coordinator"`)  
L'utilisateur qui lance le service.

`group` (par défaut : `"guix-build-coordinator"`)  
Le groupe système qui lance le service.

`database-uri-string` (par défaut :  
`"sqlite:///var/lib/guix-build-coordinator/guix_build_coordinator.db"`)  
L'URI à utiliser pour la base de données.

`agent-communication-uri` (par défaut : `"http://0.0.0.0:8745"`)  
L'URI décrivant comment écouter les requêtes des processus agents.

`client-communication-uri` (par défaut : `"http://127.0.0.1:8746"`)  
L'URI décrivant comment écouter les requêtes des clients. L'API cliente permet de soumettre des constructions et n'est pas actuellement authentifiée, donc faites attention lors de la configuration de cette valeur.

`allocation-strategy` (par défaut : `#~basic-build-allocation-strategy`)  
Une G-expression pour la stratégie d'allocation à utiliser. C'est une procédure qui prend le stockage de données en argument et remplit le plan d'allocation dans la base de données.

`hooks` (par défaut : `'()`)  
Une liste d'association de crochets. Ils fournissent une manière d'exécuter du code arbitraire en fonction d'évènements, comme le traitement des résultats des constructions.

`parallel-hooks` (par défaut : `'()`)  
Les crochets peuvent être configurés pour être lancés en parallèle. Ce paramètre est une liste d'association de crochets à lancer en parallèle, où la clé est le symbole pour le crochet et la valeur est le nombre de threads à lancer.

`guile` (par défaut : `guile-3.0-latest`)  
Le paquet Guile à utiliser pour lancer Guix Build Coordinator.

**Variable Scheme** *guix-build-coordinator-agent-service-type* [Variable]  
Le type de service pour un agent Guix Build Coordinator. Sa valeur doit être un `guix-build-coordinator-agent-configuration`.

- guix-build-coordinator-agent-configuration** [Type de données]  
Type de données représentant la configuration d'un agent Guix Build Coordinator.
- package** (par défaut : `guix-build-coordinator/agent-only`)  
Le paquet Guix Build Coordinator à utiliser.
- user** (par défaut : `"guix-build-coordinator-agent"`)  
L'utilisateur qui lance le service.
- coordinator** (par défaut : `"http://localhost:8745"`)  
L'URI à utiliser lors de la connexion au coordinateur.
- authentification**  
Enregistrement décrivant comme cet agent devrait s'authentifier avec le coordinateur. Les types d'enregistrement possibles sont décrit plus bas.
- systems** (par défaut : `#f`)  
Les systèmes pour lesquels cet agent devrait récupérer les constructions. Le processus de l'agent utilisera le système actuel comme valeur par défaut.
- max-parallel-builds** (par défaut : `1`)  
Le nombre de constructions qui peuvent tourner simultanément.
- max-allocated-builds** (par défaut : `#f`)  
Le nombre maximum de constructions que cette agent peut allouer.
- max-1min-load-average** (par défaut : `#f`)  
Valeur moyenne de charge à vérifier avant de commencer de nouvelles construction. Si la valeur moyenne à une minute dépasse cette valeur, l'agent attendra avant de commencer de nouvelles constructions.  
Cela sera non spécifié si la valeur est `#f`, et l'agent utilisera le nombre de cœurs rapportés par le système comme la moyenne de charge à 1 minute maximale.
- derivation-substitute-urls** (par défaut : `#f`)  
URL à partir desquelles essayer de récupérer les substituts pour les dérivations, si les dérivations ne sont pas déjà disponibles.
- non-derivation-substitute-urls** (par défaut : `#f`)  
URL à partir desquelles essayer de chercher les substituts pour les entrées des constructions, si les éléments du dépôt des entrées ne sont pas déjà disponibles.
- guix-build-coordinator-agent-password-auth** [Type de données]  
Type de données représentant un moyen d'authentification avec un coordinateur pour les agents, via un UUID et un mot de passe.
- uuid** L'UUID de l'agent. Il devrait être généré par le processus du coordinateur, stocké dans la base du coordinateur et utilisé par l'agent prévu.
- password** Le mot de passe à utiliser lors de la connexion au coordinateur.

**guix-build-coordinator-agent-password-file-auth** [Type de données]  
 Type de données représentant un moyen d'authentification avec un coordinateur via un UUID et un mot de passe lu depuis un fichier.

**uuid** L'UUID de l'agent. Il devrait être généré par le processus du coordinateur, stocké dans la base du coordinateur et utilisé par l'agent prévu.

**password-file**  
 Un fichier contenant le mot de passe à utiliser pour la connexion avec le coordinateur.

**guix-build-coordinator-agent-dynamic-auth** [Type de données]  
 Type de données représentant un moyen d'authentification à un coordinateur via un jeton d'authentification dynamique et le nom de l'agent.

**agent-name**  
 Nom d'un agent, c'est utilisé pour le faire correspondre à une entrée existante dans la base de données s'il existe. Lorsqu'aucune entrée n'est trouvée, une nouvelle entrée est ajoutée automatiquement.

**token** Jeton d'authentification dynamique, créé et stocké dans la base de données du coordinateur, et utilisé par l'agent pour s'authentifier.

**guix-build-coordinator-agent-dynamic-auth-with-file** [Type de données]  
 Type de données représentant une méthode d'authentification à un coordinateur via un jeton d'authentification dynamique lu depuis un fichier et un nom d'agent.

**agent-name**  
 Nom d'un agent, c'est utilisé pour le faire correspondre à une entrée existante dans la base de données s'il existe. Lorsqu'aucune entrée n'est trouvée, une nouvelle entrée est ajoutée automatiquement.

**token-file**  
 Fichier contenant le jeton d'authentification dynamique, créé et stocké dans la base de données du coordinateur, et utilisé par l'agent pour s'authentifier.

Le paquet Guix Build Coordinator contient un script pour demander à une instance de Guix Data Service les dérivations à construire, puis pour soumettre les constructions de ces dérivations au coordinateur. Le type de service ci-dessous vous aide à lancer ce script. C'est un outil supplémentaire qui peut être utile lors de la construction de dérivations contenues dans une instance du Guix Data Service.

**Variable Scheme** *guix-build-coordinator-queue-builds-service-type* [Variable]  
 Type de service pour le script *guix-build-coordinator-queue-builds-from-guix-data-service*. Sa valeur doit être un objet *guix-build-coordinator-queue-builds-configuration*.

**guix-build-coordinator-queue-builds-configuration** [Type de données]  
 Le type de données représentant les options du script qui soumet les constructions à partir du guix data service.

**package** (par défaut : *guix-build-coordinator*)  
 Le paquet Guix Build Coordinator à utiliser.

- user** (par défaut : "guix-build-coordinator-queue-builds")  
L'utilisateur qui lance le service.
- coordinator** (par défaut : "http://localhost:8746")  
L'URI à utiliser lors de la connexion au coordinateur.
- systems** (par défaut : #f)  
Les systèmes pour lesquels récupérer des dérivations à construire.
- systems-and-targets** (par défaut : #f)  
Une liste d'association de paires de systèmes et de cibles pour lesquelles chercher des dérivations à construire.
- guix-data-service** (par défaut : "https://data.guix.gnu.org")  
L'instance du Guix Data Service à laquelle demander les dérivations à construire.
- guix-data-service-build-server-id** (par défaut : #f)  
L'identifiant du serveur de construction du service Guix Data qui correspond aux constructions soumises. Fournir ceci accélère la soumission des constructions car les dérivations déjà soumises peuvent être passées avant de demander au coordinateur de les construire.
- processed-commits-file** (par défaut :  
"/var/cache/guix-build-coordinator-queue-builds/processed-commits")  
Un fichier qui enregistre les commits qui ont été traités, pour éviter de les traiter de nouveau inutilement si le service est redémarré.

## Guix Data Service

Le Guix Data Service (<http://data.guix.gnu.org>) traite, stocke et fournit des données à propos de GNU Guix. Cela comprend des informations sur les paquets, les dérivations et les messages d'avertissement de formatage.

Les données sont stockées dans une base PostgreSQL, et sont disponibles à travers une interface web.

**Variable Scheme** *guix-data-service-type* [Variable]

Le type de service pour le Guix Data Service. Sa valeur doit être un objet `guix-data-service-configuration`. Le service étend éventuellement le service `getmail`, car la liste de diffusion `guix-commits` est utilisée pour récupérer les changements dans le dépôt git de Guix.

**guix-data-service-configuration** [Type de données]

Le type de données représentant la configuration du Guix Data Service.

**package** (par défaut : `guix-data-service`)  
Le paquet Guix Data Service à utiliser.

**user** (par défaut : "guix-data-service")  
L'utilisateur qui lance le service.

**group** (par défaut : "guix-data-service")  
Le groupe système qui lance le service.

- port** (par défaut : 8765)  
Le port sur lequel lier le service web.
- host** (par défaut : "127.0.0.1")  
L'hôte sur lequel lier le service web.
- getmail-idle-mailboxes** (par défaut : #f)  
Si une valeur est indiquée, c'est la liste des boîtes de courriel que le service getmail récupère.
- commits-getmail-retriever-configuration** (par défaut : #f)  
Si la valeur est indiquée, c'est l'objet `getmail-retriever-configuration` avec lequel configurer getmail pour récupérer les courriels à part de la liste de diffusion guix-commits.
- extra-options** (par défaut : '())  
Liste d'options supplémentaires de la ligne de commande pour `guix-data-service`.
- extra-process-jobs-options** (par défaut : '())  
Liste d'options supplémentaires de la ligne de commande pour `guix-data-service-process-jobs`.

## Nar Herder

Nar Herder (<https://git.cbaines.net/guix/nar-herder/about/>) est un utilitaire pour gérer une collection de nars.

**Variable Scheme** *nar-herder-type* [Variable]

Le type de service pour le Guix Data Service. Sa valeur doit être un objet `nar-herder-configuration`. Le service étend éventuellement le service getmail, car la liste de diffusion guix-commits est utilisée pour récupérer les changements dans le dépôt git de Guix.

**nar-herder-configuration** [Type de données]

Le type de données représentant la configuration du Guix Data Service.

- package** (par défaut : `nar-herder`)  
Le paquet Nar Herder à utiliser.
- user** (par défaut : `"nar-herder"`)  
L'utilisateur qui lance le service.
- group** (par défaut : `"nar-herder"`)  
Le groupe système qui lance le service.
- port** (par défaut : 8734)  
Le port sur lequel lier le serveur.
- host** (par défaut : "127.0.0.1")  
L'hôte sur lequel lier le serveur.
- mirror** (par défaut : #f)  
URL facultative de l'autre instance de Nar Herder qui doit être répliquée. Cela signifie que cette instance de Nar Herder téléchargera sa base de données et la gardera à jour.

- database** (par défaut : `"/var/lib/nar-herder/nar_herder.db"`)  
Emplacement de la base de données. Si cette instance de Nar Herder en réplique une autre, la base de données sera téléchargée si elle n'existe pas. Si cette instance de Nar Herder n'en réplique pas une autre, une base de données vide sera créée.
- database-dump** (par défaut : `"/var/lib/nar-herder/nar_herder_dump.db"`)  
Emplacement de la décharge de la base de données. Elle est créée et régulièrement mise à jour en prenant une copie de la base de données. C'est la version de la base de données qui est disponible au téléchargement.
- storage** (par défaut : `#f`)  
Emplacement facultatif dans lequel stocker les nars.
- storage-limit** (par défaut : `"none"`)  
Limite en octets pour les nars stockés dans l'emplacement de stockage. Cela peut valoir « none » pour qu'il n'y ait pas de limite.  
Lorsque l'emplacement de stockage dépasse cette taille, les nars sont supprimés en fonction des critères de suppression des nars.
- storage-nar-removal-criteria** (par défaut : `'()`)  
Critères utilisés pour supprimer les nars de l'emplacement de stockage. Ils sont utilisés en plus de la limite de stockage.  
Lorsque l'emplacement de stockage dépasse la limite de taille, les nars passeront aux critères de suppression et ceux qui correspondent aux critères seront supprimés. Cela continue jusqu'à ce que l'emplacement de stockage repasse en dessous de la limite de taille.  
Chaque critère est spécifié par une chaîne, puis un signe égal, puis une autre chaîne. Actuellement, seul un critère est pris en charge, vérifier si un nar est stocké sur une autre instance de Nar Herder.
- ttl** (par défaut : `#f`)  
Produit des en-têtes HTTP `Cache-Control` qui annoncent une durée de vie (TTL) de *ttl*. *ttl* peut dénoter une durée : `5d` signifie 5 jours, `1m` signifie un mois, etc.  
Cela permet au Guix de l'utilisateur ou de l'utilisatrice de garder les informations de substituts en cache pendant *ttl*.
- negative-ttl** (par défaut : `#f`)  
Produit de la même manière des en-têtes HTTP `Cache-Control` pour annoncer la durée de vie (TLL) d'une recherche *négative* — des entrées du dépôt manquantes, pour lesquelles une erreur 404 est renvoyée. Par défaut, aucun TTL négatif n'est annoncé.
- log-level** (par défaut : `'DEBUG`)  
Niveau de journalisation à utiliser, spécifie un niveau de journalisation comme `'INFO` pour arrêter de journaliser les requêtes individuelles.

### 12.9.34 Services Linux

## Service Early OOM

Early OOM (<https://github.com/rfjakob/earlyoom>), aussi appelé Earlyoom, est un démon de gestion du remplissage mémoire (OOM) minimaliste qui se lance en espace utilisateur et fournit une alternative plus rapide et configurable que le gestionnaire du noyau. Il est utile pour éviter que le système ne cesse de répondre lorsqu'il n'a plus de mémoire.

**earlyoom-service-type** [Variable Scheme]

Le type de service pour le service `earlyoom`, le démon Early OOM. Sa valeur doit être un objet `earlyoom-configuration`, décrit plus bas. Vous pouvez instancier le service avec sa configuration par défaut avec :

```
(service earlyoom-service-type)
```

**earlyoom-configuration** [Type de données]

La configuration pour `earlyoom-service-type`.

**earlyoom** (par défaut : `earlyoom`)

Le paquet Earlyoom à utiliser.

**minimum-available-memory** (par défaut : 10)

La limite de mémoire *disponible* minimum, en pourcentage.

**minimum-free-swap** (par défaut : 10)

La limite d'espace d'échange libre minimum, en pourcentage.

**prefer-regex** (par défaut : `#f`)

Une expression régulière (une chaîne) qui correspond aux noms des processus à tuer en priorité.

**avoid-regex** (par défaut : `#f`)

Une expression régulière (une chaîne) des noms des processus qui ne doivent *pas* être tués.

**memory-report-interval** (par défaut : 0)

L'intervalle en seconde d'affichage du rapport mémoire. Il est désactivé par défaut.

**ignore-positive-oom-score-adj?** (par défaut : `#f`)

Un booléen indiquant si les ajustements positifs indiqués dans `/proc/*/oom_score_adj` doivent être ignorés.

**show-debug-messages?** (par défaut : `#f`)

Un booléen indiquant si les messages de débogages doivent être affichés. Les journaux sont sauvegardés dans `/var/log/earlyoom.log`.

**send-notification-command** (par défaut : `#f`)

On peut utiliser cette option pour fournir une commande personnalisée pour envoyer les notifications.

## Service de chargement de modules du noyau

Le service de chargement de modules du noyau vous permet de charger des modules du noyau au démarrage. C'est particulièrement utile pour les modules qui ne sont pas chargés automatiquement et qui doivent être chargés manuellement, comme c'est le cas avec `ddcci`.

**kernel-module-loader-service-type** [Variable Scheme]

Le type de service pour charger des modules du noyau au démarrage avec `modprobe`. Sa valeur doit être une liste de chaînes de caractères représentant les noms des modules. Par exemple pour charger les pilotes fournis par `ddcci-driver-linux`, en mode débogage en passant certains paramètres au module, on peut faire :

```
(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)))
```

## Service Rasdaemon

Le service Rasdaemon fournit un démon qui surveille les rapports RAS (Reliability, Availability, and Serviceability) de la plateforme dans les événements de trace du noyau Linux, et les envoie à `syslogd`.

La fiabilité, la disponibilité et la serviabilité est un concept utilisé sur les serveur conçu pour mesurer leur robustesse.

La **fiabilité** est la probabilité pour qu'un système produise les résultats attendus :

- Généralement mesuré en temps moyen entre échecs (*Mean Time Between Failures*, MTBF) et
- Amélioré par les fonctionnalités qui aident à éviter, à détecter et à réparer les fautes matérielles

La **disponibilité** est la probabilité pour qu'un système soit opérationnel à un moment donné :

- Généralement mesuré en pourcentage de temps d'arrêt par période de temps et
- Utilise souvent des mécanismes de détection et de correction des fautes matérielles à l'exécution.

La **serviabilité** est la simplicité et la rapidité avec laquelle un système peut être réparé ou maintenu :

- Généralement mesuré en temps moyen entre réparations (*Mean Time Between Repair*, MTBR).



Entre autres mesures de surveillance, les plus courantes sont :

- Le CPU — détecte les erreurs d'exécution d'instructions et au niveau des caches L1/L2/L3 ;
- La mémoire — ajoute des codes correcteurs d'erreur (*error correction codes*, ECC) et corrige les erreurs ;
- Les entrées-sorties — ajoute des sommes de contrôles CRC pour les données transférées ;
- Le stockage — RAID, systèmes de fichiers journalisés, sommes de contrôle, SMART (*Self-Monitoring, Analysis and Reporting Technology*).

En surveillant le nombre de détection d'erreurs, il est possible d'identifier si la probabilité d'erreurs matérielles augmente, et dans ce cas, d'effectuer une maintenance préventive pour remplacer un composant dégradé tant que les erreurs peuvent être corrigées.

Pour des informations détaillées sur les types d'événements d'erreur récupérés et comment les comprendre, voir le guide de l'administrateur du noyau sur <https://www.kernel.org/doc/html/latest/admin-guide/ras.html>.

**rasdaemon-service-type** [Variable Scheme]

Le type de service pour le service `rasdaemon`. Il accepte un objet `rasdaemon-configuration`. Instantiez-le avec

```
(service rasdaemon-service-type)
```

chargera une configuration par défaut, qui surveille tous les événements et les enregistre avec `syslogd`.

**rasdaemon-configuration** [Type de données]

Type de données représentant la configuration de `rasdaemon`.

`record?` (par défaut : `#f`)

Un booléen indiquant s'il faut enregistrer les événements dans une base de données SQLite. Cela donne un accès plus structuré aux informations contenues dans le fichier journal. L'emplacement de la base de données est codée en dur : `/var/lib/rasdaemon/ras-mc_event.db`.

## Service Zram

Le service de périphérique Zram fournit un périphérique de swap compressé en mémoire système. La documentation du noyau Linux a plus d'information à propos de ces périphériques `zram` (<https://www.kernel.org/doc/html/latest/admin-guide/blockdev/zram.html>).

**zram-device-service-type** [Variable Scheme]

Ce service crée un périphérique de bloc `zram`, le formate en swap et active l'espace d'échange. La valeur du service est un enregistrement `zram-device-configuration`.

**zram-device-configuration** [Type de données]

C'est le type de données représentant la configuration du service `zram-device`.

`size` (par défaut : `"1G"`)

C'est l'espace que vous voulez fournir à votre périphérique `zram`. Il accepte une chaîne et peut être un nombre d'octets ou utiliser un suffixe, p. ex. : `"512M"` ou `"1024000"`.

`compression-algorithm` (par défaut : `'lzo'`)

C'est l'algorithme de compression à utiliser. Il est difficile de lister toutes les possibilités, mais les options habituelles prises en charge par le noyau Linux Libre de Guix sont `'lzo'`, `'lz4'` et `'zstd'`.

`memory-limit` (par défaut : `0`)

C'est la quantité de mémoire maximal que le périphérique zram peut utiliser. le mettre à « 0 » désactive la limite. Bien qu'il soit généralement accepté que la compression aura un ratio de 2 pour 1, il est possible que des données non-comprimables soient écrites en espace d'échange et c'est une méthode pour limiter la quantité de mémoire qui peut être utilisée. Le paramètre accepte une chaîne qui peut être un nombre d'octets ou utiliser un suffixe, p. ex. `"2G"`.

`priority` (par défaut : `#f`)

C'est la priorité du périphérique d'échange créé à partir du périphérique zram. Voir Section 12.5 [Espace d'échange], page 258, pour une description des priorités des espaces d'échange. Vous voudrez peut-être indiquer une priorité spécifique pour le périphérique zram, sinon il pourrait ne pas être utilisé du tout pour les raisons qui y sont décrites.

### 12.9.35 Services Hurd

`hurd-console-service-type` [Variable Scheme]

Ce service démarre le client VGA sophistiqué en console sur le Hurd.

La valeur du service est un enregistrement `hurd-console-configuration`.

`hurd-console-configuration` [Type de données]

C'est le type de données représentant la configuration de `hurd-console-service`.

`hurd` (par défaut : `hurd`)

Le paquet Hurd à utiliser.

`hurd-getty-service-type` [Variable Scheme]

Ce service démarre un tty avec le programme `getty`.

La valeur du service est un enregistrement `hurd-getty-configuration`.

`hurd-getty-configuration` [Type de données]

Ce type de données représente la configuration de `hurd-getty-service`.

`hurd` (par défaut : `hurd`)

Le paquet Hurd à utiliser.

`tty` Le nom de la console sur laquelle tourne ce Getty, p. ex. `"tty1"`.

`baud-rate` (par défaut : `38400`)

Un entier spécifiant le taux de Baud de ce tty.

### 12.9.36 Services divers

## Service d'empreintes digitales

Le module (`gnu services authentication`) fournit un service DBus pour lire et identifier les empreintes digitales via un lecteur d'empreinte.

`fprintd-service-type` [Variable Scheme]

Le type de service pour `fprintd`, qui fournit des capacités de lecture d'empreinte.

```
(service fprintd-service-type)
```

## Service de contrôle du système

Le module (`gnu services sysctl`) fournit un service pour configurer les paramètres du noyau au démarrage.

`sysctl-service-type` [Variable Scheme]

Le type de service pour `sysctl`, qui modifie les paramètres du noyau dans `/proc/sys/`. Pour activer le transfert d'IPv4, vous pouvez l'instancier ainsi :

```
(service sysctl-service-type
  (sysctl-configuration
    (settings '(("net.ipv4.ip_forward" . "1")))))
```

Comme le service `sysctl-service-type` fait partie des services par défaut, `%base-services` et `%desktop-services`, vous pouvez utiliser `modify-services` pour changer sa configuration et ajouter les paramètres du noyau que vous voulez (voir Section 12.18.3 [Référence de service], page 602).

```
(modify-services %base-services
  (sysctl-service-type config =>
    (sysctl-configuration
      (settings (append '(("net.ipv4.ip_forward" . "1"))
        %default-sysctl-settings)))))
```

`sysctl-configuration` [Type de données]

Le type de données représentant la configuration de `sysctl`.

`sysctl` (par défaut : `(file-append procps "/sbin/sysctl")`)

L'exécutable `sysctl` à utiliser.

`settings` (par défaut : `%default-sysctl-settings`)

Une liste d'association spécifiant les paramètres du noyau et leur valeur.

`%default-sysctl-settings` [Variable Scheme]

Une liste d'association spécifiant les paramètres `sysctl` par défaut sur le système Guix.

## Service du démon PC/SC Smart Card

Le module (`gnu services security-token`) fournit le service suivant qui lance `pcscd`, le démon PC/SC Smart Card. `pcscd` est le démon pour `pcsc-lite` et `MuscleCard`. C'est un gestionnaire de ressource qui coordonne les communications avec les lecteurs de smart cards, les smart cards et les jetons cryptographiques connectés au système.

**pcscd-service-type** [Variable Scheme]

Le type de service pour le service `pcscd`. Sa valeur doit être un objet `pcscd-configuration`. Pour lancer `pcscd` dans sa configuration par défaut, instantiez-le avec :

```
(service pcscd-service-type)
```

**pcscd-configuration** [Type de données]

Type de données représentant la configuration de `pcscd`.

**pcsc-lite** (par défaut : `pcsc-lite`)

Le paquet `pcsc-lite` qui fournit `pcscd`.

**usb-drivers** (par défaut : `(list ccid)`)

Liste des paquets qui fournissent des pilotes USB à `pcscd`. Les pilotes doivent être dans `pcsc/drivers` dans le répertoire du dépôt du paquet.

## Service Lirc

Le module (`gnu services lirc`) fournit le service suivant.

**lirc-service** [`#:lirc lirc`] [`#:device #f`] [`#:driver #f`] [Procédure Scheme]  
[`#:config-file #f`] [`#:extra-options '()`]

Renvoie un service qui lance LIRC (<http://www.lirc.org>), un démon qui décode les signaux infrarouges des télécommandes.

Éventuellement, `device`, `driver` et `config-file` (le nom du fichier de configuration) peuvent être spécifiés. Voir le manuel de `lircd` pour plus de détails.

Enfin, `extra-options` est une liste d'options de la ligne de commande supplémentaires à passer à `lircd`.

## Service Spice

Le module (`gnu services spice`) fournit le service suivant.

**spice-vdagent-service** [`#:spice-vdagent`] [Procédure Scheme]

Renvoie un service qui lance VDAGENT (<https://www.spice-space.org>), un démon qui permet le partage du presse-papier avec une vm et de configurer la résolution d'affichage du client lorsque la fenêtre de la console graphique est redimensionnée.

## service inputattach

Le service `inputattach` (<https://linuxwacom.github.io/>) vous permet d'utiliser des périphériques d'entrée comme les tablettes Wacom, les écrans tactiles ou les joysticks avec le serveur d'affichage Xorg.

**inputattach-service-type** [Variable Scheme]

Type d'un service qui lance `inputattach` sur un appareil et envie les événements qu'il reçoit.

**inputattach-configuration** [Type de données]

**device-type** (par défaut : `"wacom"`)

Le type du périphérique à gérer. Lancez `inputattach --help`, du paquet `inputattach`, pour voir la liste des types de périphériques supportés.

- device** (par défaut : `"/dev/ttyS0"`)  
Le fichier de périphérique pour s'y connecter.
- baud-rate** (par défaut : `#f`)  
Taux de Baud à utiliser pour la connexion série. Cela doit être un nombre ou `#f`.
- log-file** (par défaut : `#f`)  
Si la valeur est vraie, cela doit être le nom d'un fichier où enregistrer les messages.

## Service de dictionnaire

Le module (`gnu services dict`) fournit le service suivant :

- dicod-service-type** [Variable Scheme]  
C'est le type de service qui lance le démon `dicod`, une implémentation du serveur DICT (voir Section "Dicod" dans *GNU Dico Manual*).
- dicod-service** [`#:config (dicod-configuration)`] [Procédure Scheme]  
Renvoie un service qui lance le démon `dicod`, une implémentation du serveur DICT (voir Section "Dicod" dans *GNU Dico Manual*).  
L'argument `config` facultatif spécifie la configuration pour `dicod`, qui devrait être un objet `<dicod-configuration>`, par défaut il sert le dictionnaire international collaboratif de GNU pour l'anglais.  
Vous pouvez ajouter `open localhost` à votre fichier `~/ .dico` pour faire de `localhost` le serveur par défaut du client `dico` (voir Section "Initialization File" dans *GNU Dico Manual*).
- dicod-configuration** [Type de données]  
Type de données représentant la configuration de `dicod`.
- dico** (par défaut : `dico`)  
Objet de paquet du serveur de dictionnaire GNU Dico.
- interfaces** (par défaut : `'("localhost")`)  
C'est la liste des adresses IP et des ports et éventuellement des noms de fichiers de socket sur lesquels écouter (voir Section "Server Settings" dans *GNU Dico Manual*).
- handlers** (par défaut : `'()`)  
Liste des objets `<dicod-handler>` qui définissent des gestionnaires (des instances de modules).
- databases** (par défaut : `(list %dicod-database:gcide)`)  
Liste d'objets `<dicod-database>` qui définissent des dictionnaires à servir.
- dicod-handler** [Type de données]  
Type de données représentant un gestionnaire de dictionnaire (instance de module).
- name** Nom du gestionnaire (instance de module).

**module** (par défaut : *#f*)  
 Nom du module dicod du gestionnaire (instance). Si la valeur est *#f*, le module a le même nom que le gestionnaire. (voir Section “Modules” dans *GNU Dico Manual*).

**options** Liste de chaînes ou de gexps représentant les arguments pour le gestionnaire de module

**dicod-database** [Type de données]  
 Type de données représentant une base de données de dictionnaire.

**name** Nom de la base de données, qui sera utilisée dans les commande DICT.

**handler** Nom du gestionnaire dicod (instance de module) utilisé par cette base de données (voir Section “Handlers” dans *GNU Dico Manual*).

**complex?** (par défaut : *#f*)  
 Indique si la configuration est pour une base de données complexe. La configuration complexe a besoin d’un objet *<dicod-handler>* correspondant, sinon inutile.

**options** Liste de chaînes ou de gexps représentant les arguments pour la base de données (voir Section “Databases” dans *GNU Dico Manual*).

**%dicod-database:gcide** [Variable Scheme]  
 Un objet *<dicod-database>* servant le dictionnaire international collaboratif en anglais via le paquet *gcide*.

Voici un exemple de configuration de *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))))))
```

## Service Docker

Le module (*gnu services docker*) fournit les services suivants.

**docker-service-type** [Variable Scheme]  
 C’est le type du service qui lance Docker (<https://www.docker.com>), un démon qui peut exécuter des lots applicatifs (aussi appelés « conteneurs ») dans des environnements isolés.

**docker-configuration** [Type de données]  
 Le type de données qui représente la configuration de Docker et Containerd.

**docker** (par défaut : `docker`)  
 Le paquet du démon Docker à utiliser.

**docker-cli** (par défaut : `docker-cli`)  
 Le paquet du client Docker à utiliser.

**containerd** (par défaut : `containerd`)  
 Le paquet Containerd à utiliser.

**proxy** (par défaut : `docker-libnetwork-cmd-proxy`)  
 Le paquet du mandataire réseau en espace utilisateur de Docker à utiliser.

**enable-proxy?** (par défaut : `#t`)  
 Indique s'il faut utiliser le mandataire réseau en espace utilisateur de Docker.

**debug?** (par défaut : `#f`)  
 Indique s'il faut activer la sortie de débogage.

**enable-iptables?** (par défaut : `#t`)  
 Indique s'il faut ajouter des règles iptables.

**environment-variables** (par défaut : `()`)  
 Liste de variables d'environnement à initialiser pour `dockerd`.  
 Cela doit être une liste de chaînes où chaque chaîne a pour forme '`clé=valeur`' comme dans cet exemple :

```
(list "LANGUAGE=eo:ca:eu"
      "TMPDIR=/tmp/dockerd")
```

**singularity-service-type** [Variable Scheme]  
 C'est le type de service qui vous permet de lancer Singularity (<https://www.sylabs.io/singularity/>), un outil similaire à Docker pour créer et lancer des lots applicatifs (aussi appelés « conteneurs »). la valeur de ce service est le paquet Singularity à utiliser.

Le service n'installe pas de démon : à la place, il installe des utilitaires en `setuid-root` (voir Section 12.10 [Programmes `setuid`], page 571) pour que les utilisateurs non privilégiés puisse invoquer `singularity run` et les commandes similaires.

## Service `auditd`

Le module (`gnu services auditd`) fournit le service suivant.

**auditd-service-type** [Variable Scheme]  
 C'est le type du service qui lance `auditd` (<https://people.redhat.com/sgrubb/audit/>), un démon qui suit les informations de sécurité de votre système.

Exemples de ce qui peut être suivi :

1. Les accès aux fichiers
2. Les appels système

3. Les commandes invoquées
4. Les tentatives de connexion échouées
5. Le filtrage du pare-feu
6. Les accès réseau

`auditctl` du paquet `audit` peut être utilisé pour ajouter ou supprimer des événements à suivre (jusqu'au prochain redémarrage). Pour suivre les événements de manière permanente, ajoutez des arguments à la commande `auditctl` dans un fichier nommé `audit.rules` dans le répertoire de configuration (voir plus bas). `aureport` du paquet `audit` peut être utilisé pour visualiser un rapport de tous les événements enregistrés. le démon d'audit enregistre par défaut dans le fichier `/var/log/audit.log`.

`auditd-configuration` [Type de données]

Le type de données qui représente la configuration de `auditd`.

`audit` (par défaut : `audit`)

Le paquet `audit` à utiliser.

`configuration-directory` (par défaut :

`%default-auditd-configuration-directory`)

Le répertoire contenant le fichier de configuration du paquet `audit`, qui doit être nommé `auditd.conf`, et éventuellement des règles d'audit à instancier au démarrage.

## Service R-Shiny

Le module (`gnu services science`) fournit le service suivant.

`rshiny-service-type` [Variable Scheme]

C'est le type de service utilisé pour lancer une appli web créée avec `r-shiny`. Ce service initialise la variable d'environnement `R_LIBS_USER` et lance le script fournit pour appeler `runApp`.

`rshiny-configuration` [Type de données]

Le type de données qui représente la configuration de `rshiny`.

`package` (par défaut : `r-shiny`)

Le paquet à utiliser.

`binary` (par défaut : `"rshiny"`)

Le nom du binaire ou du script shell situé dans `package/bin/` à lancer au démarrage du service.

La manière habituelle de créer ce fichier est :

```
...
(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))))
```

## Service Nix

Le module (`gnu services nix`) fournit le service suivant.

`nix-service-type` [Variable Scheme]

C'est le type du service qui lance le démon de construction du gestionnaire de paquets Nix (<https://nixos.org/nix/>). Voici un exemple qui montre comment l'utiliser :

```
(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)))
```

Après `guix system reconfigure`, configurez Nix pour votre utilisateur :

- Ajoutez un canal Nix et mettez-le à jour. Voir Guide du gestionnaire de paquets Nix (<https://nixos.org/nix/manual/>).
- Créez un lien symbolique vers votre profil et activez le profil de Nix :

```
$ ln -s "/nix/var/nix/profiles/per-user/$USER/profile" ~/.nix-profile
$ source /run/current-system/profile/etc/profile.d/nix.sh
```

`nix-configuration` [Type de données]

Ce type de données représente la configuration du démon Nix.

`nix` (par défaut : `nix`)

Le paquet Nix à utiliser.

`sandbox` (par défaut : `#t`)

Spécifie si les constructions sont effectuées dans un bac à sable par défaut.

`build-sandbox-items` (par défaut : `'()`)

C'est une liste de chaînes de caractères ou d'objets ajoutés au champ `build-sandbox-items` du fichier de configuration.

`extra-config` (par défaut : `'()`)

C'est une liste de chaînes de caractères ou d'objets ajoutés au fichier de configuration. Elle est utile pour ajouter du texte supplémentaire directement dans le fichier de configuration.

`extra-options` (par défaut : '()')

Options supplémentaires de la ligne de commande pour `nix-service-type`.

## Service Fail2Ban

`fail2ban` (<http://www.fail2ban.org/>) scanne les fichiers journaux (p. ex. `/var/log/apache/error_log`) et bannit les adresses IP qui montrent des signes de malhonnêteté — des échecs de connexion répétés, des tentatives d'utiliser des exploits, etc.

Le type de service `fail2ban-service-type` est fournit par le module (`gnu services security`).

Ce type de service lance le démon `fail2ban`. Il peut être configuré de plusieurs façon qui sont :

### Configuration basique

Les paramètres de base du service Fail2Ban peut être configurés via sa configuration `fail2ban`, qui est documentée ci-dessous.

### Extensions personnalisées des prisons

La fonction `fail2ban-jail-service` peut être utilisée pour ajouter de nouvelles prisons Fail2Ban.

### Mécanisme d'extension du Shepherd

Les développeurs de services peuvent étendre le type de service `fail2ban-service-type` lui-même avec le mécanisme d'extension habituel des services.

`fail2ban-service-type` [Variable Scheme]

C'est le type de service qui lance le démon `fail2ban`. Voici un exemple de configuration de base explicite :

```
(append
 (list
  (service fail2ban-service-type
   (fail2ban-configuration
    (extra-jails
     (list
      (fail2ban-jail-configuration
       (name "sshd")
       (enabled? #t))))))
  ;; Il n'y a pas de dépendance implicite à un service SSH,
  ;; donc il faut en fournir un.
  (service openssh-service-type))
 %base-services)
```

`fail2ban-jail-service svc-type jail` [Procédure Scheme]

Étend `svc-type`, un objet `<service-type>` avec `jail`, un objet `fail2ban-jail-configuration`.

Par exemple :

```
(append
 (list
```

```
(service
  ;; La procédure « fail2ban-jail-service » peut étendre n'importe quel type de
  ;; avec une prison fail2ban. Cela supprime la nécessité d'étendre les services
  ;; avec le fail2ban-service-type.
  (fail2ban-jail-service
    openssh-service-type
    (fail2ban-jail-configuration
      (name "sshd")
      (enabled? #t)))
    (openssh-configuration ...))))
```

Ci-dessous se trouve la référence des différents enregistrement de configuration `jail-service-type`.

**fail2ban-configuration** [Type de données]

Les champs de `fail2ban-configuration` disponibles sont :

**fail2ban** (par défaut : `fail2ban`) (type : paquet)  
Le paquet `fail2ban` à utiliser. Il est utilisé à la fois pour les binaires et pour la configuration de base par défaut qui doit être étendue avec des objets `<fail2ban-jail-configuration>`.

**run-directory** (par défaut : `"/var/run/fail2ban"`) (type : chaîne)  
Le répertoire d'état pour le démon `fail2ban`.

**jails** (par défaut : `()`) (type : liste-de-fail2ban-jail-configurations)  
Instances de `<fail2ban-jail-configuration>` collectées par les extensions.

**extra-jails** (par défaut : `()`) (type : liste-de-fail2ban-jail-configurations)  
Instances de `<fail2ban-jail-configuration>` fournies explicitement.

**extra-content** (par défaut : `()`) (type : text-config)  
Contenu brut supplémentaire à ajouter à la fin du fichier `jail.local`, à fournir sous forme d'une liste de simili-fichiers.

**fail2ban-ignore-cache-configuration** [Type de données]

Les champs de `fail2ban-ignore-cache-configuration` disponibles sont :

**key** (type : chaîne)  
Clé de cache.

**max-count** (type : entier)  
Taille du cache.

**max-time** (type : entier)  
Durée du cache.

**fail2ban-jail-action-configuration** [Type de données]

Les champs de `fail2ban-jail-action-configuration` disponibles sont :

**name** (type : string)  
Nom de l'action.

**arguments** (par défaut : ()) (type : liste-d'arguments)  
Arguments de l'action.

**fail2ban-jail-configuration** [Type de données]

Les champs de **fail2ban-jail-configuration** disponibles sont :

**name** (type : string)

Nom requis de cette configuration de prison.

**enabled?** (par défaut : #t) (type : booléen)

Spécifie si cette prison est activée.

**backend** (type : peut-être-symbole)

Le moteur à utiliser pour détecter les changements dans **log-path**. La valeur par défaut est 'auto. Pour consulter les valeurs par défaut de la configuration de prison, consultez le fichier `/etc/fail2ban/jail.conf` du paquet **fail2ban**.

**max-retry** (type : peut-être-entier)

Le nombre d'échecs avant qu'un hôte ne soit banni (p. ex. (**max-retry** 5)).

**max-matches** (type : peut-être-entier)

Le nombre de correspondances stockées dans le ticket (accessible avec l'étiquette `<matches>`) dans l'action.

**find-time** (type : peut-être-chaîne)

La fenêtre pendant laquelle le compte de tentative maximale doit être atteint pour qu'une adresse IP soit bannie. Un hôte est banni s'il a généré **max-retry** pendant les dernières **find-time** secondes (p. ex. `»find-time "10m"`). Elle peut être fournie en seconde ou avec le « format temporel abrégé » de Fail2ban, décrit dans `man 5 jail.conf`.

**ban-time** (type : peut-être-chaîne)

La durée, en secondes ou au format temporel abrégé, qu'un bannissement doit durer (p. ex. (**ban-time** "10m")).

**ban-time-increment?** (type : peut-être-booléen)

Indique s'il faut considérer les bannissements passés pour calculer un incrément du temps de bannissement par défaut d'une adresse IP particulière.

**ban-time-factor** (type : peut-être-chaîne)

Le coefficient à utiliser pour calculer un temps de bannissement exponentiel.

**ban-time-formula** (type : peut-être-chaîne)

C'est la formule utilisée pour calculer la prochaine valeur d'une durée de bannissement.

**ban-time-multipliers** (type : peut-être-chaîne)

Utilisé pour calculer la prochaine valeur de la durée du bannissement au lieu de la formule.

- ban-time-max-time** (type : peut-être-chaine)  
Le nombre maximum de secondes qu'un bannissement peut durer.
- ban-time-rnd-time** (type : peut-être-chaine)  
Le nombre maximum de secondes maximum qu'un bannissement à durée aléatoire peut durer. C'est utile pour arrêter les robots « intelligents » qui calculent le moment exacte où une adresse IP peut être dé-bannie.
- ban-time-overall-jails?** (type : peut-être-booléen)  
Lorsque la valeur est vraie, elle spécifie que la recherche d'une adresse IP dans la base de donnée doit se faire dans toutes les prisons. Sinon, seule la prison actuelle pour l'adresse IP bannie est prise en compte.
- ignore-self?** (type : peut-être-booléen)  
Ne jamais bannir l'adresse IP de la machine locale.
- ignore-ip** (par défaut : ()) (type : liste-de-chaines)  
Une liste d'adresse IP, de masques CIDR ou d'hôtes DNS à ignorer. **fail2ban** ne bannira par un hôte qui correspond à une adresse de cette liste.
- ignore-cache** (type : peut-être-fail2ban-ignore-cache-configuration)  
fournit des paramètres de cache pour ignorer les vérifications d'échecs.
- filter** (type : peut-fail2ban-jail-filter-configuration)  
Le filtre à utiliser par la prison, spécifiée par un objet `<fail2ban-jail-filter-configuration>`. Par défaut, les prisons ont un nom qui correspond au nom de leur filtre.
- log-time-zone** (type : peut-être-chaine)  
Le fuseau horaire par défaut pour les lignes de journaux qui n'en indiquent pas.
- log-encoding** (type : peut-être-symbole)  
L'encodage des journaux traités par la prison. Les valeurs possibles sont : 'ascii', 'utf-8 et 'auto.
- log-path** (par défaut : ()) (type : liste-de-chaines)  
Le nom de fichier des journaux à surveiller.
- action** (par défaut : ()) (type : liste-de-fail2ban-jail-action)  
Une liste de `<fail2ban-jail-action-configuration>`.
- extra-content** (par défaut : ()) (type : text-config)  
Contenu supplémentaire pour la configuration de la prison, à fournir sous la forme d'une liste de simili-fichiers.

**fail2ban-jail-filter-configuration** [Type de données]

Les champs de `fail2ban-jail-filter-configuration` disponibles sont :

- name** (type : string)  
Filtre à utiliser.
- mode** (type : peut-être-chaine)  
Mode de filtrage.

## 12.10 Programmes setuid

Certains programmes doivent être lancés avec des privilèges élevés même lorsqu'ils sont lancés par un utilisateur non privilégié. Un exemple notoire est le programme `passwd`, que les utilisateurs peuvent appeler pour modifier leur mot de passe et qui doit accéder à `/etc/passwd` et `/etc/shadow` — ce qui est normalement réservé à `root`, pour des raisons de sécurité évidentes. Pour contourner cela, `passwd` devrait être `setuid-root`, ce qui signifie qu'il sera toujours lancé avec les privilèges `root` (voir Section "How Change Persona" dans *The GNU C Library Reference Manual*, pour plus d'informations sur le mécanisme `setuid`).

Le dépôt lui-même ne *peut pas* contenir de programmes `setuid` ; cela serait un problème de sécurité puisque n'importe quel utilisateur du système peut écrire une dérivation qui remplit le dépôt (voir Section 9.9 [Le dépôt], page 157). Donc, un mécanisme différent est utilisé : au lieu de changer les bits `setuid` ou `setgid` directement sur les fichiers qui sont dans le dépôt, nous laissons à l'administrateur système le soin de *déclarer* les programmes qui devraient avoir ces privilèges supplémentaires.

Le champ `setuid-programs` d'une déclaration `operating-system` contient une liste de `<setuid-program>` qui dénotent les noms des programmes qui auront le bit `setuid` ou `setgid` (voir Section 12.1 [Utiliser le système de configuration], page 238). Par exemple, le programme `mount.nfs`, qui fait partie du paquet `nfs-utils`, avec le bit `setuid root`, peut être désigné comme ceci :

```
(setuid-program
  (program (file-append nfs-utils "/sbin/mount.nfs")))
```

Ensuite, pour rendre `mount.nfs` `setuid` sur votre système, ajoutez l'exemple précédent à votre déclaration de système d'exploitation en l'ajoutant aux `%setuid-programs` de cette façon :

```
(operating-system
  ;; Certains champs omis...
  (setuid-programs
    (append (list (setuid-program
                  (program (file-append nfs-utils "/sbin/mount.nfs")))))
            %setuid-programs)))
```

`setuid-program` [Type de données]

Ce type de données représente un programme avec le bit `setuid` ou `setgid`.

`program` Un objet simili-fichier dont le bit `setuid` ou `setgid` est activé.

`setuid?` (par défaut : `#t`)

Indique s'il faut activer le bit `setuid`.

`setgid?` (par défaut : `#f`)

Indique s'il faut active le bit `setgid`.

`user` (par défaut : `0`)

L'UID (entier) ou le nom d'utilisateur (chaîne) du propriétaire du programme, `root` par défaut.

`group` (par défaut : `0`)

Le GID (entier) ou le nom du groupe (chaîne) pour le groupe propriétaire du programme, `root` par défaut.

Un ensemble de programmes par défaut est défini par la variable `%setuid-programs` du module (`gnu system`).

`%setuid-programs` [Variable Scheme]

Une liste de `<setuid-program>` qui dénotent les programmes communément `setuid-root`.

La liste inclus des commandes comme `passwd`, `ping`, `su` et `sudo`.

Sous le capot, les programmes `setuid` sont créés dans le répertoire `/run/setuid-programs` au moment de l'activation du système. Les fichiers dans ce répertoire se réfèrent aux « vrais » binaires, qui sont dans le dépôt.

## 12.11 Certificats X.509

Les serveurs web disponibles par HTTPS (c'est-à-dire HTTP sur le mécanisme de la couche de transport sécurisée, TLS) envoient aux clients un *certificat X.509* que les clients peuvent utiliser pour *authentifier* le serveur. Pour cela, les clients vérifient que le certificat du serveur est signé par une *autorité de certification* (AC ou CA). Mais pour vérifier la signature de la CA, les clients doivent d'abord avoir récupéré le certificat de la CA.

Les navigateurs web comme GNU IceCat incluent leur propre liste de certificats, pour qu'ils puissent vérifier les signatures des CA directement.

Cependant, la plupart des autres programmes qui peuvent parler HTTPS — `wget`, `git`, `w3m`, etc — doivent savoir où trouver les certificats des CA.

Dans Guix, cela se fait en ajoutant un paquet qui fournit les certificats dans le champ `packages` de la déclaration `operating-system` (voir Section 12.2 [référence de `operating-system`], page 246). Guix inclut l'un de ces paquets, `nss-certs`, qui est un ensemble de certificats de CA fourni par les services de sécurité réseau de Mozilla (`nss`).

Remarquez qu'il ne fait *pas* partie de `%base-packages`, donc vous devez explicitement l'ajouter. Le répertoire `/etc/ssl/certs`, là où la plupart des applications et bibliothèques vont rechercher les certificats par défaut, pointe vers les certificats installés globalement.

Les utilisateurs non privilégiés, dont les utilisateurs de Guix sur une distro externe, peuvent aussi installer leur propre paquet de certificats dans leur profil. Un certain nombre de variables d'environnement doivent être définies pour que les applications et les bibliothèques puissent les trouver. En particulier, la bibliothèque OpenSSL prend en compte les variables `SSL_CERT_DIR` et `SSL_CERT_FILE`. Certaines applications ajoutent leurs propres variables, par exemple le système de contrôle de version Git prend en compte le lot de certificats pointé par la variable d'environnement `GIT_SSL_CAINFO`. Ainsi, vous lanceriez quelque chose comme ceci :

```
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"
```

Un autre exemple serait R, qui a besoin que la variable d'environnement `CURL_CA_BUNDLE` pointe sur le lot de certificats, donc vous lanceriez quelque chose comme ceci :

```
guix install nss-certs
export CURL_CA_BUNDLE="$HOME/.guix-profile/etc/ssl/certs/ca-certificates.crt"
```

Pour d'autres applications vous pourriez avoir besoin de chercher la variable d'environnement requise dans leur documentation.

## 12.12 Name Service Switch

Le module (`gnu system nss`) fournit des liaisons pour le fichier de configuration du *name service switch* ou *NSS* de la *libc* (voir Section “NSS Configuration File” dans *The GNU C Library Reference Manual*). En résumé, NSS est un mécanisme qui permet à la *libc* d'être étendue avec de nouvelles méthodes de résolution de « noms » dans les bases de données du système, comme les noms d'hôtes, les noms des services, les comptes utilisateurs et bien plus (voir Section “Name Service Switch” dans *The GNU C Library Reference Manual*).

La configuration de NSS spécifie, pour chaque base de données du système, quelle méthode de résolution utiliser, et comment les diverses méthodes sont enchaînées — par exemple, sous certaines circonstances, NSS devrait essayer la méthode suivante de la liste. La configuration de NSS est donnée dans le champ `name-service-switch` de la déclaration `operating-system` (voir Section 12.2 [référence de `operating-system`], page 246).

Par exemple, la déclaration ci-dessous configure NSS pour utiliser le moteur `nss-mdns` (<https://0pointer.de/lennart/projects/nss-mdns/>), qui supporte la résolution de nom d'hôte sur le DNS multicast (mDNS) pour les noms d'hôtes terminant par `.local` :

```
(name-service-switch
  (hosts (list %files      ; d'abord, vérifier /etc/hosts

          ;; Si ce qui précède n'a pas fonctionné, essayer
          ;; avec « mdns_minimal ».
          (name-service
            (name "mdns_minimal")

            ;; « mdns_minimal » fait autorité pour
            ;; « .local ». Lorsqu'il renvoie « pas trouvé »,
            ;; inutile d'essayer la méthode suivante.
            (reaction (lookup-specification
                      (not-found => return))))

          ;; Puis revenir sur DNS.
          (name-service
            (name "dns"))

          ;; Enfin, essayer avec « mdns complet ».
          (name-service
            (name "mdns")))))
```

Ne vous inquiétez pas : la variable `%mdns-host-lookup-nss` (voir plus bas) contient cette configuration, donc vous n'avez pas besoin de tout taper si vous voulez simplement que la résolution de nom en `.local` fonctionne.

Remarquez que dans ce cas, en plus de mettre en place le `name-service-switch` de la déclaration `operating-system`, vous devez aussi utiliser `avahi-service-type` (voir Section 12.9.5 [Services réseau], page 299), ou `%desktop-services` qui l'inclut (voir Sec-



tion 12.9.9 [Services de bureaux], page 347). Cela rend `nss-mdns` accessible au démon de cache du service de nom (voir Section 12.9.1 [Services de base], page 268).

Pour votre confort, les variables suivantes contiennent des configurations NSS typiques.

`%default-nss` [Variable Scheme]  
C'est la configuration NSS par défaut, un objet `name-service-switch`.

`%mdns-host-lookup-nss` [Variable Scheme]  
C'est la configuration NSS avec le support de la résolution de noms sur DNS multicast (mDNS) pour les noms d'hôtes en `.local`.

La référence pour la configuration de NSS est donnée ci-dessous. C'est une correspondance directe avec le format de fichier de la bibliothèque C, donc référez-vous au manuel de la bibliothèque C pour plus d'informations (voir Section "NSS Configuration File" dans *The GNU C Library Reference Manual*). Comparé au format de fichier de configuration de NSS, cette configuration a l'avantage non seulement d'ajouter ces bonnes vieilles parenthèses, mais aussi des vérifications statiques ; vous saurez s'il y a des erreurs de syntaxe et des coquilles dès que vous lancerez `guix system`.

`name-service-switch` [Type de données]  
C'est le type de données représentant la configuration de NSS. Chaque champ ci-dessous représente l'un des système de bases de données supportés.

`aliases`  
`ethers`  
`group`  
`gshadow`  
`hosts`  
`initgroups`  
`netgroup`  
`networks`  
`password`  
`public-key`  
`rpc`  
`services`

`shadow` Les bases de données du système gérées par NSS. Chaque champ doit être une liste d'objets `<name-service>` (voir plus bas).

`name-service` [Type de données]  
C'est le type de données représentant un service de noms et l'action de résolution associée.

`name` Une chaîne dénotant le service de nom (voir Section "Services in the NSS configuration" dans *The GNU C Library Reference Manual*).

Remarquez que les services de dnoms listés ici doivent être visibles à `nscd`. Cela se fait en passant la liste des paquets fournissant les services de noms à l'argument `#:name-services` de `nscd-service` (voir Section 12.9.1 [Services de base], page 268).

**reaction** Une action spécifiée par la macro `lookup-specification` (voir Section “Actions in the NSS configuration” dans *The GNU C Library Reference Manual*). Par exemple :

```
(lookup-specification (unavailable => continue)
                    (success => return))
```

## 12.13 Disque de RAM initial

Pour le démarrage, on passe au noyau Linux-Libre un *disque de RAM initial* ou *initrd*. Un *initrd* contient un système de fichier racine temporaire ainsi qu’un script d’initialisation. Ce dernier est responsable du montage du vrai système de fichier racine et du chargement des modules du noyau qui peuvent être nécessaires à cette tâche.

Le champ `initrd-modules` d’une déclaration `operating-system` vous permet de spécifier les modules du noyau Linux-Libre qui doivent être disponibles dans l’*initrd*. En particulier, c’est là où vous devez lister les modules requis pour effectivement piloter le disque dur où se trouve la partition racine — bien que la valeur par défaut de `initrd-modules` couvre la plupart des cas. Par exemple, en supposant que vous ayez besoin du module `megaraid_sas` en plus des modules par défaut pour accéder à votre système de fichiers racine, vous écririez :

```
(operating-system
  ;; ...
  (initrd-modules (cons "megaraid_sas" %base-initrd-modules)))
```

`%base-initrd-modules` [Variable Scheme]

C’est la liste des modules du noyau inclus dans l’*initrd* par défaut.

En plus, si vous avez besoin de paramètres plus bas niveau, le champ `initrd` d’une déclaration `operating-system` vous permet de spécifier quel *initrd* vous voudriez utiliser. Le module (`gnu system linux-initrd`) fournit trois manières de construire un *initrd* : la procédure `base-initrd` de haut niveau et les procédures `raw-initrd` et `expression->initrd` de bas niveau.

La procédure `base-initrd` est conçue pour couvrir la plupart des usages courants. Par exemple, si vous voulez ajouter des modules du noyau à charger au démarrage, vous pouvez définir le champ `initrd` de votre déclaration de système d’exploitation ainsi :

```
(initrd (lambda (file-systems . rest)
  ;; Crée un initrd standard mais paramètre le réseau
  ;; avec les paramètres que QEMU attend par défaut.
  (apply base-initrd file-systems
    #:qemu-networking? #t
    rest)))
```

La procédure `base-initrd` gère aussi les cas d’utilisation courants qui concernent l’utilisation du système comme client QEMU, ou comme un système « live » avec un système de fichier racine volatile.

La procédure `base-initrd` est construite à partir de la procédure `raw-initrd`. Contrairement à `base-initrd`, `raw-initrd` ne fait rien à haut-niveau, comme essayer de deviner les modules du noyau et les paquets qui devraient être inclus dans l’*initrd*. Un exemple

d'utilisation de `raw-initrd` serait si un utilisateur a une configuration personnalisée du noyau Linux et que les modules du noyau inclus par défaut par `base-initrd` ne sont pas disponibles.

Le disque de RAM initial produit par `base-initrd` ou `raw-initrd` prend en compte plusieurs options passées par la ligne de commande du noyau Linux (c'est-à-dire les arguments passés via la commande `linux` de GRUB ou l'option `-append` de QEMU), notamment :

`gnu.load=boot`

Dit au disque de RAM initial de charger `boot`, un fichier contenant un programme Scheme, une fois qu'il a monté le système de fichier racine.

Guix utilise cette option pour donner le contrôle à un programme de démarrage qui lance les programmes d'activation de services puis démarre le GNU Shepherd, le système d'initialisation.

`root=root`

Monte `root` comme système de fichier racine. `root` peut être un nom de périphérique comme `/dev/sda1`, une étiquette de système de fichiers ou un UUID de système de fichiers. Lorsque ce n'est pas spécifié, le nom de périphérique du système de fichier racine de la déclaration de système d'exploitation est utilisé.

`rootfstype=type`

Indique le type du système de fichiers racine. Il remplace le champ `type` du système de fichiers racine spécifié dans la déclaration `operating-system`, s'il existe.

`rootflags=options`

Indique les `options` de montage du système de fichiers racine. Elles remplacent le champ `options` du système de fichiers racine spécifié dans la déclaration `operating-system` s'il existe.

`fsck.mode=mode`

Contrôle s'il faut vérifier ou pas le système de fichier `root` avant de le monter. `mode` peut valoir `skip` (ne jamais vérifier), `force` (toujours vérifier), ou `auto` pour respecter le réglage `check?` de l'objet `<file-system>` pour la racine (voir Section 12.3 [Systèmes de fichiers], page 250) et n'exécuter une analyse complète que si le système de fichier n'a pas été éteint proprement.

`auto` est la valeur par défaut si cette option n'est pas renseignée ou si `mode` n'a pas l'une des valeurs ci-dessus.

`fsck.repair=niveau`

Le niveau de réparation à effectuer automatiquement si des erreurs sont détectées dans le système de fichier `root`. `level` prend les valeurs `no` (ne pas écrire du tout sur `root` si possible), `yes` (réparer autant que possible), ou `preen` pour réparer les problèmes considérés réparables automatiquement sans risque.

`preen` est la valeur par défaut si cette option n'est pas renseignée ou si `level` n'est pas l'une des valeurs ci-dessus.

- gnu.system=systeme**  
S'assure que `/run/booted-system` et `/run/current-system` pointent vers `system`.
- modprobe.blacklist=modules...**  
Dit au disque de RAM initial ainsi qu'à la commande `modprobe` (du paquet `kmod`) de refuser de charger `modules`. `modules` doit être une liste de noms de modules séparés par des virgules — p. ex. `usbkbd,9pnet`.
- gnu.repl** Démarre une boucle lecture-évaluation-affichage (REPL) depuis le disque de RAM initial avant qu'il n'essaye de charger les modules du noyau et de monter le système de fichiers racine. Notre équipe commerciale appelle cela *boot-to-Guile*. Le Schemeur en vous va adorer. Voir Section "Using Guile Interactively" dans *GNU Guile Reference Manual*, pour plus d'information sur le REPL de Guile.

Maintenant que vous connaissez toutes les fonctionnalités des disques de RAM initiaux produits par `base-initrd` et `raw-initrd`, voici comment l'utiliser le personnalisé plus avant.

**raw-initrd *file-systems* [#:linux-modules '()]** [Procédure Scheme]  
[#:pre-mount #t] [#:mapped-devices '()] [#:keyboard-layout #f]  
[#:helper-packages '()] [#:qemu-networking? #f] [#:volatile-root? #f]

Renvoie une dérivation qui construit un `initrd`. `file-systems` est une liste de systèmes de fichiers à monter par l'`initrd`, éventuellement en plus du système de fichier racine spécifié sur la ligne de commande du noyau via `root`. `linux-modules` est une liste de modules du noyau à charger au démarrage. `mapped-devices` est une liste de correspondances de périphériques à réaliser avant que les `file-systems` ne soient montés (voir Section 12.4 [Périphériques mappés], page 256). `helper-packages` est une liste de paquets à copier dans l'`initrd`. Elle peut inclure `e2fsck/static` ou d'autres paquets requis par l'`initrd` pour vérifier le système de fichiers racine.

Lorsque la valeur est vraie, `keyboard-layout` est un enregistrement `<keyboard-layout>` dénotant la disposition du clavier désirée pour la console. Cela est effectuée avant que les `mapped-devices` ne soient créés et avant que les `file-systems` ne soient montés, de sorte que, si l'utilisateur au besoin de saisir une phrase de passe ou d'utiliser le REPL, cela arrive avec la disposition du clavier voulue.

Lorsque `qemu-networking?` est vrai, paramètre le réseau avec les paramètres QEMU standards. Lorsque `virtio?` est vrai, charge des modules supplémentaires pour que l'`initrd` puisse être utilisé comme client QEMU avec les pilotes I/O para-virtualisés.

Lorsque `volatile-root?` est vrai, le système de fichier racine est inscriptible mais tous les changements seront perdus.

**base-initrd *file-systems* [#:mapped-devices '()]** [Procédure Scheme]  
[#:keyboard-layout #f] [#:qemu-networking? #f] [#:volatile-root? #f]  
[#:linux-modules '()]

Renvoie un objet simili-fichier contenant un `initrd` générique, avec les modules du noyau de `linux`. `file-systems` est une liste de systèmes de fichiers à monter par l'`initrd`, éventuellement en plus du système de fichiers racine spécifié sur la ligne de commande

du noyau via `root.mapped-devices` est une liste de correspondances de périphériques à réaliser avant de monter les `file-systems`.

Lorsque la valeur est vraie, `keyboard-layout` est un enregistrement `<keyboard-layout>` dénotant la disposition du clavier désirée pour la console. Cela est effectuée avant que les `mapped-devices` ne soient créés et avant que les `file-systems` ne soient montés, de sorte que, si l'utilisateur au besoin de saisir une phrase de passe ou d'utiliser le REPL, cela arrive avec la disposition du clavier voulue.

`qemu-networking?` et `volatile-root?` se comportent comme pour `raw-initrd`.

L'`initrd` est automatiquement remplie avec tous les modules du noyau requis pour `file-systems` et pour les options données. On peut lister des modules supplémentaires dans `linux-modules`. Ils seront ajoutés à l'`initrd` et chargés au démarrage dans l'ordre dans lequel ils apparaissent.

Inutile de le dire, les `initrds` que nous produisons et utilisons incluent une version de Guile liée statiquement, et le programme d'initialisation est un programme Guile. Cela donne beaucoup de flexibilité. La procédure `expression->initrd` construit un tel `initrd`, étant donné le programme à lancer dans cet `initrd`.

`expression->initrd exp [#:guile %guile-static-stripped] [Procédure Scheme]`  
`[#:name "guile-initrd"]`

Renvoie un objet simili-fichier contenant un `initrd` Linux (une archive `cpio` compressée avec `gzip`) contenant `guile` et qui évalue `exp`, une G-expression, au démarrage. Toutes les dérivations référencées par `exp` sont automatiquement copiées dans l'`initrd`.

## 12.14 Configuration du chargeur d'amorçage

Le système d'exploitation supporte plusieurs chargeurs d'amorçage. La configuration du chargeur d'amorçage se fait avec la déclaration `bootloader-configuration`. Tous les champs de cette structure sont indépendants du chargeur d'amorçage sauf un, `bootloader` qui indique le chargeur d'amorçage à configurer et à installer.

Certains chargeurs d'amorçage ne prennent pas en compte tous les champs de `bootloader-configuration`. Par exemple, le chargeur d'amorçage `extlinux` ne supporte pas les thèmes et ignore donc le champ `theme`.

`bootloader-configuration` [Type de données]

Le type d'une déclaration de configuration de chargeur d'amorçage.

`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.

Les chargeurs d'amorçage disponibles sont décrits dans les modules (`gnu bootloader ...`). En particulier, (`gnu bootloader u-boot`) contient des définitions de chargeurs d'amorçage pour une large gamme de systèmes ARM et AArch, à l'aide du chargeur d'amorçage U-Boot (<https://www.denx.de/wiki/U-Boot/>).

`grub-efi-bootloader` permet de démarrer sur un système moderne qui utilise l'UEFI (*Unified Extensible Firmware Interface*). C'est ce que vous devriez utiliser si l'image d'installation contient un répertoire `/sys/firmware/efi` lorsque vous démarrez dessus sur votre machine.

`grub-bootloader` vous permet de démarrer en particulier sur des machines Intel en mode BIOS « legacy ».

`grub-efi-netboot-bootloader` vous permet de démarrer votre système via le réseau avec TFTP. En plus d'un système de fichier racine NFS, cela vous permet de démarrer un système Guix sans disque.

The installation of the `grub-efi-netboot-bootloader` generates the content of the TFTP root directory at `targets` (voir Section 12.14 [Configuration du chargeur d'amorçage], page 578), 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.

Si vous voulez aussi utiliser un système de fichier racine NFS (en fait si vous montez le dépôt depuis un partage NFS), alors le serveur TFTP doit aussi servir le fichier `/boot/grub/grub.cfg` et d'autres fichiers à partir du dépôt (comme l'image de fond de GRUB, le noyau (voir Section 12.2 [référence de operating-system], page 246) et l'initrd (voir Section 12.2 [référence de operating-system], page 246)). Tous ces fichiers du dépôt seront accessibles pour GRUB à travers TFTP avec leurs chemins normaux, par exemple `tftp://tftp-server/gnu/store/...-initrd/initrd.cpio.gz`.

Deux liens symboliques sont créés pour rendre cela possible. Pour chaque cible du champ `targets`, le premier lien est `'target'/efi/Guix/boot/grub/grub.cfg` pointant vers `../../../../boot/grub/grub.cfg`, où `'target'` peut être `/boot`. Dans ce cas le lien ne quitte pas le répertoire racine du serveur TFTP. Le second lien est `'target'/gnu/store` qui pointe vers `../gnu/store`. Ce lien est en dehors du répertoire racine du serveur TFTP.

L'hypothèse derrière tout ceci est que vous avez un serveur NFS qui exporte le système de fichiers racine de votre système Guix, et en plus, un serveur TFTP qui exporte vos répertoires `targets` — habituellement `/boot` uniquement — à partir du même système de fichiers racine que votre système Guix. Dans ce cas les liens symboliques vont fonctionner.

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.

En dehors de `grub-efi-netboot-bootloader`, des serveurs TFTP et NFS déjà mentionnés, vous pouvez utiliser un serveur DHCP correcte-

ment configuré pour permettre le démarrage réseau. Pour cela nous ne pouvons que vous recommander de regarder les instructions sur PXE (Pre-boot eXecution Environment).

`grub-efi-removable-bootloader` vous permet de démarrer votre système à partir d'un média amovible en écrivant le fichier GRUB à l'emplacement spécifié par `UEFI/EFI/BOOT/BOOTX64.efi` du répertoire de démarrage, typiquement `/boot/efi`. C'est aussi utile pour certains micrologiciels UEFI qui « oublient » la configuration de leur stockage non-volatile. Comme pour `grub-efi-bootloader`, cela peut aussi être utilisé si le répertoire `/sys/firmware/efi` est disponible.

**Remarque:** Cela *remplacera* le fichier GRUB des autres systèmes d'exploitation qui placent aussi un fichier GRUB dans l'emplacement spécifié par UEFI. Cela les rends impossible à démarrer.

**targets** C'est une liste de chaînes qui dénotent les cibles sur lesquelles installer le chargeur d'amorçage.

L'interprétation des cibles dépend du chargeur d'amorçage en question. Pour `grub-bootloader` par exemple, cela devrait être le nom des périphériques compris par la commande `install` du chargeur d'amorçage, comme `/dev/sda` ou `(hd0)` (voir Section "Invoking grub-install" dans *GNU GRUB Manual*). Pour `grub-efi-bootloader` et `grub-efi-removable-bootloader`, cela devrait être les points de montage des systèmes de fichiers EFI, typiquement `/boot/efi`. Pour `grub-efi-netboot-bootloader`, `targets` doit contenir les points de montage correspondant aux répertoires racines de TFTP sur votre serveur TFTP.

**menu-entries** (par défaut : `()`)

Une liste éventuellement vide d'objets `menu-entry` (voir plus bas), dénotant les entrées qui doivent apparaître dans le menu du chargeur d'amorçage, en plus de l'entrée pour le système actuel et l'entrée pointant vers les générations précédentes.

**default-entry** (par défaut : `0`)

L'index de l'entrée du menu de démarrage par défaut. L'index 0 correspond au système actuel.

**timeout** (par défaut : `5`)

Le nombre de secondes à attendre une entrée clavier avant de démarrer. Indiquez 0 pour démarre immédiatement, et -1 pour attendre indéfiniment.

**keyboard-layout** (par défaut : `#f`)

Si c'est `#f`, le menu du chargeur d'amorçage (s'il y en a un) utilise la disposition du clavier par défaut, normalement pour l'anglais américain (« qwerty »).

Sinon, cela doit être un objet `keyboard-layout` (voir Section 12.7 [Disposition du clavier], page 263).

**Remarque:** Cette option est actuellement ignorée par les chargeurs d'amorçage autre que `grub` et `grub-efi`.

`theme` (par défaut : `#f`)

L'objet de thème du chargeur d'amorçage décrivant le thème utilisé. Si aucun thème n'est fourni, certains chargeurs d'amorçage peuvent utiliser un thème par défaut, c'est le cas de GRUB.

`terminal-outputs` (par défaut : `'(gfxterm)`)

Les terminaux de sortie utilisés par le menu de démarrage du chargeur d'amorçage, en tant que liste de symboles. GRUB accepte les valeurs `console`, `serial`, `serial_{0-3}`, `gfxterm`, `vga_text`, `mda_text`, `morse` et `pkmodem`. Ce champ correspond à la variable GRUB `GRUB_TERMINAL_OUTPUT` (voir Section "Simple configuration" dans *GNU GRUB manual*).

`terminal-inputs` (par défaut : `'()`)

Les terminaux d'entrée utilisés par le menu de démarrage du chargeur d'amorçage, en tant que liste de symboles. Pour GRUB, la valeur par défaut est le terminal natif de la plate-forme déterminé à l'exécution. GRUB accepte les valeurs `console`, `serial`, `serial_{0-3}`, `at_keyboard` et `usb_keyboard`. Ce champ correspond à la variable GRUB `GRUB_TERMINAL_INPUT` (voir Section "Simple configuration" dans *GNU GRUB manual*).

`serial-unit` (par défaut : `#f`)

L'unité série utilisée par le chargeur d'amorçage, en tant qu'entier entre 0 et 3. Pour GRUB, il est choisi à l'exécution ; actuellement GRUB choisi 0, ce qui correspond à COM1 (voir Section "Serial terminal" dans *GNU GRUB manual*).

`serial-speed` (par défaut : `#f`)

La vitesse de l'interface série, en tant qu'entier. Pour GRUB, la valeur par défaut est choisie à l'exécution ; actuellement GRUB choisi 9600 bps (voir Section "Serial terminal" dans *GNU GRUB manual*).

`device-tree-support?` (par défaut : `#t`)

Indique s'il faut prendre en charge le chargement des fichiers `device tree` (<https://en.wikipedia.org/wiki/Devicetree>) de Linux.

Cette option est activée par défaut. Dans certains cas qui impliquent le chargeur d'amorçage `u-boot`, où l'arborescence de l'appareil est déjà chargée en RAM, il peut être pratique de désactiver cette option en indiquant `#f`.

Si vous voulez lister des entrées du menu de démarrage supplémentaires via le champ `menu-entries` ci-dessus, vous devrez les créer avec la forme `menu-entry`. Par exemple, imaginons que vous souhaitiez pouvoir démarrer sur une autre distro (c'est difficile à concevoir !), vous pourriez alors définir une entrée du menu comme ceci :

```
(menu-entry
  (label "L'autre distro")
  (linux "/boot/old/vmlinuz-2.6.32"))
```



```
(linux-arguments ("root=/dev/sda2"))
(initrd "/boot/old/initrd"))
```

Les détails suivent.

**menu-entry** [Type de données]

Le type d'une entrée dans le menu du chargeur d'amorçage.

**label** L'étiquette à montrer dans le menu — p. ex. "GNU".

**linux** (par défaut : #f)

L'image du noyau Linux à démarrer, par exemple :

```
(file-append linux-libre "/bzImage")
```

Pour GRUB, il est aussi possible de spécifier un périphérique explicitement dans le chemin de fichier avec la convention de nommage de GRUB (voir Section "Naming convention" dans *GNU GRUB manual*), par exemple :

```
"(hd0,msdos1)/boot/vmlinuz"
```

Si le périphérique est spécifié explicitement comme au-dessus, le champ **device** est complètement ignoré.

**linux-arguments** (par défaut : ())

La liste des arguments de la ligne de commande du noyau supplémentaires — p. ex. ("console=ttyS0").

**initrd** (par défaut : #f)

Une G-expression ou une chaîne dénotant le nom de fichier du disque de RAM initial à utiliser (voir Section 9.12 [G-Expressions], page 166).

**device** (par défaut : #f)

Le périphérique où le noyau et l'initrd se trouvent — c.-à-d. pour GRUB, l'option *root* de cette entrée de menu (voir Section "root" dans *GNU GRUB manual*).

Cela peut être une étiquette de système de fichiers (une chaîne), un UUID de système de fichiers (un vecteur d'octets, voir Section 12.3 [Systèmes de fichiers], page 250) ou #f, auquel cas le chargeur d'amorçage recherchera le périphérique contenant le fichier spécifié par le champ **linux** (voir Section "search" dans *GNU GRUB manual*). Cela ne doit *pas* être un nom de périphérique donné par l'OS comme */dev/sda1*.

**multiboot-kernel** (par défaut : #f)

Le noyau à démarrer en mode Multiboot (voir Section "multiboot" dans *GNU GRUB manual*). Lorsque ce champ est spécifié, une entrée de menu Multiboot est générée. Par exemple :

```
(file-append mach "/boot/gnumach")
```

**multiboot-arguments** (par défaut : ())

Liste d'arguments de la ligne de commande supplémentaires pour le multiboot-kernel.

**multiboot-modules** (par défaut : ())

Liste de commandes pour charger les modules Multiboot. Par exemple :

```
(list (list (file-append hurd "/hurd/ext2fs.static") "ext2fs"
          ...)
      (list (file-append libc "/lib/ld.so.1") "exec"
          ...))
```

**chain-loader** (par défaut : #f)

Une chaîne qui peut être acceptée par la directive `chainloader` de `grub`. Cela n'a aucun effet si le champ `linux` ou `multiboot-kernel` est renseigné. Voici un exemple de chargement chaîné d'un autre système GNU/Linux.

```
(bootloader
 (bootloader-configuration
 ;; ...
 (menu-entries
 (list
 (menu-entry
 (label "GNU/Linux")
 (device (uuid "1C31-A17C" 'fat))
 (chain-loader "/EFI/GNULinux/grubx64.efi"))))))))
```

Pour l'instant seul GRUB prend en charge les thèmes. On crée un thème GRUB avec la forme `grub-theme`, qui n'est pas encore complètement documentée.

**grub-theme** [Type de données]

Type de données représentant la configuration du thème GRUB.

**gfxmode** (par défaut : ("auto"))

Le `gfxmode` de GRUB à utiliser (une liste de chaînes de résolution d'écran, voir Section “`gfxmode`” dans *GNU GRUB manual*).

**grub-theme** [Procédure Scheme]

Renvoie le thème par défaut de GRUB utilisé par le système d'exploitation si aucun champ `theme` n'est spécifié dans l'enregistrement `bootloader-configuration`.

Il contient une image de fond sympathique avec les logos de GNU et de Guix.

Par exemple, pour changer la résolution par défaut, vous pouvez utiliser quelque chose comme

```
(bootloader
 (bootloader-configuration
 ;; ...
 (theme (grub-theme
 (inherit (grub-theme))
 (gfxmode '("1024x786x32" "auto"))))))))
```

## 12.15 Invoquer guix system

Une fois que vous avez écrit une déclaration de système d'exploitation comme nous l'avons vu dans les sections précédentes, elle peut être instanciée avec la commande `guix system`. Voici le résumé de la commande :

```
guix system options... action file
```

`file` doit être le nom d'un fichier contenant une déclaration `operating-system`. `action` spécifie comment le système d'exploitation est instancié. Actuellement les valeurs suivantes sont supportées :

**search** Affiche les définitions des types de services disponibles qui correspondent aux expressions régulières données, triées par pertinence :

```
$ 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

...
```

Comme pour `guix package --search`, le résultat est écrit au format `recutils`, ce qui rend facile le filtrage de la sortie (voir *GNU recutils manual*).

**edit** Modifier ou visualiser la définition d'un type de service donné.

Par exemple, la commande plus bas ouvre votre éditeur, spécifié par la variable d'environnement `EDITOR`, sur la définition du type de service `openssh` :

```
guix system edit openssh
```

### reconfigure

Construit le système d'exploitation décrit dans *file*, l'active et passe dessus<sup>9</sup>.

**Remarque:** Il est grandement recommandé de lancer `guix pull` une fois avant de lancer `guix system reconfigure` pour la première fois (voir Section 6.6 [Invoquer `guix pull`], page 63). Sans cela, vous verriez une version plus ancienne de Guix une fois `reconfigure` terminé.

Cela met en application toute la configuration spécifiée dans *file* : les comptes utilisateurs, les services du système, la liste globale des paquets, les programmes `setuid`, etc. La commande démarre les services systèmes spécifiés dans *file* qui ne sont pas actuellement lancés ; si un service est actuellement exécuté cette commande s'arrange pour qu'il soit mis à jour la prochaine fois qu'il est stoppé (p. ex par `herd stop X` ou `herd restart X`).

Cette commande crée une nouvelle génération dont le numéro est un de plus que la génération actuelle (rapportée par `guix system list-generations`). Si cette génération existe déjà, elle sera réécrite. Ce comportement correspond à celui de `guix package` (voir Section 6.2 [Invoquer `guix package`], page 44).

Elle ajoute aussi une entrée de menu du chargeur d'amorçage pour la nouvelle configuration, à moins que `--no-bootloader` ne soit passé. Pour GRUB, elle déplace les entrées pour les anciennes configurations dans un sous-menu, ce qui vous permet de choisir une ancienne génération au démarrage si vous en avez besoin.

À la fin, le nouveau système d'exploitation est déployé dans `/run/current-system`. Ce répertoire contient les *métadonnées de provenance* : la liste des canaux utilisés (voir Chapitre 7 [Canaux], page 75) et *fichier* lui-même, s'il est disponible. Vous pouvez les voir avec :

```
guix system describe
```

Cette information est utile si vous voulez plus tard inspecter comment une génération particulière a été construite. En fait, en supposant que *file* est auto-contenu, vous pouvez reconstruire la génération *n* de votre système d'exploitation avec :

```
guix time-machine \
  -C /var/guix/profiles/system-n-link/channels.scm -- \
  system reconfigure \
  /var/guix/profiles/system-n-link/configuration.scm
```

Vous pouvez considérer cela comme une sorte de contrôle de version intégré ! Votre système n'est pas seulement un artéfact binaire : *il contient ses propres sources*. Voir Section 12.18.3 [Référence de service], page 602, pour plus d'informations sur le suivi de provenance.

<sup>9</sup> Cette action (et les action liées que sont `switch-generation` et `roll-back`) ne sont utilisables que sur les systèmes sous Guix System.

Par défaut, **reconfigure** vous empêche de faire revenir en arrière votre système, ce qui pourrait (ré)introduire des vulnérabilités de sécurité et cause des problèmes avec les services « avec état » comme les systèmes de gestion de bases de données. Vous pouvez contrôler ce comportement en passant `--allow-downgrades`.

#### **switch-generation**

Passer à une génération existante du système. Cette action change automatiquement le profil système vers la génération spécifiée. Elle réarrange aussi les entrées existantes du menu du chargeur d'amorçage du système. Elle fait de l'entrée du menu pour la génération spécifiée l'entrée par défaut et déplace les entrées pour les autres générations dans un sous-menu, si cela est pris en charge par le chargeur d'amorçage utilisé. Lors du prochain démarrage du système, la génération du système spécifiée sera utilisée.

Le chargeur d'amorçage lui-même n'est pas réinstallé avec cette commande. Ainsi, le chargeur d'amorçage est utilisé avec un fichier de configuration plus à jour.

La génération cible peut être spécifiée explicitement par son numéro de génération. Par exemple, l'invocation suivante passerait à la génération 7 du système :

```
guix system switch-generation 7
```

La génération cible peut aussi être spécifiée relativement à la génération actuelle avec la forme `+N` ou `-N`, où `+3` signifie « trois générations après la génération actuelle » et `-1` signifie « une génération précédent la génération actuelle ». Lorsque vous spécifiez un nombre négatif comme `-1`, il doit être précédé de `--` pour éviter qu'il ne soit compris comme une option. Par exemple :

```
guix system switch-generation -- -1
```

Actuellement, l'effet de l'invocation de cette action est *uniquement* de passer au profil du système vers une autre génération existante et de réarranger le menu du chargeur d'amorçage. Pour vraiment commencer à utiliser la génération spécifiée, vous devez redémarrer après avoir lancé cette action. Dans le futur, elle sera corrigée pour faire la même chose que **reconfigure**, comme réactiver et désactiver les services.

Cette action échouera si la génération spécifiée n'existe pas.

#### **roll-back**

Passer à la génération précédente du système. Au prochain démarrage, la génération précédente sera utilisée. C'est le contraire de **reconfigure**, et c'est exactement comme invoquer **switch-generation** avec pour argument `-1`.

Actuellement, comme pour **switch-generation**, vous devez redémarrer après avoir lancé cette action pour vraiment démarrer sur la génération précédente du système.

#### **delete-generations**

Supprimer des générations du système, ce qui les rend disponibles pour le ramasse-miettes (voir Section 6.5 [Invoquer guix gc], page 60, pour des informations sur la manière de lancer le « ramasse-miettes »).

Cela fonctionne comme pour ‘`guix package --delete-generations`’ (voir Section 6.2 [Invoquer `guix package`], page 44). Avec aucun argument, toutes les générations du système sauf la génération actuelle sont supprimées :

```
guix system delete-generations
```

Vous pouvez aussi choisir les générations que vous voulez supprimer. L’exemple plus bas supprime toutes les génération du système plus vieilles que deux mois :

```
guix system delete-generations 2m
```

Lancer cette commande réinstalle automatiquement le chargeur d’amorçage avec une liste à jour d’entrées de menu — p. ex. le sous-menu « anciennes générations » dans GRUB ne liste plus les générations qui ont été supprimées.

**build** Construit la dérivation du système d’exploitation, ce qui comprend tous les fichiers de configuration et les programmes requis pour démarrer et lancer le système. Cette action n’installe rien.

**init** Rempli le répertoire donné avec tous les fichiers nécessaires à lancer le système d’exploitation spécifié dans *file*. C’est utile pour la première installation de Guix System. Par exemple :

```
guix system init my-os-config.scm /mnt
```

copie tous les éléments du dépôt requis par la configuration spécifiée dans `my-os-config.scm` dans `/mnt`. Cela comprend les fichiers de configuration, les paquets, etc. Elle crée aussi d’autres fichiers essentiels requis pour que le système fonctionne correctement — p. ex. les répertoires `/etc`, `/var` et `/run` et le fichier `/bin/sh`.

Cette commande installe aussi le chargeur d’amorçage sur les cibles spécifiées dans `my-os-config`, à moins que l’option `--no-bootloader` ne soit passée.

**vm** Construit une machine virtuelle (VM) qui contient le système d’exploitation déclaré dans *file* et renvoie un script pour lancer cette VM.

**Remarque:** Les actions `vm` et les autres plus bas peuvent utiliser la prise en charge KVM du noyau Linux-libre. Plus spécifiquement, si la machine prend en charge la virtualisation matérielle, le module noyau KVM correspondant devrait être chargé, et le nœud de périphérique `/dev/kvm` devrait exister et être lisible et inscriptible pour l’utilisateur et pour les utilisateurs de construction du démon (voir Section 2.4.1 [Réglages de l’environnement de construction], page 11).

Les arguments passés au script sont passés à QEMU comme dans l’exemple ci-dessous, qui active le réseau et demande 1 Go de RAM pour la machine émulée :

```
$ /gnu/store/...-run-vm.sh -m 1024 -smp 2 -nic user,model=virtio-net-pci
```

Il est possible de combiner les deux étapes en une :

```
$ $(guix system vm my-config.scm) -m 1024 -smp 2 -nic user,model=virtio-net-
```

La VM partage sont dépôt avec le système hôte.

Par défaut, le système de fichiers racine de la VM sera monté de manière volatile ; l'option `--persistent` peut être fournie pour le rendre persistant à la place. Dans ce cas, le fichier d'image disque VM sera copiée à partir du dépôt vers le répertoire `TMPTDIR` pour le rendre inscriptible.

Vous pouvez partager des fichiers supplémentaires entre l'hôte et la VM avec les options en ligne de commande `--share` et `--expose` : la première spécifie un répertoire à partager avec accès en écriture, tandis que le deuxième fournit un accès en lecture-seule au répertoire partagé.

L'exemple ci-dessous crée une VM dans laquelle le répertoire personnel de l'utilisateur est accessible en lecture-seule, et où le répertoire `/exchange` est une correspondance en lecture-écriture à `$HOME/tmp` sur l'hôte :

```
guix system vm my-config.scm \
  --expose=$HOME --share=$HOME/tmp=/exchange
```

Sur GNU/Linux, le comportement par défaut consiste à démarrer directement sur le noyau ; cela a l'avantage de n'avoir besoin que d'une toute petite image disque puisque le dépôt de l'hôte peut ensuite être monté.

L'option `--full-boot` force une séquence de démarrage complète, en commençant par le chargeur d'amorçage. Cela requiert plus d'espace disque puisqu'une image racine contenant au moins le noyau, l'`initrd` et les fichiers de données du chargeur d'amorçage doit être créé.

On peut utiliser l'option `--image-size` pour spécifier la taille de l'image.

L'option `--no-graphic` demandera à `guix system` de créer une VM sans interface qui utilisera le `tty` qui l'invoque pour ses entrées-sorties. Entre autres, cela active le copier-coller et le défilement. Utilisez le préfixe `ctrl-a` pour lancer des commandes QEMU ; p. ex. `ctrl-a h` affiche l'aide, `ctrl-a x` quitte la VM et `ctrl-a c` alterne entre le moniteur QEMU et la VM.

**image** La commande `image` peut produire divers types d'images. Vous pouvez choisir le type d'image en utilisant l'option `--image-type`. Sa valeur par défaut est `efi-raw`. Lorsque sa valeur est `iso9660`, vous pouvez utiliser l'option `--label` pour spécifier un identifiant de volume avec `image`. Par défaut, le système de fichiers racine d'une image disque est montée en non-volatile ; vous pouvez fournir l'option `--volatile` pour la rendre volatile. Si vous utilisez `image`, le chargeur d'amorçage installé sur l'image générée est celui de la définition `operating-system` fournie. L'exemple suivant montre comment générer une image qui utilise le chargeur d'amorçage `grub-efi-bootloader` et la démarrer avec QEMU :

```
image=$(guix system image --image-type=qcow2 \
  gnu/system/examples/lightweight-desktop.tpl)
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
```

Lorsque vous utilisez le type d'image `efi-raw`, une image disque brute est produite ; elle peut être copiée telle quelle sur un périphérique USB par exemple.

En supposant que `/dev/sdc` est le périphérique correspondant à une clé USB, on peut copier l'image dessus avec la commande suivante :

```
# dd if=$(guix system image my-os.scm) of=/dev/sdc status=progress
```

La commande `--list-image-types` liste tous les types d'images disponibles.

Lorsque vous utilisez le type d'image `qcow2`, l'image renvoyée est au format `qcow2`, que l'émulateur QEMU peut utiliser efficacement. Voir Section 12.17 [Lancer Guix dans une VM], page 597, pour plus d'information sur la manière de lancer l'image dans une machine virtuelle. Le chargeur d'amorçage `grub-bootloader` est toujours utilisé quelque soit celui déclaré dans le fichier `operating-system` passé en argument. Cela facilite l'utilisation de QEMU, qui utilise le BIOS SeaBIOS par défaut, et s'attend à un chargeur d'amorçage installé dans le Master Boot Record (MBR).

En utilisant le type d'image `docker`, on produit une image Docker. Guix construit l'image de zéro, et non à partir d'une image Docker de base pré-existante. En conséquence, elle contient *exactly* ce que vous avez défini dans le fichier de configuration du système. Vous pouvez ensuite charger l'image et lancer un conteneur Docker avec des commande comme :

```
image_id="$(docker load < guix-system-docker-image.tar.gz)"
container_id="$(docker create $image_id)"
docker start $container_id
```

Cette commande démarre un nouveau conteneur Docker à partir de l'image spécifiée. Il démarrera le système Guix de la manière habituelle, ce qui signifie qu'il démarrera tous les services que vous avez définis dans la configuration du système d'exploitation. Vous pouvez obtenir un shell interactif dans le conteneur en utilisant `docker exec` :

```
docker exec -ti $container_id /run/current-system/profile/bin/bash --login
```

En fonction de ce que vous lancez dans le conteneur Docker, il peut être nécessaire de donner des permissions supplémentaires au conteneur. Par exemple, si vous voulez construire des paquets avec Guix dans le conteneur Docker, vous devriez passer `--privileged` à `docker create`.

Enfin, l'option `--network` s'applique à `guix system docker-image` : elle produit une image où le réseau est partagé avec l'hôte, et donc sans services comme `nscd` ou `NetworkManager`.

### conteneur

Renvoie un script qui lance le système d'exploitation déclaré dans *file* dans un conteneur. Les conteneurs sont un ensemble de mécanismes d'isolation légers fournis par le noyau Linux-libre. Les conteneurs sont substantiellement moins gourmands en ressources que les machines virtuelles complètes car le noyau, les objets partagés et d'autres ressources peuvent être partagés avec le système hôte ; cela signifie aussi une isolation moins complète.

Actuellement, le script doit être lancé en root pour pouvoir supporter plus d'un utilisateur et d'un groupe. Le conteneur partage son dépôt avec le système hôte.

Comme avec l'action `vm` (voir [guix system vm], page 587), des systèmes de fichiers supplémentaires peuvent être partagés entre l'hôte et le conteneur avec les options `--share` et `--expose` :



```
guix system container my-config.scm \
  --expose=$HOME --share=$HOME/tmp=/exchange
```

Les options `--share` et `--expose` peuvent aussi être passées au script généré pour effectuer un montage lié de répertoires supplémentaires dans le conteneur.

**Remarque:** Cette option requiert Linux-libre ou supérieur.

`options` peut contenir n'importe quelle option commune de construction (voir Section 10.1.1 [Options de construction communes], page 180). En plus, `options` peut contenir l'une de ces options :

`--expression=expr`

`-e expr` Considère le système d'exploitation en lequel s'évalue `expr`. C'est une alternative à la spécification d'un fichier qui s'évalue en un système d'exploitation. C'est utilisé pour générer l'installateur du système Guix (voir Section 3.9 [Construire l'image d'installation], page 36).

`--system=système`

`-s système`

Essaye de construire pour `system` au lieu du type du système hôte. Cela fonctionne comme pour `guix build` (voir Section 10.1 [Invoquer guix build], page 180).

`--target=triplet`

Effectuer une compilation croisée pour `triplet` qui doit être un triplet GNU valide, comme `"aarch64-linux-gnu"` (voir Section "Specifying target triplets" dans *Autoconf*).

`--derivation`

`-d` Renvoie le nom du fichier de dérivation du système d'exploitation donné sans rien construire.

`--save-provenance`

Comme nous en avons parlé précédemment, `guix system init` et `guix system reconfigure` enregistrent toujours les informations de provenance via un service dédié (voir Section 12.18.3 [Référence de service], page 602). Cependant, d'autres commandes ne le font pas par défaut. Si vous voulez, disons, créer une image de machine virtuelle contenant les informations de provenance, vous pouvez lancer :

```
guix system image -t qcow2 --save-provenance config.scm
```

De cette façon, l'image qui en résulte va effectivement « intégrer sa propre source » sous la forme de méta-données dans `/run/current-system`. Avec cette information, on peut reconstruire l'image pour s'assurer qu'elle contient vraiment ce qu'elle prétend contenir ; ou on peut utiliser ça pour dériver une variante de l'image.

`--image-type=type`

`-t type` Pour l'action `image`, crée une image du `type` donné.

Lorsque cette option est omise, `guix system` utilise le type d'image `efi-raw`.

`--image-type=iso9660` produit une image ISO-9660, qu'il est possible de graver sur un CD ou un DVD.

- image-size=size**  
 Pour l'action `image`, crée une image de la taille donnée *size*. *size* peut être un nombre d'octets ou contenir un suffixe d'unité (voir Section "Block size" dans *GNU Coreutils*).  
 Lorsque cette option est omise, `guix system` calcule une estimation de la taille de l'image en fonction de la taille du système déclaré dans *file*.
- network**  
**-N** Pour l'action `container`, permet aux conteneurs d'accéder au réseau de l'hôte, c'est-à-dire, ne crée pas d'espace de nom réseau.
- root=fichier**  
**-r fichier**  
 Fait de *fichier* un lien symbolique vers le résultat, et l'enregistre en tant que racine du ramasse-miettes.
- skip-checks**  
 Passe les vérifications de sécurité avant l'installation.  
 Par défaut, `guix system init` et `guix system reconfigure` effectuent des vérifications de sécurité : ils s'assurent que les systèmes de fichiers qui apparaissent dans la déclaration `operating-system` existent vraiment (voir Section 12.3 [Systèmes de fichiers], page 250) et que les modules de noyau Linux qui peuvent être requis au démarrage sont listés dans `initrd-modules` (voir Section 12.13 [Disque de RAM initial], page 575). Passer cette option saute ces vérifications complètement.
- allow-downgrades**  
 Dit à `guix system reconfigure` de permettre le retour en arrière du système.  
 Par défaut, `reconfigure` vous empêche de faire revenir votre système en arrière. Il fait cela en comparant les informations de provenance de votre système (décrites par `guix system describe`) avec celles de la commande `guix` (décrites par `guix describe`). Si les commits de `guix` ne descendent pas de ceux utilisés pour votre système, `guix system reconfigure` renvoie une erreur. Passer `--allow-downgrades` vous permet de passer ces tests.  
**Remarque:** Assurez-vous de comprendre les implications de sa sécurité avant d'utiliser `--allow-downgrades`.
- on-error=strategy**  
 Applique *strategy* lorsqu'une erreur arrive lors de la lecture de *file*. *strategy* peut être l'une des valeurs suivantes :
- nothing-special**  
 Rapporte l'erreur de manière concise et quitte. C'est la stratégie par défaut.
- backtrace**  
 Pareil, mais affiche aussi une trace de débogage.
- debug**  
 Rapporte l'erreur et entre dans le débogueur Guile. À partir de là, vous pouvez lancer des commandes comme `,bt` pour obtenir une

trace de débogage, `,locals` pour afficher les valeurs des variables locales et plus généralement inspecter l'état du programme. Voir Section “Debug Commands” dans *GNU Guile Reference Manual*, pour une liste de commandes de débogage disponibles.

Une fois que vous avez construit, re-configuré et re-re-configuré votre installation Guix, vous pourriez trouver utile de lister les générations du système disponibles sur le disque — et que vous pouvez choisir dans le menu du chargeur d'amorçage :

**describe** Décrit la génération du système lancé : son nom de fichier, son noyau et chargeur d'amorçage utilisé, etc, ainsi que les informations de provenance si elles sont disponibles.

le drapeau `--list-installed` est disponible, avec la même syntaxe que celui utilisé dans `guix package --list-installed` (voir Section 6.2 [Invoquer guix package], page 44). Lorsque le drapeau est utilisé, la description inclura une liste des paquets qui sont actuellement installés dans le profil du système, avec un filtre facultatif sur une expression régulière.

**Remarque:** La génération du système *en cours* — référencé par `/run/current-system` — n'est pas nécessairement la génération *actuelle* du système — référencée par `/var/guix/profiles/system` : elles sont différentes si par exemple, vous choisissez dans le menu du chargeur d'amorçage de démarrer une ancienne génération.

Elles peuvent aussi être différentes de la génération du système *démarrée* — référencée par `/run/booted-system` — par exemple parce que vous avez reconfiguré le système entre temps.

#### list-generations

Affiche un résumé de chaque génération du système d'exploitation disponible sur le disque, dans un format lisible pour un humain. C'est similaire à l'option `--list-generations` de `guix package` (voir Section 6.2 [Invoquer guix package], page 44).

Éventuellement, on peut spécifier un motif, avec la même syntaxe utilisée pour `guix package --list-generations`, pour restreindre la liste des générations affichées. Par exemple, la commande suivante affiche les générations de moins de 10 jours :

```
$ guix system list-generations 10d
```

Le drapeau `--list-installed` peut aussi être spécifié, avec la même syntaxe que celle utilisée dans `guix package --list-installed`. Cela peut être utile si vous essayez de déterminer si un paquet a été ajouté au système.

La commande `guix system` a même plus à proposer ! Les sous-commandes suivantes vous permettent de visualiser comme vos services systèmes sont liés les uns aux autres :

#### extension-graph

Emettre vers la sortie standard le *graphe d'extension de service* du système d'exploitation défini dans *file* (voir Section 12.18.1 [Composition de services], page 599, pour plus d'informations sur les extensions de service). Par défaut, le

format de sortie est le format Dot/Graphviz, mais vous pouvez choisir un autre format avec `--graph-backend`, comme avec `guix graph` (voir Section 10.10 [Invoquer `guix graph`], page 218) :

La commande :

```
$ guix system extension-graph file | xdot -
```

montre les relations d'extension entre les services.

**Remarque:** Le programme `dot` est fournit par le paquet `graphviz`.

### shepherd-graph

Affiche le *graphe de dépendance* des services shepherd du système d'exploitation défini dans *file* sur la sortie standard. Voir Section 12.18.4 [Services Shepherd], page 607, pour plus d'informations et un exemple de graphe.

Encore une fois, le format de sortie par défaut est Dot/Graphviz, mais vous pouvez passer `--graph-backend` pour en choisir un autre.

## 12.16 Invoquer `guix deploy`

Nous avons déjà vu les déclarations de `operating-system` utilisées pour gérer la configuration d'une machine localement. Supposez que vous ayez besoin de configurer plusieurs machines — peut-être que vous gérez un service web comprenant plusieurs serveurs. `guix deploy` vous permet d'utiliser les mêmes déclarations de systèmes d'exploitation pour gérer plusieurs hôtes distants en même temps dans le même « déploiement » logique.

**Remarque:** La fonctionnalité décrite dans cette section est toujours en cours de développement et est sujette à changement. Contactez-nous sur [guix-devel@gnu.org](mailto:guix-devel@gnu.org) !

```
guix deploy fichier
```

Une telle invocation déploiera les machines en lesquelles le code de *fichier* s'évalue. Par exemple, *fichier* peut contenir une définition comme celle-ci :

```
;; Ceci est le déploiement Guix d'un système « minimal » sans
;; serveur d'affichage X11, vers une machine dont le démon SSH
;; écoute sur localhost:2222. Une configuration comme celle-ci
;; est appropriée pour les machines virtuelles dont les ports sont relayés
;; vers l'interface de bouclage de l'hôte.
```

```
(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)
                  (target ("/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))))))

```

Le fichier devrait s'évaluer en une liste d'objets *machine*. Cet exemple, lors du déploiement, créera une nouvelle génération sur le système distant en réalisant la déclaration `operating-system %system`. `environment` et `configuration` spécifient comme la machine devrait être provisionnée — c.-à-d. comment les ressources sont créées et gérées. L'exemple ci-dessus ne crée aucune ressource, car un `'managed-host` est une machine qui fait déjà tourner le système Guix et est disponible sur le réseau. Cela est un cas particulièrement simple ; un déploiement plus complexe pourrait impliquer, par exemple, le démarrage de machines virtuels chez un fournisseur de serveurs privés virtuels (VPS). Dans ce cas, une type d'*environnement* différent devrait être utilisé.

Prenez note que vous devez d'abord générer une paire de clés sur la machine coordinatrice pour permettre au démon d'exporter les archives signées des fichiers du dépôt (voir Section 6.10 [Invoquer guix archive], page 71), bien que cette étape soit automatique sur Guix System :

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

Chaque machine cible doit autoriser la clé de la machine maître afin qu'elle accepte les objets de stockage qu'elle reçoit du coordinateur :

```
# guix archive --authorize < coordinator-public-key.txt
```

`user`, dans cet exemple, spécifie le nom du compte utilisateur à utiliser pour se connecter lors du déploiement. Sa valeur par défaut est `root`, mais la connexion SSH en `root` peut être interdite dans certains cas. Pour contourner cela, `guix deploy` peut se connecter en tant qu'utilisateur non-privilegié et utiliser `sudo` pour récupérer les privilèges. Cela ne fonctionnera que si `sudo` est actuellement installé sur la machine distante et peut être invoqué de manière non interactive par `user`. C'est-à-dire que la ligne `sudoers` permettant à `user` d'utiliser `sudo` doit contenir l'attribut `NOPASSWD`. Cela peut se faire avec le bout de configuration du système d'exploitation suivant :

```

(use-modules ...
  (gnu system))
;pour %sudoers-specification

(define %user "username")

(operating-system
  ...
  (sudoers-file
    (plain-file "sudoers"
      (string-append (plain-file-content %sudoers-specification)
        (format #f "~a ALL = NOPASSWD: ALL~%"
          %user))))))

```

Pour plus d'informations sur le format du fichier `sudoers`, consultez `man sudoers`.

Une fois que vous avez déployé un système sur un ensemble de machines, vous trouverez utile de lancer une commande sur chacune. Les options `--execute` ou `-x` vous le permet. L'exemple ci-dessous lance `uname -a` sur toutes les machines listées dans le fichier de déploiement :

```
guix deploy fichier -x -- uname -a
```

Ce que vous devrez souvent faire après un déploiement est de redémarrer les services spécifiques sur toutes les machines, ce qui peut se faire de cette manière :

```
guix deploy file -x -- herd restart service
```

La commande `guix deploy -x` renvoie zéro si et seulement si la commande réussit sur toutes les machines.

Voici les types de données que vous devrez connaître pour écrire un fichier de déploiement.

**machine** [Type de données]

C'est le type de données représentant une seule machine dans un déploiement Guix hétérogène.

**operating-system**

L'objet de la configuration du système d'exploitation à déployer.

**environment**

Un `environment-type` décrit comment la machine est provisionnée.

**configuration** (par défaut : `#f`)

Un objet décrivant la configuration de l'environnement de la machine. Si l'objet `environment` a une configuration par défaut, vous pouvez utiliser `#f`. Si vous utilisez `#f` mais que l'environnement n'a pas de configuration par défaut, une erreur sera renvoyée.

**machine-ssh-configuration** [Type de données]

C'est le type de données représentant les paramètres du client SSH pour une machine qui a pour `environment managed-host-environment-type`.

- host-name**  
**build-locally?** (par défaut : **#t**)  
Si la valeur est fausse, les dérivations du système seront construites sur la machine qui est déployée.
- system** Le type de système décrivant l'architecture de la machine déployée pour— par exemple, "x86\_64-linux".
- authorize?** (par défaut : **#t**)  
Si la valeur est vraie, la clé de signature du coordinateur sera ajoutée au porteclé des ACL de la machine distante.
- port** (par défaut : 22)  
**user** (par défaut : "root")  
**identity** (par défaut : **#f**)  
Si spécifié, le chemin d'accès à la clé privée SSH à utiliser pour s'authentifier auprès de l'hôte distant.
- host-key** (par défaut : **#f**)  
Cela devrait être la clé de l'hôte SSH de la machine, qui ressemble à cela :  

```
ssh-ed25519 AAAAC3Nz... root@example.org
```

  
Quand **host-key** est **#f**, le serveur est authentifié avec le fichier `~/.ssh/known_hosts`, exactement comme le ferait le client `ssh` OpenSSH.
- allow-downgrades?** (par défaut : **#f**)  
Autoriser ou non d'éventuels déclassements.  
  
Comme `guix sysetm reconfigure`, `guix deploy` compare les commits du canal actuellement déployé sur l'hôte distant (renvoyé par `guix system describe`) à ceux utilisés (renvoyés par `guix describe`) pour déterminer si les commits actuellement utilisés descendent de ceux déployés. Lorsque ce n'est pas le cas et que **allow-downgrades?** est faux, cela lève une erreur. Cela permet de s'assurer que vous ne déployez pas accidentellement une version plus ancienne version les machines distantes.
- safety-checks?** (par défaut : **#t**)  
Indique s'il faut lancer des « vérifications de cohérence » avant le déploiement. Cela comprend la vérification que les périphériques et les systèmes de fichiers référencés dans la configuration du système d'exploitation existent vraiment sur la machine cible, et la vérification que les modules Linux nécessaires à l'accès aux périphériques de stockage au démarrage se trouvent dans le champ `initrd-modules` du système d'exploitation.  
  
Ces vérification s'assurent que vous ne déployez pas par inadvertance un système qui ne pourra pas démarrer. Soyez prudent avant de les désactiver !

<b>digital-ocean-configuration</b>	[Type de données] C'est le type de données décrivant le Droplet qui devrait être créé pour une machine avec un <code>environment</code> valant <code>digital-ocean-environment-type</code> .
<b>ssh-key</b>	Le chemin vers la clé SSH privée à utiliser pour s'authentifier avec l'hôte distant. Dans le futur, le champ pourrait ne plus exister.
<b>tags</b>	Une liste « d'attributs » qui identifie la machine de manière unique. Il faut faire attention à ce que deux machines du déploiement n'aient pas le même ensemble d'attributs.
<b>region</b>	Le nom d'une région de Digital Ocean, comme "nyc3".
<b>taille</b>	Le nom d'une taille de Digital Ocean, comme "s-1vcpu-1gb".
<b>enable-ipv6?</b>	Indique s'il faut créer une IPv6 pour le droplet.

## 12.17 Exécuter Guix sur une machine virtuelle

Pour exécuter Guix dans une machine virtuelle (VM), on peut soit utiliser l'image de VM Guix pré-construite sur `https://ftp.gnu.org/gnu/guix/guix-system-vm-image-1.4.0.x86_64-linux.qcow2`. Cette image est une image compressée au format QCOW. Vous devrez la passer à un émulateur comme QEMU (`https://qemu.org/`) (voir plus bas pour des détails).

Cette image démarre l'environnement graphique Xfce et contient certains outils couramment utilisés. Vous pouvez installer plus de logiciels sur l'image en lançant `guix package` dans un terminal (voir Section 6.2 [Invoquer guix package], page 44). Vous pouvez aussi reconfigurer le système à partir de son fichier de configuration initiale disponible dans `/run/current-system/configuration.scm` (voir Section 12.1 [Utiliser le système de configuration], page 238).

Au lieu d'utiliser cette image pré-construite, on peut construire sa propre image avec `guix system image` (voir Section 12.15 [Invoquer guix system], page 584).

Si vous construisez votre propre image, vous devez la copier en dehors du dépôt (voir Section 9.9 [Le dépôt], page 157) et vous donner la permission d'écrire sur la copie avant de pouvoir l'utiliser. Lorsque vous invoquez QEMU, vous devez choisir un émulateur système correspondant à votre plate-forme matérielle. Voici une invocation minimale de QEMU qui démarrera le résultat de `guix system image -t qcow2` sur un matériel x8\_64 :

```
$ 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
```

Voici la signification de ces options :

`qemu-system-x86_64`

Cela spécifie la plate-forme matérielle à émuler. Elle doit correspondre à l'hôte.

`-nic user,model=virtio-net-pci`

Active la pile réseau en mode utilisateur non privilégié. Le système invité peut accéder à l'hôte mais pas l'inverse. C'est la manière la plus simple de mettre



le système en ligne. `model` spécifie le périphérique réseau à émuler : `virtio-net-pci` est un périphérique spécial conçu pour les systèmes d'exploitation virtualisés et recommandés dans la plupart des cas. En supposant que votre plateforme matérielle est `x86_64`, vous pouvez récupérer une liste des modèles d'interfaces réseaux en lançant `qemu-system-x86_64 -nic model=help`.

**-enable-kvm**

Si votre système a des extensions de virtualisation matérielle, activer le support des machines virtuelles de Linux (KVM) accélérera les choses.

**-m 2048** RAM disponible sur l'OS émulé, en mébioctets. La valeur par défaut est 128 Mo, ce qui peut ne pas suffire pour certaines opérations.

**-device virtio-blk,drive=myhd**

Crée un lecteur `virtio-blk` nommé « myhd ». `virtio-blk` est un mécanisme de « paravirtualisation » pour les périphériques blocs qui permet à QEMU d'avoir de meilleures performances que s'il émulait un disque complet. Voir la documentation de QEMU et KVM pour plus d'info.

**-drive if=none,file=/tmp/qemu-image,id=myhd**

Utilise notre image QCOW, le fichier `/tmp/qemu-image`, comme stockage pour le lecteur « myhd ».

Le script `run-vm.sh` par défaut renvoyé par une invocation de `guix system vm` n'ajoute pas le drapeau `-nic user` par défaut. Pour avoir accès au réseau dans la vm, ajoutez le (`dhcp-client-service`) à votre définition et démarrez la VM avec `$(guix system vm config.scm) -nic user`. Un problème important avec `-nic user` pour le réseau, est que `ping` ne fonctionnera pas, car il utilise le protocole ICMP. Vous devrez utiliser une autre commande pour vérifier la connectivité réseau, par exemple `guix download`.

### 12.17.1 Se connecter par SSH

Pour activer SSH dans une VM vous devez ajouter un serveur SSH comme `openssh-service-type` à votre VM (voir Section 12.9.5 [Services réseau], page 299). En plus vous devez transférer le port 22, par défaut, à l'hôte. Vous pouvez faire cela avec

```
$(guix system vm config.scm) -nic user,model=virtio-net-pci,hostfwd=tcp::10022-:22
```

Pour vous connecter à la VM vous pouvez lancer

```
ssh -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no -p 10022 localhost
```

Le `-p` donne le port auquel vous voulez vous connecter à `ssh`, `-o UserKnownHostsFile=/dev/null` évite que `ssh` ne se plaigne à chaque fois que vous modifiez le fichier `config.scm` et `-o StrictHostKeyChecking=no` évite que vous n'ayez à autoriser une connexion à un hôte inconnu à chaque fois que vous vous connectez.

**Remarque:** Si le `'hostfwd'` d'exemple ci-dessus ne marche pas chez vous (ex : si votre client SSH reste bloqué en attente de connexion vers le port lié de votre VM), assurez-vous que votre VM Guix System prend bien en charge le réseau, en utilisant par exemple le type de service `dhcp-client-service-type`.

### 12.17.2 Utiliser virt-viewer avec Spice

Alternativement au client graphique `qemu` par défaut vous pouvez utiliser `remote-viewer` du paquet `virt-viewer`. Pour vous connecter, passez le drapeau `-spice port=5930,disable-`

`ticketing` à `qemu`. Voir les sections précédentes pour plus d'informations sur comment faire cela.

Spice a aussi de chouettes fonctionnalités comme le partage de votre presse-papier avec la VM. Pour activer cela vous devrez aussi passer les drapeaux suivants à `qemu` :

```
-device virtio-serial-pci,id=virtio-serial0,max_ports=16,bus=pci.0,addr=0x5
-chardev spicevmc,name=vdagent,id=vdagent
-device virtserialport,nr=1,bus=virtio-serial0.0,chardev=vdagent,\
name=com.redhat.spice.0
```

Vous devrez également ajouter le (`spice-vdagent-service`) à la définition de votre système (voir Section 12.9.36 [Services divers], page 559).

## 12.18 Définir des services

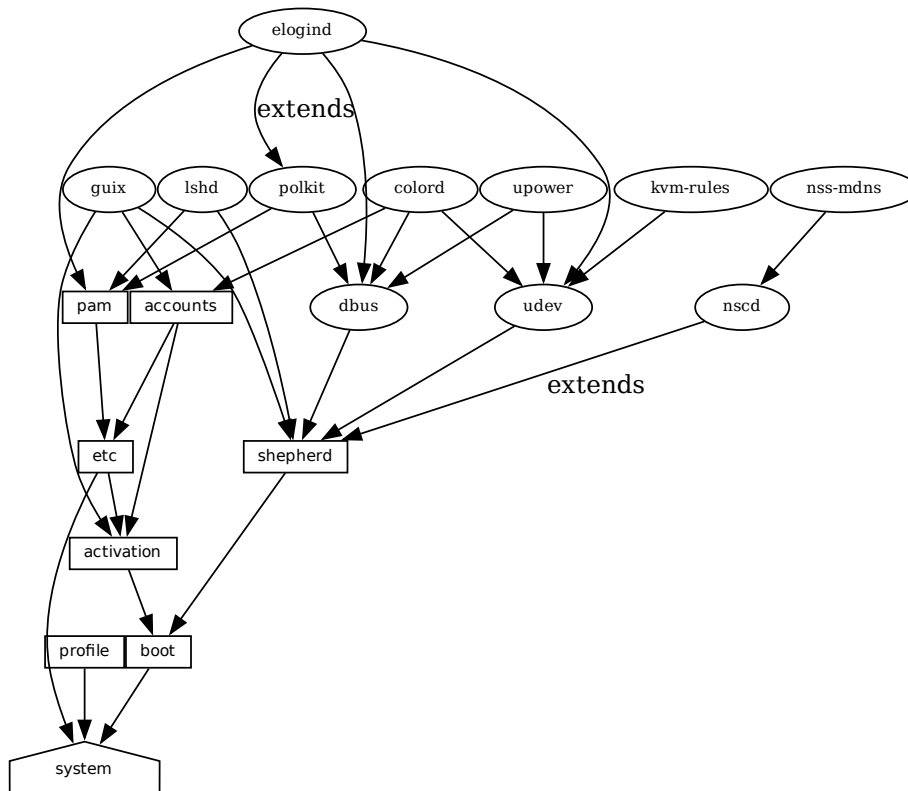
Les sections précédentes montrent les services disponibles et comment on peut les combiner dans une déclaration `operating-system`. Mais, déjà, comment les définir ? Et qu'est-ce qu'un service au fait ?

### 12.18.1 Composition de services

Ici nous définissons un *service* comme étant, assez largement, quelque chose qui étend la fonctionnalité d'un système d'exploitation. Souvent un service est un processus — un *démon* — démarré lorsque le système démarre : un serveur ssh, un serveur web, le démon de construction de Guix, etc. Parfois un service est un démon dont l'exécution peut être déclenchée par un autre démon — p. ex. un serveur FTP démarré par `inetd` ou un service D-Bus activé par `dbus-daemon`. Parfois, un service ne correspond pas à un démon. Par exemple, le service « de comptes » récupère la liste des comptes utilisateurs et s'assure qu'ils existent bien lorsque le système est lancé ; le service « udev » récupère les règles de gestion des périphériques et les rend disponible au démon `eudev` ; le service `/etc` remplit le répertoire `/etc` du système.

Les services de Guix sont connectés par des *extensions*. Par exemple, le service `ssh` étend le Shepherd — le système d'initialisation de GuixSD, qui tourne en tant que PID 1 — en lui donnant les lignes de commande pour démarrer et arrêter le démon `ssh` (voir Section 12.9.5 [Services réseau], page 299) ; le service `UPower` étend le service D-Bus en lui passant sa spécification `.service` et étend le service `udev` en lui passant des règles de gestion de périphériques (voir Section 12.9.9 [Services de bureaux], page 347) ; le démon Guix étend le Shepherd en lui passant les lignes de commande pour démarrer et arrêter le démon et étend le service de comptes en lui passant une liste des comptes utilisateurs de constructions requis (voir Section 12.9.1 [Services de base], page 268).

En définitive, les services et leurs relation « d'extensions » forment un graphe orienté acyclique (DAG). Si nous représentons les services comme des boîtes et les extensions comme des flèches, un système typique pourrait fournir quelque chose comme cela :



En bas, on voit le *service système* qui produit le répertoire contenant tout et lançant et démarrant le système, renvoyé par la commande `guix system build`. Voir Section 12.18.3 [Référence de service], page 602, pour apprendre les autres types de services montrés ici. Voir [system-extension-graph], page 592, pour plus d'informations sur la manière de générer cette représentation pour une définition de système d'exploitation particulière.

Techniquement, on peut définir des *types de services* pour exprimer ces relations. Il peut y avoir n'importe quel quantité de services d'un type donné sur le système — par exemple, un système sur lequel tournent deux instances du serveur ssh de GNU (`lsh`) a deux instance de *lsh-service-type*, avec des paramètres différents.

La section suivante décrit l'interface de programmation des types de services et des services.

## 12.18.2 Types service et services

Un *type de service* est un nœud dans le DAG décrit plus haut. Commençons avec un exemple simple, le type de service pour le démon de construction de Guix (voir Section 2.5 [Invoquer `guix-daemon`], page 17) :

```
(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)))
```

Il définit trois choses :

1. Un nom, dont le seul but de rendre l'inspection et le débogage plus faciles.
2. Une liste d'*extensions de services*, où chaque extension désigne le type de service cible et une procédure qui, étant donné les paramètres du service, renvoie une liste d'objets pour étendre le service de ce type.

Chaque type de service a au moins une extension de service. La seule exception est le *type de service boot*, qui est le service ultime.

3. Éventuellement, une valeur par défaut pour les instances de ce type.

Dans cet exemple, *guix-service-type* étend trois services :

#### **shepherd-root-service-type**

La procédure *guix-shepherd-service* définit comment le service du Shepherd est étendu. En fait, elle renvoie un objet `<shepherd-service>` qui définit comment *guix-daemon* est démarré et stoppé (voir Section 12.18.4 [Services Shepherd], page 607).

#### **account-service-type**

Cette extension pour ce service est calculée par *guix-accounts*, qui renvoie une liste d'objets `user-group` et `user-account` représentant les comptes des utilisateurs de construction (voir Section 2.5 [Invoquer *guix-daemon*], page 17).

#### **activation-service-type**

Ici, *guix-activation* est une procédure qui renvoie une *gexp*, qui est un bout de code qui s'exécute au moment de l'activation — p. ex. lorsque le service est démarré.

Un service de ce type est instancié de cette manière :

```
(service guix-service-type
 (guix-configuration
  (build-accounts 5)
  (extra-options '("--gc-keep-derivations"))))
```

Le deuxième argument de la forme `service` est une valeur représentant les paramètres de cet instance spécifique du service. Voir [guix-configuration-type], page 276, pour plus d'informations sur le type de données `guix-configuration`. Lorsque la valeur est omise, la valeur par défaut spécifiée par `guix-service-type` est utilisée :

```
(service guix-service-type)
```

*guix-service-type* est très simple car il étend d'autres services mais ne peut pas être étendu.

Le type de service pour un service *extensible* ressemble à ceci :

```
(define udev-service-type
```

```
(service-type (name 'udev)
  (extensions
    (list (service-extension shepherd-root-service-type
          udev-shepherd-service)))

  (compose concatenate) ; concatène la liste des règles
  (extend (lambda (config rules)
    (match config
      (($ <udev-configuration> udev initial-rules)
        (udev-configuration
          (udev udev) ; le paquet udev à utiliser
          (rules (append initial-rules rules))))))))))
```

C'est le type de service pour le démon de gestion des périphériques `udev` (<https://wiki.gentoo.org/wiki/Project:Eudev>). Comparé à l'exemple précédent, en plus d'une extension de `shepherd-root-service-type`, on trouve deux nouveaux champs :

- compose** C'est la procédure pour *composer* la liste des extensions de services de ce type. Les services peuvent étendre le service `udev` en lui passant des listes de règles ; on compose ces extensions simplement en les concaténant.
- extend** Cette procédure définie comme la valeur du service est *étendue* avec la composition des extensions. Les extensions Udev sont composés en une liste de règles, mais la valeur du service `udev` est elle-même un enregistrement `<udev-configuration>`. Donc ici, nous étendons cet enregistrement en ajoutant la liste des règles contribuées à la liste des règles qu'il contient déjà.

#### description

C'est une chaîne donnant un aperçu du type de service. Elle peut contenir du balisage Texinfo (voir Section "Overview" dans *GNU Texinfo*). La commande `guix system search` permet de rechercher dans ces chaînes et de les afficher (voir Section 12.15 [Invoquer `guix system`], page 584).

Il ne peut y avoir qu'une instance d'un type de service extensible comme `udev-service-type`. S'il y en avait plus, les spécifications `service-extension` seraient ambiguës.

Toujours ici ? La section suivante fournit une référence de l'interface de programmation des services.

### 12.18.3 Référence de service

Nous avons vu un résumé des types de services (voir Section 12.18.2 [Types service et services], page 600). Cette section fournit une référence sur la manière de manipuler les services et les types de services. Cette interface est fournie par le module (`gnu services`).

**service type [value]** [Procédure Scheme]

Renvoie un nouveau service de type `type`, un objet `<service-type>` (voir plus bas). `value` peut être n'importe quel objet ; il représente les paramètres de cette instance particulière de service.

Lorsque `value` est omise, la valeur par défaut spécifiée par `type` est utilisée ; si `type` ne spécifie pas de valeur par défaut, une erreur est levée.

Par exemple ceci :

```
(service openssh-service-type)
```

est équivalent à ceci :

```
(service openssh-service-type
  (openssh-configuration))
```

Dans les deux cas le résultat est une instance de `openssh-service-type` avec la configuration par défaut.

`service? obj` [Procédure Scheme]

Renvoie vrai si `obj` est un service.

`service-kind service` [Procédure Scheme]

Renvoie le type de `service` — c.-à-d. un objet `<service-type>`.

`service-value service` [Procédure Scheme]

Renvoie la valeur associée à `service`. Elle représente ses paramètres.

Voici un exemple de la manière dont un service est créé et manipulé :

```
(define s
  (service nginx-service-type
    (nginx-configuration
      (nginx nginx)
      (log-directory log-directory)
      (run-directory run-directory)
      (file config-file))))
```

```
(service? s)
```

```
⇒ #t
```

```
(eq? (service-kind s) nginx-service-type)
```

```
⇒ #t
```

La forme `modify-services` fournit une manière pratique de modifier les paramètres de certains services d'une liste comme `%base-services` (voir Section 12.9.1 [Services de base], page 268). Elle s'évalue en une liste de services. Bien sûr, vous pouvez toujours utiliser les combinateurs de liste standards comme `map` et `fold` pour cela (voir Section "SRFI-1" dans *GNU Guile Reference Manual*) ; `modify-services` fournit simplement une manière plus concise pour ce besoin commun.

`modify-services services (type variable => body) ...` [Syntaxe Scheme]

Modifie les services listés dans `services` en fonction des clauses données. Chaque clause à la forme :

```
(type variable => body)
```

où `type` est un type de service — p. ex. `guix-service-type` — et `variable` est un identifiant lié dans `body` aux paramètres du service — p. ex. une instance de `guix-configuration` — du service original de ce `type`.

La variable `body` devrait s'évaluer en de nouveaux paramètres de service, qui seront utilisés pour configurer le nouveau service. Ce nouveau service remplacera l'original

dans la liste qui en résulte. Comme les paramètres d'un service sont créés avec `define-record-type*`, vous pouvez écrire un *body* court qui s'évalue en de nouveaux paramètres pour le services en utilisant `inherit`, fourni par `define-record-type*`.

Les clauses peuvent aussi avoir les formes suivantes :

`(delete type)`

Cette clause supprime tous les services du *type* donné de *services*.

Voir Section 12.1 [Utiliser le système de configuration], page 238, pour des exemples d'utilisation.

Suit l'interface de programmation des types de services. Vous devrez la connaître pour écrire de nouvelles définitions de services, mais pas forcément lorsque vous cherchez des manières simples de personnaliser votre déclaration `operating-system`.

**service-type** [Type de données]

C'est la représentation d'un *type de service* (voir Section 12.18.2 [Types service et services], page 600).

**name** C'est un symbole, utilisé seulement pour simplifier l'inspection et le débogage.

**extensions**

Une liste non-vide d'objets `<service-extension>` (voir plus bas).

**compose** (par défaut : `#f`)

S'il s'agit de `#f`, le type de service dénote des services qui ne peuvent pas être étendus — c.-à-d. qui ne reçoivent pas de « valeurs » d'autres services.

Sinon, ce doit être une procédure à un argument. La procédure est appelée par `fold-services` et on lui passe une liste de valeurs collectées par les extensions. Elle peut renvoyer n'importe quelle valeur simple.

**extend** (par défaut : `#f`)

Si la valeur est `#f`, les services de ce type ne peuvent pas être étendus.

Sinon, il doit s'agir 'une procédure à deux arguments : `fold-services` l'appelle et lui passe la valeur initiale du service comme premier argument et le résultat de l'application de `compose` sur les valeurs d'extension en second argument. Elle doit renvoyer une valeur qui est une valeur de paramètre valide pour l'instance du service.

**description**

C'est une chaîne, éventuellement avec de la syntaxe Texinfo, décrivant une quelques phrases ce que le service fait. Cette chaîne permet aux utilisateurs de trouver le service à travers `guix system search` (voir Section 12.15 [Invoquer guix system], page 584).

**default-value** (par défaut : `&no-default-value`)

La valeur par défaut associée aux instances de ce type de service. Cela permet d'utiliser la forme `service` sans son second argument :

`(service type)`

Le service renvoyé dans ce cas a la valeur par défaut spécifiée par *type*.

Voir Section 12.18.2 [Types service et services], page 600, pour des exemples.

**service-extension** *target-type compute* [Procédure Scheme]

Renvoie une nouvelle extension pour les services de type *target-type*. *compute* doit être une procédure à un argument : **fold-services** l'appelle et lui passe la valeur associée au service qui fournit cette extension ; elle doit renvoyer une valeur valide pour le service cible.

**service-extension?** *obj* [Procédure Scheme]

Renvoie vrai si *obj* est une extension de service.

Parfois, vous voudrez simplement étendre un service existant. Cela implique de créer un nouveau type de service et de spécifier l'extension qui vous intéresse, ce qui peut être assez verbeux ; la procédure **simple-service** fournit un raccourci pour ce cas.

**simple-service** *name target value* [Procédure Scheme]

Renvoie un service qui étend *target* avec *value*. Cela fonctionne en créant un type de service singleton *name*, dont le service renvoyé est une instance.

Par exemple, cela étend **mcron** (voir Section 12.9.2 [Exécution de tâches planifiées], page 287) avec une tâche supplémentaire :

```
(simple-service 'my-mcron-job mcron-service-type
  #~(job '(next-hour (3)) "guix gc -F 2G"))
```

Au cœur de l'abstraction des services se cache la procédure **fold-services**, responsable de la « compilation » d'une liste de services en un répertoire unique qui contient tout ce qui est nécessaire au démarrage et à l'exécution du système — le répertoire indiqué par la commande **guix system build** (voir Section 12.15 [Invoquer guix system], page 584). En soit, elle propage les extensions des services le long du graphe des services, en mettant à jour chaque paramètre des nœuds sur son chemin, jusqu'à atteindre le nœud racine.

**fold-services** *services* [*#:target-type system-service-type*] [Procédure Scheme]

Replie *services* en propageant leurs extensions jusqu'à la racine de type *target-type* ; renvoie le service racine ajusté de cette manière.

Enfin, le module (**gnu services**) définit aussi divers types de services essentiels, dont certains sont listés ci-dessous.

**system-service-type** [Variable Scheme]

C'est la racine du graphe des services. Il produit le répertoire du système renvoyé par la commande **guix system build**.

**boot-service-type** [Variable Scheme]

Le type du service « boot », qui produit le *script de démarrage*. Le script de démarrage est ce que le disque de RAM initial lance au démarrage.

**etc-service-type** [Variable Scheme]

Le type du service */etc*. Ce service est utilisé pour créer des fichiers dans */etc* et peut être étendu en lui passant des tuples nom/fichier comme ceci :

```
(list `("issue" ,(plain-file "issue" "Bienvenue !\n")))
```



Dans cet exemple, l'effet serait d'ajouter un fichier `/etc/issue` pointant vers le fichier donné.

**setuid-program-service-type** [Variable Scheme]

Le type du « service setuid ». Ce service récupère des listes de noms de fichiers exécutables, passés en tant que gexps, et les ajoute à l'ensemble des programmes setuid et setgid root sur le système (voir Section 12.10 [Programmes setuid], page 571).

**profile-service-type** [Variable Scheme]

De type du service qui remplit le *profil du système* — c.-à-d. les programmes dans `/run/current-system/profile`. Les autres services peuvent l'étendre en lui passant des listes de paquets à ajouter au profil du système.

**provenance-service-type** [Variable Scheme]

C'est le type de service qui enregistre les *métadonnées de provenance* dans le système lui-même. Il crée plusieurs fichiers dans `/run/current-system` :

**channels.scm**

C'est un « fichier de canaux » qui peut être passé à `guix pull -C` ou `guix time-machine -C`, et qui décrit les canaux utilisés pour construire le système, si cette information était disponible (voir Chapitre 7 [Canaux], page 75).

**configuration.scm**

C'est le fichier qui était passé en argument à ce service **provenance-service-type**. Par défaut, `guix system reconfigure` passe automatiquement le fichier de configuration du système qu'il a reçu en argument de la ligne de commande.

**provenance**

Cela contient la même information que les deux autres fichiers mais dans un format plus facile à traiter automatiquement.

En général, ces deux informations (les canaux et le fichier de configuration) sont suffisants pour reproduire le système d'exploitation « depuis les sources ».

**Limites:** Cette information est nécessaire pour reconstruire votre système d'exploitation, mais elle n'est pas suffisante. En particulier, `configuration.scm` lui-même n'est pas suffisant s'il n'est pas auto-contenu — s'il référence des modules Guile externes ou des fichiers supplémentaires. Si vous voulez que `configuration.scm` soit auto-contenu, nous vous recommandons de faire en sorte que les modules et les fichiers auxquels il fait référence se trouvent dans un canal.

De plus, les métadonnées de provenance sont « silencieuses » dans le sens où elles ne changent pas les bits contenus dans votre système, *en dehors des bits des métadonnées elles-mêmes*. Deux configurations de système différents ou des ensembles de canaux différents peuvent créer le même système, bit-à-bit, mais lorsque **provenance-service-type** est utilisé ces deux systèmes auront des métadonnées différentes et donc des noms de fichiers dans le dépôt différents, ce qui rend la comparaison plus difficile.

Ce service est automatiquement ajouté à la configuration de votre système d'exploitation quand vous utilisez `guix system reconfigure`, `guix system init` et `guix deploy`.

`linux-loadable-module-service-type` [Variable Scheme]

Type du service qui récolte les listes de paquets contenant les modules chargeables par le noyau, et les ajoute à l'ensemble des modules chargeables par le noyau.

Ce type de service est conçu pour être étendu par d'autres types de services, comme ceci :

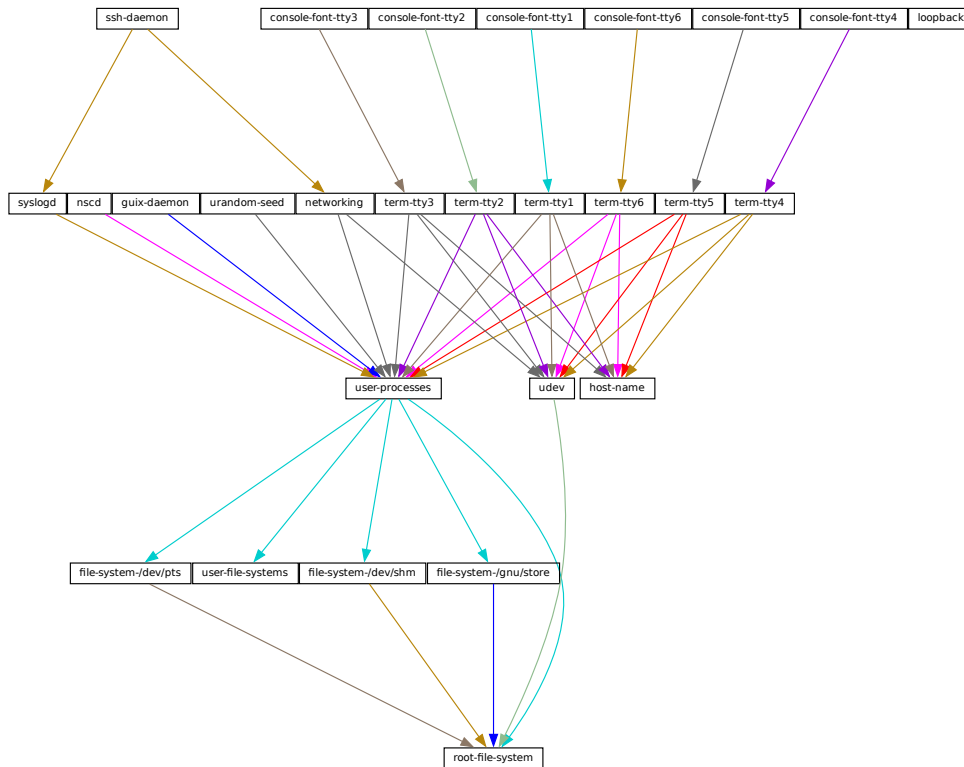
```
(simple-service 'installing-module
               linux-loadable-module-service-type
               (list module-to-install-1
                     module-to-install-2))
```

Ce service ne charge pas les modules au démarrage, il ne fait que les ajouter au profil du noyau pour qu'ils *puissent* être chargés par d'autres moyens.

#### 12.18.4 Services Shepherd

Le module (`gnu services shepherd`) fournit une manière de définir les services gérés par le GNU Shepherd, qui est le système d'initialisation — le premier processus démarré lorsque le système démarre, aussi connu comme étant le PID 1 (voir Section “Introduction” dans *The GNU Shepherd Manual*).

Les services dans le Shepherd peuvent dépendre les uns des autres. Par exemple, le démon SSH peut avoir besoin d'être démarré après le démon syslog, qui à son tour doit être démarré après le montage des systèmes de fichiers. Le système d'exploitation simple déclaré précédemment (voir Section 12.1 [Utiliser le système de configuration], page 238) crée un graphe de service comme ceci :



Vous pouvez générer un tel graphe pour n'importe quelle définition de système d'exploitation avec la commande `guix system shepherd-graph` (voir [system-shepherd-graph], page 593).

La variable `%shepherd-root-service` est un objet de service représentant le PID 1, de type `shepherd-root-service-type` ; il peut être étendu en lui passant des listes d'objets `<shepherd-service>`.

**shepherd-service** [Type de données]

Le type de données représentant un service géré par le Shepherd.

**provision**

C'est une liste de symboles dénotant ce que le service fournit.

Ce sont les noms qui peuvent être passés à `herd start`, `herd status` et les commandes similaires (voir Section “Invoking herd” dans *The GNU Shepherd Manual*). Voir Section “Slots of services” dans *The GNU Shepherd Manual*, pour plus de détails.

**requirement** (par défaut : '() )

Liste de symboles dénotant les services du Shepherd dont celui-ci dépend.

- one-shot?** (par défaut : #f)  
Indique si ce service est *ponctuel*. Les services ponctuels s'arrêtent immédiatement à la fin de leur action **start**. Voir Section "Slots of services" dans *The GNU Shepherd Manual*, pour plus d'infos.
- respawn?** (par défaut : #t)  
Indique s'il faut redémarrer le service lorsqu'il s'arrête, par exemple si le processus sous-jacent meurt.
- start**
- stop** (par défaut : #~(const #f))  
Les champs **start** et **stop** se réfèrent à la capacité du Shepherd de démarrer et d'arrêter des processus (voir Section "Service De- and Constructors" dans *The GNU Shepherd Manual*). Ils sont donnés comme des G-expressions qui sont étendues dans le fichier de configuration du Shepherd (voir Section 9.12 [G-Expressions], page 166).
- actions** (par défaut : '())  
C'est une liste d'objets **shepherd-action** (voir plus bas) définissant des *actions* supportées par le service, en plus des actions **start** et **stop** standards. Les actions listées ici sont disponibles en tant que sous-commande de **herd** :
- ```
herd action service [arguments...]
```
- auto-start?** (par défaut : #t)  
Indique si ce service doit être démarré automatiquement par le Shepherd. Si la valeur est #f, le service doit être démarré manuellement avec **herd start**.
- documentation**  
Une chaîne de documentation, montrée lorsqu'on lance :
- ```
herd doc service-name
```
- où *service-name* est l'un des symboles dans *provision* (voir Section "Invoking herd" dans *The GNU Shepherd Manual*).
- modules** (par défaut : %default-modules)  
C'est la liste des modules qui doivent être dans le contexte lorsque **start** et **stop** sont évalués.

L'exemple ci-dessous définit un service Shepherd qui lance **syslogd**, le démon de journalisation du système des utilitaires réseau de GNU (voir Section "syslogd invocation" dans *GNU Inetutils*) :

```
(let ((config (plain-file "syslogd.conf" "...")))
  (shepherd-service
    (documentation "Run the syslog daemon (syslogd).")
    (provision '(syslogd))
    (requirement '(user-processes))
    (start #~(make-forkexec-creator
              (list #$(file-append inetutils "/libexec/syslogd")
                    "--rcfile" #$(config))
```

```
#:pid-file "/var/run/syslog.pid"))
(stop #~(make-kill-destroyer)))
```

Les éléments clés de cet exemple sont les champs `start` et `stop` : ce sont des bouts de code *échelonné* qui utilisent la procédure `make-forkexec-creator` fournie par le Shepherd et son dual, `make-kill-destroyer` (voir Section “Service De- and Constructors” dans *The GNU Shepherd Manual*). Le champ `start` fera créer un `syslogd` au `shepherd` avec les options données. Remarquez que nous passons `config` après `--rcfile`, qui est un fichier de configuration déclaré plus haut (le contenu de ce fichier a été omis). De même, le champ `stop` décrit comment ce service sera arrêté. Dans ce cas, il est arrêté par l’appel système `kill` sur son PID. Le code est échelonné par des G-expressions : `#~` échelonne le code, tandis que `#$` « descend » vers le code hôte (voir Section 9.12 [G-Expressions], page 166).

**shepherd-action** [Type de données]

C’est le type de données qui définit des actions supplémentaires implémentées par un service Shepherd (voir au-dessus).

**name**           Symbole nommant l’action.

**documentation**

C’est une chaîne de documentation pour l’action. Elle peut être consultée avec :

```
herd doc service action action
```

**procedure**

Cela devrait être une `gexp` qui s’évalue en une procédure à au moins un argument, la « valeur de lancement » du service (voir Section “Slots of services” dans *The GNU Shepherd Manual*).

L’exemple suivant définit une action nommée `dire-bonjour` qui salue amicalement l’utilisateur :

```
(shepherd-action
  (name 'dire-bonjour)
  (documentation "Dit salut !")
  (procedure #~(lambda (running . args)
    (format #t "Salut, l'ami ! arguments : ~s\n"
      args)
    #t)))
```

En supposant que cette action est ajoutée dans le service `example`, vous pouvez écrire :

```
# herd dire-bonjour example
Salut, l'ami ! arguments : ()
# herd dire-bonjour example a b c
Salut, l'ami ! arguments : ("a" "b" "c")
```

Comme vous pouvez le voir, c’est une manière assez sophistiquée de dire bonjour. Voir Section “Service Convenience” dans *The GNU Shepherd Manual*, pour plus d’informations sur les actions.

**shepherd-configuration-action** [Procédure Scheme]

Renvoie une action `configuration` pour afficher *file*, qui devrait être le nom du fichier de configuration du service.

Il peut être utile d'équiper les services avec cette action. Par exemple, le service pour le routeur anonyme Tor (voir Section 12.9.5 [Services réseau], page 299) est défini globalement de cette façon :

```
(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-destroyer))
   (actions (list (shepherd-configuration-action torrc)))
   (documentation "Run the Tor anonymous network overlay.")))
```

Grâce à cette action, la personne administrant le système peut inspecter le fichier de configuration passé à `tor` avec cette commande shell :

```
cat $(herd configuration tor)
```

Cela peut être pratique pour déboguer !

**shepherd-root-service-type** [Variable Scheme]

Le type de service pour le « service racine » du Shepherd — c.-à-d. le PID 1.

C'est le type de service que les extensions ciblent lorsqu'elles veulent créer un service `shepherd` (voir Section 12.18.2 [Types service et services], page 600, pour un exemple). Chaque extension doit passer une liste de `<shepherd-service>`. Sa valeur doit être un `shepherd-configuration`, décrit plus bas.

**shepherd-configuration** [Type de données]

Ce type de données représente la configuration du Shepherd.

`shepherd` (par défaut : `shepherd`)

Le paquet Shepherd à utiliser.

`services` (par défaut : `'()`)

Une liste de `<shepherd-service>` à démarrer. Vous devriez probablement utiliser le mécanisme d'extensions des services à la place (voir Section 12.18.4 [Services Shepherd], page 607).

L'exemple suivant spécifie le paquet Shepherd pour le système d'exploitation :

```
(operating-system
 ;; ...
 (services (append (list openssh-service-type)
                  ;; ...
                  %desktop-services)
 ;; ...
```

```
;; Utiliser notre propre paquet Shepherd.
(essential-services
 (modify-services (operating-system-default-essential-services
                  this-operating-system)
 (shepherd-root-service-type config => (shepherd-configuration
                                       (inherit config)
                                       (shepherd my-shepherd))))))
```

`%shepherd-root-service` [Variable Scheme]  
Ce service représente le PID 1.

### 12.18.5 Configurations complexes

Certains programmes peuvent avoir des fichiers ou des formats de configuration complexes, et pour rendre plus simple la création de liaisons Scheme pour ces fichiers de configuration, vous pouvez utiliser les fonction auxiliaires définies dans le modules (`gnu services configuration`).

L'utilitaire principal est la macro `define-configuration`, que vous utiliserez pour définir un type d'enregistrement Scheme (voir Section "Record Overview" dans *GNU Guile Reference Manual*). L'enregistrement Scheme sera sérialisé en un fichier de configuration par des *sérialiseurs*, qui sont des procédures qui prennent une valeur Scheme et renvoient une G-expression (voir Section 9.12 [G-Expressions], page 166), ce qui devrait, une fois sérialisé sur le disque, renvoyer une chaîne. Plus de détails se trouvent plus bas.

`define-configuration` *nom clause1 clause2 ...* [Syntaxe Scheme]  
Crée un type d'enregistrement nommé *nom* qui contient les champs qui se trouvent dans les clauses.

Une clause peut avoir l'une des formes suivantes :

```
(nom-du-champ
 (type valeur-par-défaut)
 documentation)
```

```
(nom-du-champ
 (type valeur-par-défaut)
 documentation
 sérialiseur)
```

```
(nom-du-champ
 (type)
 documentation)
```

```
(nom-du-champ
 (type)
 documentation
 sérialiseur)
```

*nom-du-champ* est un identifiant qui dénote le nom du champ dans l'enregistrement généré.

*type* est le type de valeur correspondant à *nom-du-champ* ; comme Guile n'est pas typé, une procédure de prédicat — *type?* — sera appelée avec la valeur correspondant au champ pour s'assurer que la valeur est du type correct. Cela signifie que si *type* est `package`, alors une procédure nommée `package?` sera appliquée sur la valeur pour s'assurer que c'est bien un objet `<package>`.

*valeur-par-défaut* est la valeur par défaut correspondant au champ ; si aucune n'est spécifiée, l'utilisateur ou l'utilisatrice doit fournir une valeur à la création d'un objet de ce type d'enregistrement.

*documentation* est une chaîne formatée avec la syntaxe de Texinfo qui fournit une description de ce que ce champ de configuration signifie.

*sérialiseur* est le nom d'une procédure qui prend deux arguments, le premier est le nom du champ, et le second est la valeur correspondant au champ. La procédure devrait renvoyer une chaîne ou une G-expression (voir Section 9.12 [G-Expressions], page 166) qui représente le contenu qui sera sérialisé dans le fichier de configuration. Si aucun n'est spécifié, une procédure nommée `serialize-type` sera utilisée.

Une simple procédure de sérialisation pourrait ressembler à ceci :

```
(define (serialize-boolean field-name value)
  (let ((value (if value "true" "false")))
    #~(string-append #field-name #value)))
```

Dans certains cas plusieurs enregistrements de configuration différents peuvent être définis dans le même fichier, mais leurs sérialiseurs pour le même type peuvent devoir être différents, à cause de formats de configuration différents. Par exemple, la procédure `serialize-boolean` pour le service Getmail doit être différente de celle pour le service Transmission. Pour faciliter cette situation, on peut spécifier un préfixe pour sérialiseur en utilisant le littéral `prefix` dans la forme `define-configuration`. Cela signifie qu'on n'a pas à spécifier manuellement un *sérialiseur* personnalisé pour chaque champ.

```
(define (toto-serialize-string field-name value)
  ...)
```

```
(define (titi-serialize-string field-name value)
  ...)
```

```
(define-configuration toto-configuration
  (label
   (string)
   "Le nom de l'étiquette.")
  (prefix toto-))
```

```
(define-configuration titi-configuration
  (ip-address
   (string)
   "L'adresse IPv4 de cet appareil.")
  (prefix titi-))
```



Cependant, dans certains cas vous ne voudrez pas sérialiser les valeurs de l'enregistrement. Pour cela, vous pouvez utiliser le littéral `no-serialization`. Il y a aussi la macro `define-configuration/no-serialization` qui est un raccourci.

```
;; Rien ne sera sérialisé sur le disque.
(define-configuration toto-configuration
  (field
    (string "test")
    "De la documentation.")
  (no-serialization))

;; Comme au-dessus.
(define-configuration/no-serialization titi-configuration
  (field
    (string "test")
    "De la documentation."))
```

**define-maybe type** [Syntaxe Scheme]

Parfois un champ ne devrait pas être sérialisé si l'utilisateur ou l'utilisatrice de spécifie par de valeur. Pour cela, vous pouvez utiliser la macro `define-maybe` pour définir un « type peut-être » ; si la valeur d'un type peut-être n'est pas spécifiée, ou est indiquée à `%unset-value`, elle n'est pas sérialisée.

Lorsque vous définissez un « type peut-être », le sérialiseur correspondant au type normal sera utilisé par défaut. Par exemple, un champ de type `maybe-string` sera sérialisé avec la procédure `serialize-string` par défaut, vous pouvez évidemment changer cela en spécifiant une procédure de sérialisation personnalisée. De la même manière, le type de la valeur doit être une chaîne, ou la valeur doit rester non spécifiée.

```
(define-maybe string)

(define (serialize-string field-name value)
  ...)

(define-configuration tata-configuration
  (name
    ;; Si c'est une chaîne, la procédure « serialize-string » sera utilisée
    ;; pour sérialiser la chaîne. Sinon ce champ n'est pas sérialisé
    maybe-string
    "Le nom de ce module."))
```

Comme avec `define-configuration`, on peut indiquer un préfixe pour le nom de sérialiseur en utilisant le littéral `prefix`.

```
(define-maybe integer
  (prefix tata-))

(define (tata-serialize-integer field-name value)
  ...)
```

Il y a aussi le littéral `no-serialization`, qui s'il est utilisé indique qu'aucun sérialiseur ne sera défini pour le « type peut-être », indépendamment de si la valeur

est indiquée ou non. `define-maybe/no-serialization` est un raccourci pour spécifier le littéral `no-serialization`.

```
(define-maybe/no-serialization symbol)

(define-configuration/no-serialization test-configuration
  (mode
   maybe-symbol
   "Docstring."))
```

`maybe-value-set? valeur` [Procédure Scheme]  
Prédicat pour vérifier si l'utilisateur ou l'utilisatrice a spécifié la valeur d'un champ peut-être.

`serialize-configuration configuration champs` [Procédure Scheme]  
Renvoie une G-expression qui contient les valeurs correspondant aux *champs* de *configuration*, un enregistrement qui a été généré par `define-configuration`. La G-expression peut ensuite être sérialisée vers le disque en utilisant par exemple `mixed-text-file`.

`empty-serializer champ-de-nom valeur` [Procédure Scheme]  
Un sérialiseur qui renvoie juste une chaîne vide. La procédure `serialize-package` est un alias pour cette procédure.

Une fois que vous avez défini un enregistrement de configuration, vous voudrez aussi sans doute le documenter pour que d'autres personnes sachent comment l'utiliser. Pour vous aider, il y a deux procédures qui sont documentées plus bas.

`generate-documentation documentation` [Procédure Scheme]  
*documentation-name*

Génère un fragment Texinfo à partir de docstrings dans *documentation*, une liste de (*étiquette champs sous-documentation ...*). *étiquette* devrait être un symbole et devrait être le nom de l'enregistrement de configuration. *champs* devrait être une liste de tous les champs disponibles pour l'enregistrement de configuration.

*sous-documentation* est un tuple (*nom-de-champ nom-de-configuration*). *nom-de-champ* est le nom du champ qui prend un autre enregistrement de configuration comme valeur, et *nom-de-configuration* est le nom de cet enregistrement de configuration.

*sous-documentation* n'est requis que s'il y a des enregistrements de configuration imbriqués. Par exemple, l'enregistrement `getmail-configuration` (voir Section 12.9.12 [Services de courriels], page 372) accepte un enregistrement `getmail-configuration-file` dans l'un de ses champs `rcfile`, donc la documentation de `getmail-configuration-file` est imbriquée dans `getmail-configuration`.

```
(generate-documentation
  `((getmail-configuration ,getmail-configuration-fields
    (rcfile getmail-configuration-file))
    ...))
'getmail-configuration)
```

*nom-de-documentation* devrait être un symbole et devrait être le nom de l'enregistrement de configuration.

**configuration->documentation** [Procédure Scheme]  
*configuration-symbol* Prend *configuration-symbol*, le symbole correspondant au nom utilisé pour définir un enregistrement de configuration avec **define-configuration**, et affiche la documentation Texinfo pour ses champs. Cette procédure est utile s'il n'y a pas d'enregistrement de configuration imbriqué car elle n'affiche que la documentation des champs de plus haut niveau.

Actuellement, il n'y a pas de manière automatique de générer la documentation pour les enregistrements de configuration et les ajouter au manuel. Au lieu de cela, chaque fois que vous faites un changement dans les docstrings d'un enregistrement de configuration, vous devez appeler manuellement **generate-documentation** ou **configuration->documentation** et coller la sortie dans le fichier `doc/guix.texi`.

Ci-dessous se trouve un exemple d'un type d'enregistrement créé avec **define-configuration** et compagnie.

```
(use-modules (gnu services)
             (guix gexp)
             (gnu services configuration)
             (srfi srfi-26)
             (srfi srfi-1))

;; Transforme les noms de champs, qui sont des symboles Scheme, en chaînes
(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)
   "Le nom du contact."
   serialize-contact-name)
  (phone-number
   maybe-integer
   "Le numéro de téléphone de la personne.")
  (email
   maybe-string
   "L'adresse de courriel de la personne.")
  (married?
   (boolean)
   "Indique si la personne est mariée.))

(define-configuration contacts-list-configuration
  (name
   (string)
   "Le nom du propriétaire de cette liste de contacts.")
  (email
   (string)
   "L'adresse de courriel du propriétaire.")
  (contacts
   (list-of-contact-configurations '()))
  "Une liste d'enregistrements @code{contact-configuration} qui contiennent■
des informations sur tous vos contacts.))

```

Une configuration de liste de contacts pourrait alors être créée de cette manière :

```

(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))))))
```

Après la sérialisation de la configuration sur le disque, le fichier qui en résulte ressemble à ceci :

```
[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 Configuration du dossier personnel

Guix prend en charge la configuration déclarative des *environnements personnels* en utilisant le mécanisme de configuration décrit au chapitre précédent (voir Section 12.18 [Définir des services], page 599), mais pour les paquets et les fichiers de configuration des utilisateurs et utilisatrices. Cela fonctionne aussi bien sur le système Guix que sur les distributions externes et vous permet de déclarer tous les paquets et services qui devraient être installés et configurés. Une fois que vous avez écrit un fichier contenant un enregistrement `home-environment`, cette configuration peut être *instanciée* par un utilisateur ou une utilisatrice non privilégié-e avec la commande `guix home` (voir Section 13.4 [Invoquer guix home], page 636).

**Remarque:** La fonctionnalité décrite dans cette section est toujours en cours de développement et est sujette à changement. Contactez-nous sur [guix-devel@gnu.org](mailto:guix-devel@gnu.org) !

Votre environnement personnel consiste en général de trois parties : les logiciels, la configuration et l'état. Dans les distributions populaires, les logiciels sont habituellement installés au niveau du système, mais avec GNU Guix, la plupart des paquets peuvent être installés indépendamment par chaque utilisateur et utilisatrice sans nécessiter de privilèges root, et font donc partie de votre *environnement personnel*. Les paquets seuls ne sont généralement pas très utiles car ils nécessitent de la configuration supplémentaire, typiquement dans des fichiers de configuration dans `XDG_CONFIG_HOME` (`~/.config` par défaut) ou d'autres répertoires. Tout le reste peut être considéré comme faisant partie de l'état, comme les fichiers multimédia, les bases de données des applications et les journaux.

Utiliser Guix pour gérer votre environnement personnel apporte de nombreux avantages :

- Tous vos logiciels peuvent être configurés en un unique langage (Guile Scheme), cela vous donne la possibilité de partager des valeurs entre les configurations de plusieurs programmes.
- Un environnement personnel bien défini est auto-contenu et peut être créé de manière déclarative et reproductible — pas besoin de récupérer des binaires externes ou de modifier des fichiers de configuration.
- Après chaque `guix home reconfigure`, une nouvelle génération de l'environnement personnel sera créée. Cela signifie que vous pouvez revenir à une génération précédente de votre environnement personnel et que vous n'avez pas à craindre de casser votre configuration.
- Il est possible de gérer les données avec état avec Guix Home, ce qui inclut la possibilité de cloner automatiquement des dépôts Git lors de la première configuration de la machine, et de lancer régulièrement des commandes comme `rsync` pour synchroniser vos données entre vos hôtes. Cette fonctionnalité est cependant toujours expérimentale.

### 13.1 Déclarer l'environnement personnel

L'environnement personnel est configuré en fournissant une déclaration `home-environment` dans un fichier qui peut être passé à la commande `guix home` (voir Section 13.4 [Invoquer guix home], page 636). Le plus facile pour commencer est de générer une configuration initiale avec `guix home import` :

```
guix home import ~/src/guix-config
```

La commande `guix home import` lit certains « dot files » comme `~/bashrc` qui se trouvent dans votre dossier personnel et les copie dans le dossier donné, `~/src/guix-config` dans notre exemple. Il lit aussi le contenu de votre profil, `~/guix-profile`, et, en fonction de cela, il remplit `~/src/guix-config/home-configuration.scm` avec une configuration du dossier personnel qui ressemble à votre configuration actuelle.

Une configuration simple peut inclure par exemple Bash et une configuration textuelle personnalisée, comme dans l'exemple ci-dessous. N'ayez pas peur de déclarer des parties de votre environnement personnel qui correspondent à des fichiers de configuration existants : avant d'installer les fichiers de configuration, Guix Home créera une sauvegarde des fichiers existants à un emplacement distinct de votre dossier personnel.

**Remarque:** Il est fortement recommandé de gérer votre shell avec Guix Home, car il s'assurera que tous les scripts nécessaires seront sourcés par le fichier de configuration du shell. Sinon, vous devrez le faire manuellement. (voir Section 13.2 [Configurer le shell], page 621).

```
(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"))))))))
```

Le champ `packages` devrait être évident, il installera la liste des paquets dans votre profil. Le champ le plus important est `services`, qui contient une liste de *services personnels*, qui sont les blocs de base d'un environnement personnel.

Il n'y a pas de démon (en tout cas pas nécessairement) lié à un service personnel, un service personnel est seulement un élément qui est utilisé pour déclarer une partie de

l’environnement personnel et en étendre d’autres parties. Le mécanisme d’extension présenté dans le chapitre précédent (voir Section 12.18 [Définir des services], page 599) ne doit pas être confondu avec les services Shepherd (voir Section 12.18.4 [Services Shepherd], page 607). L’utilisation de ce mécanisme d’extension et de code Scheme autour vous donne la liberté de déclarer votre propre environnement personnel, hautement personnalisé.

Une fois que la configuration a l’air bonne, vous pouvez d’abord la tester dans un « conteneur » jetable :

```
guix home container config.scm
```

La commande ci-dessus démarre un shell où votre environnement personnel est lancé. Le shell tourne dans un conteneur, ce qui signifie qu’il est isolé du reste du système, donc c’est un bon moyen d’essayer votre configuration — vous pouvez vérifier s’il manque des bouts de configuration ou si elle ne se comporte pas correctement, si les démons sont démarrés, etc. Une fois que vous quittez ce shell, vous retournez à l’invite de commande de votre shell de départ « dans le monde réel ».

Une fois que votre configuration correspond à vos besoins, vous pouvez reconfigurer votre dossier personnel en exécutant :

```
guix home reconfigure config.scm
```

Cette commande « construit » votre environnement personnel et crée un lien `~/guix-home` qui pointe dessus. Et voilà !

**Remarque:** Assurez-vous que votre système d’exploitation a `elogind`, `systemd` ou un mécanisme similaire pour créer les répertoires XDG à l’exécution et défini la variable `XDG_RUNTIME_DIR`. Autrement, le script `on-first-login` ne sera pas exécuté et les processus comme le Shepherd utilisateur et ses descendants ne démarreront pas.

## 13.2 Configurer le shell

Vous pouvez sauter cette section sans problème si votre shell ou vos shells sont gérés par Guix Home. Sinon, lisez-la avec attention.

Il y a quelques scripts qui doivent être évalués par un shell de connexion pour activer l’environnement personnel. Les fichiers de démarrage du shell lus par les shells de connexion ont souvent le suffixe `profile`. Pour plus d’information sur les shells de connexion, voir Section “Invoking Bash” dans *The GNU Bash Reference Manual* et voir Section “Bash Startup Files” dans *The GNU Bash Reference Manual*.

Le premier script qui a besoin d’être sourcé est `setup-environment`, qui initialise toutes les variables d’environnement nécessaires (dont les variables que vous avez déclarées) et le second est `on-first-login`, qui démarre Shepherd pour l’utilisateur actuel et effectue les actions déclarées par les autres services personnels qui étendent `home-run-on-first-login-service-type`.

Guix Home créera toujours `~/profile`, qui contient les lignes suivantes :

```
HOME_ENVIRONMENT=$HOME/.guix-home
. $HOME_ENVIRONMENT/setup-environment
$HOME_ENVIRONMENT/on-first-login
```

Cela fait activer l’environnement personnel sur les shell POSIX. Cependant, dans la plupart des cas ce fichier ne sera pas lu par les shells les plus modernes, parce qu’ils sont lancés



en mode non POSIX par défaut et qu'ils ont leur propre fichiers de démarrage `*profile`. Par exemple Bash préférera `~/.bash_profile` s'il existe et n'utilisera `~/.profile` que s'il n'existe pas. Zsh (sans option supplémentaire) ignorera `~/.profile` même si `~/.zprofile` n'existe pas.

Pour que votre shell respecte `~/.profile`, ajoutez `. ~/.profile` ou `source ~/.profile` au fichier de démarrage pour le shell de connexion. Dans le cas de eBash, c'est `~/.bash_profile`, et dans le cas de Zsh, c'est `~/.zprofile`.

**Remarque:** Cette étape n'est requise que si votre shell n'est *pas* géré par Guix HOME. Sinon, tout sera déjà fait automatiquement.

### 13.3 Services du dossier personnel

Un *service personnel* ne correspond pas forcément à un démon géré par le Shepherd (voir Section “Jump Start” dans *The GNU Shepherd Manual*), dans la plupart des cas ce n'est pas le cas. C'est un simple bloc de construction de l'environnement personnel, souvent déclaré comme une ensemble de paquets à installer dans le profil de l'environnement personnel, un ensemble de fichiers de configuration vers lesquels créer des liens symboliques dans `XDG_CONFIG_HOME` (`~/.config` par défaut) et des variables d'environnement à initialiser dans un shell de connexion.

Il y a un mécanisme d'extension de services (voir Section 12.18.1 [Composition de services], page 599) qui permet aux services personnels d'étendre les autres services personnels et d'utiliser leurs fonctionnalités ; par exemple : déclarer des tâches mcron (voir *GNU Mcron*) en étendant Section 13.3.3 [Service personnel Mcron], page 628, ; déclarer des démons en étendant Section 13.3.5 [Service Shepherd personnel], page 630, ; ajouter des commandes qui seront invoquées par Bash en étendant Section 13.3.2 [Services shells personnels], page 625.

Une bonne manière de découvrir les services personnels disponibles est d'utiliser la commande `guix home search` (voir Section 13.4 [Invoquer guix home], page 636). Après avoir trouvé les services personnels requis, ajoutez son module avec la forme `use-modules` (voir Section “Using Guile Modules” dans *The GNU Guile Reference Manual*), ou la directive `#:use-modules` (voir Section “Creating Guile Modules” dans *The GNU Guile Reference Manual*) et déclarez un service personnels avec la fonction `service`, ou étendez un type de service en déclarant un nouveau service avec la procédure `simple-service` de (`gnu services`).

#### 13.3.1 Services personnels essentiels

Il y a quelques services essentiels définis dans (`gnu services`), ils sont surtout utilisés en interne et sont requis pour construire un environnement personnel, mais certains vous seront utiles.

`home-environment-variables-service-type` [Variable Scheme]

Le service de ce type sera instancié par chaque environnement personnel automatiquement par défaut, il n'y a pas besoin de le définir, mais vous voudrez peut-être l'étendre avec une liste de paires pour initialiser certaines variables d'environnement.

```
(list ("ENV_VAR1" . "valeur1")
      ("ENV_VAR2" . "valeur2"))
```

Le plus simple pour étendre un type de service, sans définir un nouveau type de service est d'utiliser la fonction `simple-service` de (`gnu services`).

```
(simple-service 'un-service-de-variables-utiles
home-environment-variables-service-type
`(("LESSHISTFILE" . "$XDG_CACHE_HOME/.lesshst")
  ("SHELL" . ,(file-append zsh "/bin/zsh"))
  ("USELESS_VAR" . #f)
  ("_JAVA_AWT_WM_NONREParentING" . #t)))
```

Si vous incluez un tel service dans votre définition d'environnement personnel, il ajoutera le contenu suivant au script `setup-environment` (qui doit être sourcé par le shell de connexion) :

```
export LESSHISTFILE=$XDG_CACHE_HOME/.lesshst
export SHELL=/gnu/store/2hsg15n644f0glrcbkb1kqknmmqdar03-zsh-5.8/bin/zsh
export _JAVA_AWT_WM_NONREParentING
```

**Remarque:** Assurez-vous que le module (`gnu packages shells`) est importé avec `use-modules` ou d'une autre façon ; cet espace de nom contient la définition du paquet `zsh`, utilisé dans l'exemple précédent.

La liste d'association (voir Section "Association Lists" dans *le manuel de référence de Guile*) est une structure de données contenant des paires de clé-valeurs. Pour `home-environment-variables-service-type` la clé est toujours une chaîne, la valeur peut être une chaîne, une gexp s'évaluant en une chaîne (voir Section 9.12 [G-Expressions], page 166), un objet simili-fichier (voir Section 9.12 [G-Expressions], page 166) ou un booléen. Pour les gexps, la variable sera initialisée à la valeur de la gexp ; pour les objets simili-fichiers, elle sera initialisée au chemin du fichier dans le dépôt (voir Section 9.9 [Le dépôt], page 157) ; pour `#t`, la variable sera exportée sans valeur et pour `#f`, la variable sera omise.

`home-profile-service-type` [Variable Scheme]

Le service de ce type sera instancié par chaque environnement personnel automatiquement, il n'y a pas besoin de le définir, mais vous pouvez l'étendre avec une liste de paquets si vous voulez installer des paquets supplémentaires dans votre profil. D'autres services, qui ont besoin de rendre certains programmes disponible à l'utilisateur ou l'utilisatrice étendront aussi ce type de service.

La valeur d'extension est simplement une liste de paquets :

```
(list htop vim emacs)
```

Vous pouvez utiliser la même approche avec `simple-service` (voir Section 12.18.3 [Référence de service], page 602) que pour `home-environment-variables-service-type`. Assurez-vous que les modules contenant les paquets spécifiés sont importés avec `use-modules`. Pour trouver un paquet ou des informations à propos de son module utilisez `guix search` voir Section 6.2 [Invoquer guix package], page 44). Autrement, vous pouvez utiliser `specification->package` pour récupérer l'enregistrement de paquet à partir d'une chaîne sans importer le module correspondant.

Il y a quelques autres services essentiels, mais vous n'avez pas besoin de les étendre.

**home-service-type** [Variable Scheme]

La racine du graphe des services personnels, elle génère un dossier qui sera plus tard lié à partir de `~/.guix-home`, il contient les configurations, le profil avec les binaires et les bibliothèques et certains scripts nécessaires pour tout faire fonctionner ensemble.

**home-run-on-first-login-service-type** [Variable Scheme]

Le service de ce type génère un script Guile qui devrait être lancé par le shell de connexion. Il n'est exécuté que si le fichier spécial dans `XDG_RUNTIME_DIR` n'a pas été créé, ce qui évite les exécutions supplémentaires de ce script si plusieurs shells de connexion sont ouverts.

Il peut être étendu avec une gexp. Cependant, pour démarrer une application automatiquement, vous ne *devriez pas* utiliser ce service, dans la plupart des cas il vaut mieux étendre **home-shepherd-service-type** avec un service Shepherd (voir Section 12.18.4 [Services Shepherd], page 607), ou en étendant le fichier de démarrage de shell avec la commande requise en utilisant le type de service approprié.

**home-files-service-type** [Variable Scheme]

Le service de ce type permet de spécifier une liste de fichiers, qui iront dans `~/.guix-home/files`. Ce répertoire contient généralement des fichiers de configuration (pour être plus précis, il contient des liens symboliques vers des fichiers de `/gnu/store`) qui devraient être placés dans `$XDG_CONFIG_DIR` ou dans de rares cas dans `$HOME`. Il accepte des valeurs d'extension de cette forme :

```
`(("sway/config" ,sway-file-like-object)
  ("tmux.conf" ,(local-file "./tmux.conf")))
```

Chaque liste imbriquée contient deux valeurs : un sous-répertoire et un objet similitudine fichier. Après la construction d'un environnement personnel `~/.guix-home/files` contiendra le contenu approprié et tous les répertoires imbriqués seront créés en fonction. Cependant, ces fichiers n'iront nul part ailleurs à moins qu'un autre service s'en occupe. Par défaut un service **home-symlink-manager-service-type**, qui crée les liens symboliques nécessaires vers les fichiers de `~/.guix-home/files` et sauvegarde les fichiers existants en conflit et d'autres choses, fait partie des services personnels essentiels (activés par défaut), mais il est possible d'utiliser des services alternatifs pour implémenter des cas d'usage plus avancés comme un dossier personnel en lecture-seule. Expérimentez et partagez vos trouvailles.

**home-configuration-files-service-type** [Variable Scheme]

Le service est très semblable au service **home-files-service-type** (et l'étend en fait), mais il est utilisé pour définir des fichiers qui iront dans `~/.guix-home/files/.config`, et qui seront liés symboliquement dans `$XDG_CONFIG_DIR` par **home-symlink-manager-service-type** (par exemple) à l'activation. Il accepte des valeurs d'extension dans ce format :

```
`(("sway/config" ,sway-file-like-object)
  ;; -> ~/.guix-home/files/.config/sway/config
  ;; -> $XDG_CONFIG_DIR/sway/config (par symlink-manager)
  ("tmux/tmux.conf" ,(local-file "./tmux.conf")))
```

**home-activation-service-type** [Variable Scheme]

Le service de ce type génère un script guile qui est lancé à chaque invocation de `guix home reconfigure` ou n'importe quelle autre action qui active l'environnement personnel.

**home-symlink-manager-service-type** [Variable Scheme]

Le service de ce type génère un script guile qui sera exécuté à l'activation de l'environnement personnel, et effectuera les actions suivantes :

1. Lire le contenu du répertoire `files/` des environnements personnels actuels et en attente.
2. Nettoyer tous les liens symboliques créés par `symlink-manager` à l'activation précédente. En plus, les sous-répertoires qui deviennent vides seront aussi nettoyés.
3. Créer de nouveaux liens symboliques de cette façon : il cherche dans le répertoire `files/` (normalement défini avec `home-files-service-type`, `home-xdg-configuration-files-service-type` et éventuellement d'autres), prend les fichiers du sous-répertoire `files/.config/` et place des liens symboliques correspondant dans `XDG_CONFIG_DIR`. Par exemple le lien symbolique pour `files/.config/sway/config` se retrouvera dans `$XDG_CONFIG_DIR/sway/config`. Le reste des fichiers de `files/` en dehors du sous-répertoire `files/.config/` seront traités un peu différemment : un lien symbolique sera créé dans `$HOME`. `files/.un-programme/config` se retrouvera dans `$HOME/.un-programme/config`.
4. Si certains sous-répertoires sont absents, ils seront créés.
5. Si des fichiers sont en conflit, ils seront sauvegardés.

`symlink-manager` fait partie des services personnels essentiels et est activé et utilisé par défaut.

### 13.3.2 Shells

Les shells jouent un rôle important dans le processus d'initialisation de l'environnement, et vous pouvez les configurer manuellement comme décrit dans la section Section 13.2 [Configurer le shell], page 621, mais il est recommandé d'utiliser les services personnels listés plus bas. C'est à la fois plus facile et plus fiable.

Chaque environnement personnel instancie `home-shell-profile-service-type`, qui crée un fichier de démarrage `~/.profile` pour tous les shell POSIX. Ce fichier contient toutes les étapes nécessaires à l'initialisation de l'environnement, mais de nombreux shells modernes comme Bash ou Zsh préfèrent leur propres fichiers de démarrage, c'est pourquoi les services personnels correspondants (`home-bash-service-type` et `home-zsh-service-type`) s'assurent que `~/.profile` est sourcé par `~/.bash_profile` et `~/.zprofile`, respectivement.

### Service de profil du shell

**home-shell-profile-configuration** [Type de données]

Les champs de `home-shell-profile-configuration` disponibles sont :

**profile** (par défaut : ()) (type : text-config)

`home-shell-profile` est instancié automatiquement par `home-environment`, NE créez PAS ce service manuellement, il ne peut qu'être étendu. `profile` est une liste d'objets simili-fichiers, qui iront dans `~/.profile`. Par défaut `~/.profile` contient le code d'initialisation, qui doit être évalué par le shell de connexion pour rendre le profil de l'environnement personnel disponible, mais vous pouvez ajouter d'autres commandes au fichier si c'est vraiment nécessaire. Dans la plupart des cas les personnalisations devraient aller dans les fichiers de configuration de votre shell. Étendez le service `home-shell-profile` qui si vous savez vraiment ce que vous faites.

## Services personnel Bash

**home-bash-configuration** [Type de données]

Les champs de `home-bash-configuration` disponibles sont :

**package** (par défaut : `bash`) (type : paquet)

Le paquet Bash à utiliser.

**guix-defaults?** (par défaut : `#t`) (type : booléen)

Ajoute des instructions par défaut correctes comme la lecture de `/etc/bashrc` et la coloration de la sortie de `ls` au début du fichier `.bashrc`.

**environment-variables** (par défaut : ()) (type : liste d'association)

Liste d'association de variables d'environnement à initialiser dans la session Bash. Les règles de `home-environment-variables-service-type` s'appliquent ici (voir Section 13.3.1 [Services personnels essentiels], page 622). Le contenu de ce champ sera ajouté après le contenu du champ `bash-profile`.

**aliases** (par défaut : ()) (type : liste d'association)

Liste d'association d'alias à initialiser pour la session Bash. Les alias seront définis après le contenu du champ `bashrc` dans le fichier `.bashrc`. Les alias seront automatiquement encadrés de guillemets, pour que ceci :

```
'(("ls" . "ls -a1F"))
```

devienne

```
alias ls="ls -a1F"
```

**bash-profile** (par défaut : ()) (type : text-config)

Liste d'objets simili-fichiers qui seront ajoutés à `.bash_profile`. C'est utilisé pour exécuter des commandes au démarrage d'un shell de connexion (dans la plupart des cas le shell qui démarre sur un tty après la connexion). `.bash_login` ne sera jamais lu parce que `.bash_profile` est toujours présent.

**bashrc** (par défaut : ()) (type : text-config)

Liste d'objets simili-fichiers qui seront ajoutés à `.bashrc`. C'est utilisé pour exécuter des commandes au lancement d'un shell interactif (le shell

utilisé de manière interactive, démarré en tapant **bash** ou par votre application de terminal ou tout autre programme).

**bash-logout** (par défaut : ()) (type : text-config)

Liste d'objets simili-fichiers qui seront ajoutés à **.bash\_logout**. C'est utilisé pour exécuter des commandes à la sortie d'un shell de connexion. Il ne sera pas lu dans certains cas (si le shell termine en utilisant **exec** sur un autre processus par exemple).

Vous pouvez étendre le service Bash en utilisant un enregistrement de configuration **home-bash-extension**, dont les champs correspondent à ceux de **home-bash-configuration** (voir [home-bash-configuration], page 626). Le contenu des extensions seront ajoutées à la fin des fichiers de configuration Bash correspondants (voir Section "Bash Startup Files" dans *le manuel de référence de Bash*).

Par exemple, voici comment définir un service qui étend le service Bash de sorte que **~/.bash\_profile** définisse une variable d'environnement supplémentaire, **PS1** :

```
(define bash-fancy-prompt-service
  (simple-service 'bash-fancy-prompt
    home-bash-service-type
    (home-bash-extension
      (environment-variables
        '(("PS1" . "\\u \\wλ "))))))
```

Vous ajouterez ensuite **bash-fancy-prompt-service** à la liste dans le champ **services** de votre **home-environment**. Voici la documentation de référence pour **home-bash-extension**.

**home-bash-extension** [Type de données]

Les champs de **home-bash-extension** disponibles sont :

**environment-variables** (par défaut : ()) (type : liste d'association)

Variables d'environnement supplémentaires à définir. Elles seront combinées aux variables d'environnement des autres extensions et du service de base pour former un bloc cohérent de variables d'environnement.

**aliases** (par défaut : ()) (type : liste d'association)

Alias supplémentaires à définir. Ils seront combinés aux alias des autres extensions et du service de base.

**bash-profile** (par défaut : ()) (type : text-config)

Blocs de texte supplémentaires à ajouter à **.bash\_profile**, qui seront combinés avec les blocs de texte des autres extensions et du service de base.

**bashrc** (par défaut : ()) (type : text-config)

Blocs de texte supplémentaires à ajouter à **.bashrc**, qui seront combinés avec les blocs de texte des autres extensions et du service de base.

**bash-logout** (par défaut : ()) (type : text-config)

Blocs de texte supplémentaires à ajouter à **.bash\_logout**, qui seront combinés avec les blocs de texte des autres extensions et du service de base.

## Services personnel Zsh

`home-zsh-configuration` [Type de données]

Les champs de `home-zsh-configuration` disponibles sont :

`package` (par défaut : `zsh`) (type : paquet)

Le paquet Zsh à utiliser.

`xdg-flavor?` (par défaut : `#t`) (type : booléen)

Place tous les fichiers de configuration dans `$XDG_CONFIG_HOME/zsh`. Cela fait initialiser `ZDOTDIR` à `$XDG_CONFIG_HOME/zsh` dans `~/.zshenv`. Le processus de démarrage du shell continuera avec `$XDG_CONFIG_HOME/zsh/.zshenv`.

`environment-variables` (par défaut : `()`) (type : liste d'association)

Une liste d'association de variables d'environnement à initialiser dans la session Zsh.

`zshenv` (par défaut : `()`) (type : text-config)

Liste d'objets simili-fichiers qui seront ajoutés à `.zshenv`. C'est utilisé pour initialiser les variables d'environnement du shell. Elle ne doit pas contenir de commande qui suppose la présence d'un tty ou qui produise une sortie. Cela sera toujours lu, et toujours avant tout autre fichiers dans `ZDOTDIR`.

`zprofile` (par défaut : `()`) (type : text-config)

Liste d'objets simili-fichiers qui seront ajoutés à `.zprofile`. C'est utilisé pour exécuter des commandes au démarrage d'un shell de connexion (dans la plupart des cas le shell démarré sur un tty après la connexion). Elle sera lue avant `.zlogin`.

`zshrc` (par défaut : `()`) (type : text-config)

Liste d'objets simili-fichiers qui seront ajoutés à `.zshrc`. C'est utilisé pour exécuter des commandes au démarrage d'un shell interactif (le shell utilisé de manière interactive démarré en tapant `zsh` ou par votre application de terminal ou toute autre application).

`zlogin` (par défaut : `()`) (type : text-config)

Liste d'objets simili-fichiers qui seront ajoutés à `.zlogin`. C'est utilisé pour exécuter des commandes à la fin du processus de démarrage du shell de connexion.

`zlogout` (par défaut : `()`) (type : text-config)

Liste d'objets simili-fichiers qui seront ajoutés à `.zlogout`. C'est utilisé pour exécuter des commandes à la sortie d'un shell de connexion. Elle ne sera pas lue dans certains cas (si le shell termine après avoir utilisé `exec` sur un autre processus par exemple).

### 13.3.3 Exécution de tâches planifiées personnelles

Le module (`gnu home services mcron`) fournit une interface pour GNU `mcron`, un démon qui lance des tâches planifiées (voir *GNU mcron*). Les informations sur le `mcron` du système

s'appliquent ici aussi (voir Section 12.9.2 [Exécution de tâches planifiées], page 287), la seule différence pour les services personnels et qu'ils doivent être déclarés dans un enregistrement `home-environment` au lieu d'un enregistrement `operating-system`.

**home-mcron-service-type** [Variable Scheme]

C'est le type du service personnel `mcron`, dont la valeur est un objet `home-mcron-configuration`. Il permet de gérer des tâches planifiées.

Ce type de service peut être la cible d'une extension de service qui fournit des spécifications de tâches supplémentaires (voir Section 12.18.1 [Composition de services], page 599). En d'autres termes, il est possible de définir des services qui fournissent des tâches `mcron` à lancer.

**home-mcron-configuration** [Type de données]

Les champs de `home-mcron-configuration` disponibles sont :

`mcron` (par défaut : `mcron`) (type : simili-fichier)

Le paquet `mcron` à utiliser.

`jobs` (par défaut : `()`) (type : liste-de-gexps)

C'est la liste des gexps (voir Section 9.12 [G-Expressions], page 166), où chaque gexp correspond à une spécification de tâche de `mcron` (voir Section "Syntax" dans *GNU mcron*).

`log?` (par défaut : `#t`) (type : booléen)

Journalise les messages sur la sortie standard.

`log-format` (par défaut : `"~1@*~a ~a: ~a~%"`) (type : chaîne)

Chaîne de format (`ice-9 format`) pour les message de journaux. La valeur par défaut produit des messages de type « `'pid name: message'` » (voir Section "Invoking `mcron`" dans *GNU mcron*). Chaque message est aussi préfixé par un horodatage par le GNU Shepherd.

### 13.3.4 Services personnels de gestion de l'énergie

Le module (`gnu home services pm`) fournit des services personnels pour la gestion de la batterie.

**home-batsignal-service-type** [Variable Scheme]

Service pour `batsignal`, un programme un surveille le niveau de batterie et vous averti à travers des notifications du bureau lorsque la batterie est presque vide. Vous pouvez aussi configurer une commande à lancer lorsque le niveau de batterie descend en dessous d'un seuil « dangereux ». Ce service est configuré avec l'enregistrement `home-batsignal-configuration`.

**home-batsignal-configuration** [Type de données]

Type données qui représente la configuration de `batsignal`.

`warning-level` (par défaut : `15`)

Le niveau de batterie auquel envoyer un message d'avertissement.

`warning-message` (par défaut : `#f`)

Le message à envoyer dans une notification lorsque le niveau de batterie atteint le `warning-level`. Indiquer `#f` utilise le message par défaut.



- critical-level** (par défaut : 5)  
Le niveau de batterie auquel envoyer un message critique.
- critical-message** (par défaut : #f)  
Le message à envoyer dans une notification lorsque le niveau de batterie atteint le **critical-level**. Indiquer #f utilise le message par défaut.
- danger-level** (par défaut : 2)  
Le niveau de batterie auquel lancer la **danger-command**.
- danger-command** (par défaut : #f)  
La commande à lancer lorsque le niveau de batterie atteint le **danger-level**. Indiquer #f désactive complètement l'exécution de la commande.
- full-level** (par défaut : #f)  
Le niveau de batterie auquel envoyer le message de plein. Indiquer #f désactive l'envoi du message de plein.
- full-message** (par défaut : #f)  
Le message à envoyer dans une notification lorsque le niveau de batterie atteint le **full-level**. Indiquer #f utiliser le message par défaut.
- batteries** (par défaut : '() )  
Les batteries à surveiller. Indiquer '() essaye de trouver les batteries automatiquement.
- poll-delay** (par défaut : 60)  
Le temps d'attente en secondes avant de revérifier la batterie.
- icon** (par défaut : #f)  
Un objet simili-fichier à utiliser comme icône pour les notifications de la batterie. Indiquer #f désactive complètement l'icône de notification.
- notifications?** (par défaut : #t)  
S'il faut envoyer des notifications.
- notifications-expire?** (par défaut : #f)  
Indique si les notifications envoyées expirent après un certain temps.
- notification-command** (par défaut : #f)  
Commande à utiliser pour envoyer les messages. Indiquer #f envoie une notification à travers **libnotify**.
- ignore-missing?** (par défaut : #f)  
Indique s'il faut ignorer les erreurs de batterie manquante.

### 13.3.5 Gérer les démons personnels

Le module (`gnu home services shepherd`) prend en charge la définition de services Shepherd par utilisateur (voir Section "Introduction" dans *le manuel de GNU Shepherd*). Vous pouvez étendre `home-shepherd-service-type` avec de nouveaux services ; Guix Home se chargera ensuite de démarrer le démon `shepherd` pour vous à la connexion, ce qui démarrera ensuite les services que vous avez demandés.

**home-shepherd-service-type** [Variable Scheme]

Le type de service pour le shepherd en espace utilisateur, qui vous permet de gérer les processus longs ou les tâches uniques. Le shepherd personnel n'est pas un processus d'init (PID 1), mais presque toutes les informations décrites dans Section 12.18.4 [Services Shepherd], page 607, sont applicables ici aussi.

C'est le type de service que les extensions ciblent lorsqu'elles veulent créer un service shepherd (voir Section 12.18.2 [Types service et services], page 600, pour un exemple). Chaque extension doit passer une liste de `<shepherd-service>`. Sa valeur doit être un `home-shepherd-configuration`, décrit plus bas.

**home-shepherd-configuration** [Type de données]

Ce type de données représente la configuration du Shepherd.

**shepherd** (par défaut : `shepherd`)

Le paquet Shepherd à utiliser.

**auto-start?** (par défaut : `#t`)

Indique s'il faut lancer le Shepherd à la première connexion.

**services** (par défaut : `'()`)

Une liste de `<shepherd-service>` à démarrer. Vous devriez probablement utiliser le mécanisme d'extensions des services à la place (voir Section 12.18.4 [Services Shepherd], page 607).

### 13.3.6 Shell sécurisé

Le paquet OpenSSH (<https://www.openssh.com>) contient un client, la commande `ssh`, qui vous permet de vous connecter à des machines distantes avec le protocole SSH (shell sécurisé). Avec le module (`gnu home services ssh`), vous pouvez configurer OpenSSH pour qu'il fonctionne de manière prédictible, presque indépendamment de l'état de la machine locale. Pour cela, vous devez instancier `home-openssh-service-type` dans votre configuration personnelle, comme expliqué plus bas.

**home-openssh-service-type** [Variable Scheme]

C'est le type de service pour configurer le client OpenSSH. Il a les fonctionnalités suivantes :

- fournir un fichier `~/.ssh/config` basé sur votre configuration pour que `ssh` connaisse les hôtes auxquels vous vous connectez régulièrement et leurs paramètres associés ;
- fournir un `~/.ssh/authorized_keys`, qui liste les clés publiques que le serveur SSH local, `sshd`, peut accepter pour se connecter à ce compte utilisateur ;
- éventuellement fournir un fichier `~/.ssh/known_hosts` pour que `ssh` puisse authentifier les hôtes auxquels vous vous connectez.

Voici un exemple de service et sa configuration que vous pouvez ajouter au champ `services` de votre `home-environment` :

```
(service home-openssh-service-type
  (home-openssh-configuration
    (hosts
```

```
(list (openssh-host (name "ci.guix.gnu.org")
                    (user "charlie"))
      (openssh-host (name "chbouib")
                    (host-name "chbouib.example.org")
                    (user "supercharlie")
                    (port 10022)))
      (authorized-keys (list (local-file "alice.pub"))))
```

L'exemple au-dessus liste deux hôtes et leurs paramètres. Par exemple, lancer `ssh chbouib` se connectera automatiquement à `chbouib.example.org` sur le port 10022, en se connectant en tant que 'supercharlie'. En plus, il marque la clé publique dans `alice.pub` comme autorisée pour les connexions entrantes.

La valeur associée à une instance de `home-openssh-service-type` doit être un enregistrement `home-openssh-configuration`, décrit ci-dessous.

`home-openssh-configuration` [Type de données]

C'est le type de données représentant la configuration du client et du serveur OpenSSH dans l'environnement personnel. Il possède les champs suivants :

`hosts` (par défaut : '() )

Une liste d'enregistrements `openssh-host` qui spécifient les noms d'hôte et les paramètres de connexion associés (voir plus bas). Cette liste d'hôtes va dans `~/.ssh/config`, que `ssh` lit au démarrage.

`known-hosts` (par défaut : `*unspecified*`)

Cela doit être soit :

- `*unspecified*`, auquel cas `home-openssh-service-type` laisse à `ssh` et à l'utilisateur le soin de maintenir la liste des hôtes connus dans `~/.ssh/known_hosts`, ou
- une liste d'objets simili-fichiers, auquel cas ils sont concaténés et émis dans `~/.ssh/known_hosts`.

Le fichier `~/.ssh/known_hosts` contient une liste de paires de nom d'hôte et de clé d'hôte qui permet à `ssh` d'authentifier les hôtes auxquels vous vous connectez et de détecter de potentielles attaques par usurpation. Par défaut, `ssh` la met à jour sur la base du principe *TOFU*, *confiance à la première utilisation*, ce qui signifie qu'il enregistre la clé d'hôte dans ce fichier la première fois que vous vous connectez. Ce comportement est préservé si `known-hosts` est indiqué à `*unspecified*`.

Si vous fournissez à la place une liste de clés d'hôtes dès le départ dans le champ `known-hosts`, votre configuration devient auto-contenue et sans état : elle peut être répliquée ailleurs ou à un autre moment. Préparer cette liste peut être plutôt compliqué cependant, c'est pourquoi `*unspecified*` reste la valeur par défaut.

`authorized-keys` (par défaut : '() )

Cela doit être une liste d'objets simili-fichiers, chacun contenant une clé publique SSH qui devrait être autorisée à se connecter à cette machine.

Concrètement, ces fichiers sont concaténés et rendus disponibles dans `~/.ssh/authorized_keys`. Si un serveur OpenSSH, `sshd`, est lancé sur

cette machine, alors il *peut* prendre ce fichier en compte : c'est ce que `sshd` fait par défaut, mais soyez conscient qu'il peut aussi être configuré pour l'ignorer.

`openssh-host` [Type de données]

Les champs de `openssh-host` disponibles sont :

`name` (type : string)

Nom de cette déclaration d'hôte.

`host-name` (type : peut-être-chaine)

Nom d'hôte — p. ex. `"toto.exemple.org"` ou `"192.168.1.2"`.

`address-family` (type : address-family)

Famille d'adresse à utiliser pour se connecter à l'hôte : soit `AF_INET` (pour IPv4 uniquement), `AF_INET6` (pour IPv6 uniquement), ou `*unspecified*` (pour permettre n'importe quelle famille d'adresse).

`identity-file` (type : peut-être-chaine)

Le fichier d'identité à utiliser — p. ex. `"/home/charlie/.ssh/id_ed25519"`.

`port` (type : peut-être-entier-naturel)

Numéro de port TCP sur lequel se connecter.

`user` (type : peut-être-chaine)

Le nom d'utilisateur sur la machine distante.

`forward-x11?` (par défaut : `#f`) (type : booléen)

Indique s'il faut relayer les connexions clientes distantes vers l'affichage graphique X11 local.

`forward-x11-trusted?` (par défaut : `#f`) (type : booléen)

Indique si les client X11 distants ont l'accès complet à l'affichage graphique X11 original.

`forward-agent?` (par défaut : `#f`) (type : booléen)

Indique si l'agent d'authentification (s'il y en a un) est relayé vers la machine distante.

`compression?` (par défaut : `#f`) (type : booléen)

Indique s'il faut compresser les données en transit.

`proxy-command` (type : peut-être-chaine)

La commande à utiliser pour se connecter au serveur. Par exemple, une commande pour se connecter via un mandataire HTTP sur 192.0.2.0 serait : `"nc -X connect -x 192.0.2.0:8080 %h %p"`.

`host-key-algorithms` (type : peut-être-liste-de-chaine)

La liste des algorithmes de clé hôtes acceptées — p. ex. `'("ssh-ed25519")`.

`accepted-key-types` (type : peut-être-liste-de-chaine)

La liste des types de clés publiques utilisatrices acceptées.

**extra-content** (par défaut : "") (type : raw-configuration-string)  
Contenu supplémentaire ajouté tel-quel à la fin de ce block Host dans  
~/.ssh/config.

### 13.3.7 Services personnels pour ordinateur de bureau

Le module (`gnu home services desktop`) fournit des services que vous trouverez pratique sur un système « de bureau » possédant un environnement utilisateur graphique comme Xorg.

**home-redshift-service-type** [Variable Scheme]

Il s'agit du type de service pour Redshift (<https://github.com/jonls/redshift>), un programme qui ajuste la température des couleurs de l'affichage suivant l'heure de la journée. Sa valeur associée doit être un `home-redshift-configuration` comme indiqué ci-dessous.

Une configuration typique, où nous spécifions la latitude et la longitude manuellement, ressemblerait à ceci :

```
(service home-redshift-service-type
  (home-redshift-configuration
    (location-provider 'manual)
    (latitude 35.81)    ;hémisphère nord
    (longitude -0.80))) ;ouest de Greenwich
```

**home-redshift-configuration** [Type de données]

Les champs de `home-redshift-configuration` disponibles sont :

**redshift** (par défaut : `redshift`) (type : simili-fichier)

Le paquet Redshift à utiliser.

**location-provider** (par défaut : `geoclue2`) (type : symbol)

Fournisseur de géolocalisation — `'manual` ou `'geoclue2`. Dans le premier cas, vous devez aussi spécifier les champs `latitude` et `longitude` pour que Redshift puisse déterminer les heures de jour et de nuit là où vous vous trouvez. Dans le deuxième cas, le service système Geoclue doit être lancé ; Redshift lui demandera les informations de localisation.

**adjustment-method** (par défaut : `randr`) (type : symbol)

Méthode d'ajustement des couleurs.

**daytime-temperature** (par défaut : `6500`) (type : integer)

Température des couleurs pour le jour (en kelvins).

**nighttime-temperature** (par défaut : `4500`) (type : entier)

Température des couleurs pour la nuit (en kelvins).

**daytime-brightness** (type : peut-être-nombre-inexact)

Luminosité de l'écran en journée, entre 0.1 et 1.0, ou non spécifiée.

**nighttime-brightness** (type : peut-être-nombre-inexact)

Luminosité de l'écran la nuit, entre 0.1 et 1.0, ou non spécifiée.

**latitude** (type : peut-être-nombre-inexact)

Latitude, lorsque `location-provider` est `'manual`.

- longitude** (type : peut-être-nombre-inexact)  
Longitude, lorsque `location-provider` est `'manual'`.
- dawn-time** (type : peut-être-chaine)  
Heure personnalisée de transition entre la nuit et le jour le matin — au format `"HH:MM"`. Lorsque la valeur est spécifiée, l'élévation du soleil n'est pas utilisée pour déterminer le cycle jour-nuit.
- dusk-time** (type : peut-être-chaine)  
De même, heure personnalisée pour la transition entre le jour et la nuit en soirée.
- extra-content** (par défaut : `"`) (type : raw-configuration-string)  
Contenu supplémentaire ajouté tel quel à la fin du fichier de configuration de Redshift. Lancez `man redshift` pour plus d'informations sur le format du fichier de configuration.

**home-dbus-service-type** [Variable Scheme]  
C'est le type de service pour lancer un D-Bus spécifique à la session, pour les applications non privilégiées qui nécessitent que D-Bus soit lancé.

**home-dbus-configuration** [Type de données]  
L'enregistrement de configuration pour `home-dbus-service-type`.

**dbus** (par défaut : `dbus`)  
Le paquet qui fournit la commande `/bin/dbus-daemon`.

### 13.3.8 Services personnels Guix

Le module (`gnu home services guix`) fournit des services pour la configuration de Guix de l'utilisateur ou utilisatrice.

**home-channels-service-type** [Variable Scheme]  
C'est le type de service qui gère `$XDG_CONFIG_HOME/guix/channels.scm`, le fichier qui contrôle les canaux reçus par `guix pull` (voir Chapitre 7 [Canaux], page 75). Sa valeur associée est une liste d'enregistrements `channel`, définis dans le module (`guix channels`).

En général il vaut mieux étendre ce service que de le configurer directement, car sa valeur par défaut est le canal `guix` par défaut défini par `%default-channels`. Si vous configurez ce service directement, assurez-vous d'inclure un canal `guix`. Voir Section 7.1 [Spécifier des canaux supplémentaires], page 75, et Section 7.2 [Utiliser un canal Guix personnalisé], page 75, pour plus de détails.

Une extension typique pour ajouter un canal ressemblerait à ceci :

```
(simple-service 'variant-packages-service
  home-channels-service-type
  (list
    (channel
      (name 'variant-packages)
      (url "https://example.org/variant-packages.git")))))
```

## 13.4 Invoquer guix home

Une fois que vous avez écrit une déclaration d'environnement personnel (voir Section 13.1 [Déclarer l'environnement personnel], page 619), elle peut être *instanciée* avec la commande `guix home`. Voici le résumé de la commande :

```
guix home options... action file
```

*file* doit être le nom d'un fichier contenant une déclaration `home-environment`. *action* spécifie comment l'environnement personnel est instancié, mais il y a quelques actions supplémentaires qui ne l'instancient pas. Actuellement les valeurs suivantes sont prises en charge :

`search` Affiche les définitions des types de services personnels disponibles qui correspondent aux expressions régulières données, triées par pertinence :

```
$ 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

...
```

Comme pour `guix search`, le résultat est écrit au format `recutils`, ce qui rend facile le filtrage de la sortie (voir *GNU recutils manual*).

`conteneur`

Démarré un shell dans un environnement isolé — un *conteneur* — contenant votre dossier personnel spécifié par *file*.

Par exemple, voici comment vous pouvez démarrer un shell interactif dans un conteneur avec votre dossier personnel :

```
guix home container config.scm
```

C'est un conteneur jetable où vous pouvez gaiement jouer avec vos fichiers ; les changements effectués dans le conteneur et tous les processus qui y sont démarrés disparaissent aussitôt que vous sortez de ce shell.

Comme pour `guix shell`, plusieurs options contrôlent ce conteneur :

```
--network
```

`-N` Active le réseau dans le conteneur (il est désactivé par défaut).

```
--expose=source[=cible]
```

```
--share=source[=cible]
```

Comme pour `guix shell`, rend le répertoire *source* du système hôte disponible en tant que *target* dans le conteneur — en lecture-seule si vous utilisez `--expose`, et inscriptible si vous utilisez `--share` (voir Section 8.1 [Invoquer `guix shell`], page 83).

En plus, vous pouvez lancer une commande dans ce conteneur, au lieu de démarrer un shell interactif. Par exemple, voici comment vérifier que les service Shepherd sont démarrés dans un conteneur personnel jetable :

```
guix home container config.scm -- herd status
```

La commande à lancer dans le conteneur doit être indiquée après `--` (double tirets).

**edit** Modifier ou visualiser la définition des types de services personnels donnés.

Par exemple, la commande plus bas ouvre votre éditeur, spécifié par la variable d'environnement `EDITOR`, sur la définition du type de service `home-mcron` :

```
guix home edit home-mcron
```

**reconfigure**

Construit l'environnement personnel décrit dans *file* et y bascule. Le basculement signifie que le script d'activation sera évalué et (dans le scénario de base) les liens symboliques vers les fichiers de configuration générés dans la déclaration `home-environment` seront créés dans `~`. Si un fichier avec le même chemin existe dans le répertoire personnel il sera déplacé vers `~/horodatage-guix-home-legacy-confis-backup`, où *horodatage* est l'époque UNIX actuelle.

**Remarque:** Il est grandement recommandé de lancer `guix pull` une fois avant de lancer `guix home reconfigure` pour la première fois (voir Section 6.6 [Invoquer `guix pull`], page 63).

Cela met en application toute la configuration spécifiée dans *file*. La commande démarre les services Shepherd spécifiés dans *file* qui ne sont pas actuellement lancés ; si un service est actuellement exécuté cette commande s'arrange pour qu'il soit mis à jour la prochaine fois qu'il est stoppé (p. ex par `herd stop service` ou `herd restart service`).

Cette commande crée une nouvelle génération dont le numéro est un de plus que la génération actuelle (rapportée par `guix home list-generations`). Si cette génération existe déjà, elle sera réécrite. Ce comportement correspond à celui de `guix package` (voir Section 6.2 [Invoquer `guix package`], page 44).



À la fin, le nouveau dossier personnel est déployé dans `~/guix-home`. Ce répertoire contient les *métadonnées de provenance* : la liste des canaux utilisés (voir Chapitre 7 [Canaux], page 75) et le *fichier* lui-même, s'il est disponible. Vous pouvez voir les informations de provenance avec :

```
guix home describe
```

Cette information est utile si vous voulez plus tard inspecter comment une génération particulière a été construite. En fait, en supposant que *file* est auto-contenu, vous pouvez reconstruire la génération *n* de votre environnement personnel avec :

```
guix time-machine \
  -C /var/guix/profiles/per-user/USER/guix-home-n-link/channels.scm -- \
  home reconfigure \
  /var/guix/profiles/per-user/USER/guix-home-n-link/configuration.scm
```

Vous pouvez considérer cela comme une sorte de contrôle de version intégré ! Votre dossier personnel n'est pas seulement un artefact binaire : *il contient ses propres sources*.

#### switch-generation

Bascule à une génération du dossier personnel existante. Cette action bascule de manière atomique à la génération du dossier personnel spécifiée.

La génération cible peut être spécifiée explicitement par son numéro de génération. Par exemple, l'invocation suivante passerait à la génération 7 du dossier personnel :

```
guix home switch-generation 7
```

La génération cible peut aussi être spécifiée relativement à la génération actuelle avec la forme `+N` ou `-N`, où `+3` signifie « trois générations après la génération actuelle » et `-1` signifie « une génération précédent la génération actuelle ». Lorsque vous spécifiez un nombre négatif comme `-1`, il doit être précédé de `--` pour éviter qu'il ne soit compris comme une option. Par exemple :

```
guix home switch-generation -- -1
```

Cette action échouera si la génération spécifiée n'existe pas.

#### roll-back

Passe à la génération précédente du dossier personnel. C'est le contraire de `reconfigure`, et c'est exactement comme invoquer `switch-generation` avec pour argument `-1`.

#### delete-generations

Supprimer des générations du dossier personnel, ce qui les rend disponibles pour le ramasse-miettes (voir Section 6.5 [Invoquer guix gc], page 60, pour des informations sur la manière de lancer le « ramasse-miettes »).

Cela fonctionne comme pour `'guix package --delete-generations'` (voir Section 6.2 [Invoquer guix package], page 44). Avec aucun argument, toutes les générations du dossier personnel sauf la génération actuelle sont supprimées :

```
guix home delete-generations
```

Vous pouvez aussi choisir les générations que vous voulez supprimer. L'exemple plus bas supprime toutes les génération du dossier personnel plus vieilles que deux mois :

```
guix home delete-generations 2m
```

**build** Construit la dérivation de l'environnement personnel, ce qui comprend tous les fichiers de configuration et les programmes requis. Cette action n'installe rien.

**describe** Décrit la génération du dossier personnel actuel : son nom de fichier, ainsi que les informations de provenance si elles sont disponibles.

Pour afficher les paquets installés dans le profil de la génération actuelle du dossier personnel, le drapeau `--list-installed` est proposé, avec la même syntaxe que celle utilisée pour `guix package --list-installed` (voir Section 6.2 [Invoquer `guix package`], page 44). Par exemple, la commande suivante affiche un tableau de tous les paquets avec « emacs » dans leur nom qui sont installés dans la génération actuelle du profil du dossier personnel :

```
guix home describe --list-installed=emacs
```

**list-generations**

Affiche un résumé de chaque génération de l'environnement personnel disponible sur le disque, dans un format lisible pour un humain. C'est similaire à l'option `--list-generations` de `guix package` (voir Section 6.2 [Invoquer `guix package`], page 44).

Éventuellement, on peut spécifier un motif, avec la même syntaxe utilisée pour `guix package --list-generations`, pour restreindre la liste des générations affichées. Par exemple, la commande suivante affiche les générations de moins de 10 jours :

```
guix home list-generations 10d
```

Le drapeau `--list-installed` peut aussi être spécifié, avec la même syntaxe que celle utilisée dans `guix home describe`. Cela peut être utile si vous essayez de déterminer si un paquet a été ajouté au profil personnel.

**import** Génère un *environnement personnel* à partir des paquets du profil par défaut et des fichiers de configuration qui se trouvent dans votre répertoire personnel. Les fichiers de configuration sont copiés vers le répertoire donné, et un fichier `home-configuration.scm` sera rempli avec l'environnement personnel. Remarquez que les services personnels existants ne sont pas tous pris en charge (voir Section 13.3 [Services du dossier personnel], page 622).

```
$ guix home import ~/guix-config
```

```
guix home: « /home/alice/guix-config » rempli avec tous les fichiers de conf
```

Et plus encore ! `guix home` fournit aussi les sous-commandes suivantes pour visualiser la manière dont les services de votre environnement personnel sont liés les uns aux autres :

**extension-graph**

Émettre vers la sortie standard le *graphe d'extension de service* de l'environnement personnel défini dans *file* (voir Section 12.18.1 [Composition de services], page 599, pour plus d'informations sur les extensions de service). Par défaut, le format de sortie est le format Dot/Graphviz, mais vous pouvez

choisir un autre format avec `--graph-backend`, comme avec `guix graph` (voir Section 10.10 [Invoquer `guix graph`], page 218) :

La commande :

```
guix home extension-graph file | xdot -
```

montre les relations d'extension entre les services.

#### `shepherd-graph`

Affiche le *graphe de dépendance* des services `shepherd` de l'environnement personnel défini dans *file* sur la sortie standard. Voir Section 12.18.4 [Services Shepherd], page 607, pour plus d'informations et un exemple de graphe.

Encore une fois, le format de sortie par défaut est `Dot/Graphviz`, mais vous pouvez passer `--graph-backend` pour en choisir un autre.

*options* peut contenir n'importe quelle option commune de construction (voir Section 10.1.1 [Options de construction communes], page 180). En plus, *options* peut contenir l'une de ces options :

`--expression=expr`

`-e expr` Considère l'environnement personnel en lequel s'évalue *expr*. C'est une alternative à la spécification d'un fichier qui s'évalue en un environnement personnel.

`--allow-downgrades`

Dit à `guix home reconfigure` de permettre le retour en arrière du système.

Comme pour `guix system`, `guix home reconfigure`, par défaut, vous empêche de faire revenir votre dossier personnel à une révision plus vieille ou non liée par rapport aux révisions des canaux utilisés pour le déployer — cette information est affichée par `guix home describe`. Avec `--allow-downgrades`, vous pouvez outrepasser cette vérification, au risque de faire revenir votre dossier personnel en arrière — faites attention !

## 14 Documentation

Dans la plupart des cas les paquets installés avec Guix ont une documentation. Il y a deux formats de documentation principaux : « Info », un format hypertexte navigable utilisé par les logiciels GNU et les « pages de manuel » (ou « pages de man »), le format de documentation linéaire traditionnel chez Unix. Les manuels Info sont disponibles via la commande `info` ou avec Emacs, et les pages de man sont accessibles via la commande `man`.

Vous pouvez chercher de la documentation pour les logiciels installés sur votre système par mot-clef. Par exemple, la commande suivante recherche des informations sur « TLS » dans les manuels Info :

```
$ info -k TLS
"(emacs)Network Security" -- STARTTLS
"(emacs)Network Security" -- TLS
"(gnutls)Core TLS API" -- gnutls_certificate_set_verify_flags
"(gnutls)Core TLS API" -- gnutls_certificate_set_verify_function
...
```

La commande ci-dessous recherche le même mot-clé dans les pages de manuel<sup>1</sup> :

```
$ man -k TLS
SSL (7)           - OpenSSL SSL/TLS library
certtool (1)     - GnuTLS certificate tool
...
```

Ces recherches sont purement locales à votre ordinateur donc vous savez que la documentation trouvée correspond à ce qui est effectivement installé, vous pouvez y accéder hors ligne et votre vie privée est préservée.

Une fois que vous avez ces résultats, vous pouvez visualiser la documentation appropriée avec, disons :

```
$ info "(gnutls)Core TLS API"
```

ou :

```
$ man certtool
```

Les manuels Info contiennent des sections et des index ainsi que des hyperliens comme ce qu'on trouve sur les pages Web. Le lecteur `info` (voir *Stand-alone GNU Info*) et sa contrepartie dans Emacs (voir Section “Misc Help” dans *The GNU Emacs Manual*) fournissent des raccourcis claviers intuitifs pour naviguer dans les manuels Voir Section “Getting Started” dans *Info: An Introduction* pour trouver une introduction sur la navigation dans info.

---

<sup>1</sup> La base de donnée parcourue par `man -k` est uniquement créée dans les profils qui contiennent le paquet `man-db`.

## 15 Plateformes

Les paquets et les systèmes construits par Guix sont conçus, comme la plupart des programmes informatiques, pour tourner sur un CPU avec un jeu d'instructions spécifique, et sous un système d'exploitation particulier. Ces programmes ciblent le plus souvent aussi un noyau et une bibliothèque système particulière. Ces contraintes sont reflétées par Guix dans les enregistrements `platform`.

### 15.1 Référence de `platform`

Le type de données `platform` décrit une *plateforme* : un ISA (jeu d'instructions architectural (en anglais *instruction set architecture*)) combiné avec un système d'exploitation et éventuellement des paramètres systèmes supplémentaires comme l'ABI (interface binaire applicative (en anglais *application binary interface*)).

`platform` [Type de données]

C'est le type de donnée représentant une plateforme.

`target` Ce champ spécifie le triplet GNU de la plateforme en une chaîne (voir Section "Specifying Target Triplets" dans *Autoconf*).

`system` Cette chaîne est le type de système connu par Guix et passé, par exemple, à l'option `--system` de la plupart des commandes.

Elle a le plus souvent la forme "`cpu-noyau`", où `cpu` est le CPU cible et `noyau` est le noyau du système d'exploitation cible.

Elle peut être par exemple "`aarch64-linux`" ou "`armhf-linux`". Vous rencontrerez ces types de systèmes en effectuant des constructions natives (voir Section 11.2 [Constructions natives], page 236).

`linux-architecture` (par défaut : `#false`)

Cette chaîne facultative n'est pertinente que si le noyau est Linux. Dans ce cas, elle correspond à la variable ARCH utilisée lors de la construction de Linux, "`mips`" par exemple.

`glibc-dynamic-linker`

Ce champ est le nom de l'éditeur des liens dynamique de la bibliothèque C de GNU pour le système correspondant, en une chaîne. Elle peut être "`/lib/ld-linux-armhf.so.3`".

### 15.2 Plateformes prises en charge

Les modules (`guix platforms ...`) exportent les variables suivantes, chacune d'entre elle étant liée à un enregistrement `platform`.

`armv7-linux` [Variable Scheme]

Une plateforme ciblant un CPU ARM v7 tournant sous GNU/Linux.

`aarch64-linux` [Variable Scheme]

Une plateforme ciblant un CPU ARM v8 tournant sous GNU/Linux.

- mips64-linux** [Variable Scheme]  
Une plateforme ciblant un CPU MIPS petit-boutiste 64-bits tournant sous GNU/Linux.
- powerpc-linux** [Variable Scheme]  
Une plateforme ciblant un CPU PowerPC grand-boutiste 32-bits tournant sous GNU/Linux.
- powerpc64le-linux** [Variable Scheme]  
Une plateforme ciblant un CPU PowerPC petit-boutiste 64-bits tournant sous GNU/Linux.
- riscv64-linux** [Variable Scheme]  
Une plateforme ciblant un CPU RISC-V 64-bits tournant sous GNU/Linux.
- i686-linux** [Variable Scheme]  
Une plateforme ciblant un CPU x86 tournant sous GNU/Linux.
- x86\_64-linux** [Variable Scheme]  
Une plateforme ciblant un CPU x86 64-bits tournant sous GNU/Linux.
- i686-mingw** [Variable Scheme]  
Une plateforme ciblant un CPU x86 tournant sous Windows, avec la prise en charge à l'exécution de MinGW.
- x86\_64-mingw** [Variable Scheme]  
Une plateforme ciblant un CPU x86 64-bits tournant sous Windows, avec la prise en charge à l'exécution de MinGW.
- i586-gnu** [Variable Scheme]  
Une plateforme ciblant un CPU x86 tournant sous GNU/Hurd (aussi appelé « GNU »).

## 16 Créer des images systèmes

Lorsque vient le moment d'installer Guix System pour la première fois sur une nouvelle machine, vous pouvez globalement procéder de trois manières différentes. La première manière est d'utiliser un système d'exploitation existant sur une machine pour lancer la commande `guix system init` (voir Section 12.15 [Invoquer guix system], page 584). La seconde consiste à produire une image d'installation (voir Section 3.9 [Construire l'image d'installation], page 36). C'est un système amorçable dont le rôle est de lancer `guix system init`. Enfin, la troisième possibilité est de produire une image qui est une instantiation directe du système que vous voulez exécuter. Cette image peut ensuite être copiée sur un périphérique amorçable comme un périphérique USB ou une carte mémoire. La machine cible démarrerait alors directement dessus, sans aucune procédure d'installation.

La commande `guix system image` peut transformer une définition de système d'exploitation en une image amorçable. Cette commande prend en charge différents types d'images, comme `efi-raw`, `iso9660` et `docker`. Toute machine `x86_64` récente sera sans doute capable de démarrer à partir d'une image `iso9660`. Cependant, il y a quelques machines qui nécessitent un type d'image spécifique. Ces machines, en général sous un processeur ARM, peuvent s'attendre à des partitions spécifiques à certaines positions particulières.

Ce chapitre explique comment définir des images systèmes personnalisées et comment les transformer en images amorçables.

### 16.1 Référence de image

L'enregistrement `image`, décrit juste après, vous permet de définir une image système amorçable personnalisée.

`image` [Type de données]

C'est le type de donnée représentant une image système.

`name` (par défaut : `#false`)

Le nom de l'image en tant que symbole, `'my-iso9660` par exemple. Le nom est facultatif et vaut `#false` par défaut.

`format` Le format d'image en tant que symbole. Les formats suivants sont pris en charge :

- `disk-image`, une image disque brute composée d'une ou plusieurs partitions.
- `compressed-qcow2`, une image qcow2 compressée composée d'une ou plusieurs partitions.
- `docker`, une image Docker.
- `iso9660`, une image ISO-9660.
- `tarball`, une archive d'image en tar.gz.
- `ws12`, une image WSL2.

`platform` (par défaut : `#false`)

L'enregistrement `platform` que l'image cible (voir Chapitre 15 [Plateformes], page 642), par exemple `aarch64-linux`. Par défaut, ce champ est à `#false` et l'image ciblera la plateforme de l'hôte.

`size` (par défaut : `'guess'`)

La taille de l'image en octets ou `'guess'`. Le symbole `'guess'`, qui est utilisé par défaut, signifie que la taille de l'image sera inférée à partir de son contenu.

`operating-system`

L'enregistrement `operating-system` à instancier dans l'image.

`partition-table-type` (par défaut : `'mbr'`)

Le type de table des partitions de l'image en tant que symbole. Les valeurs possibles sont `'mbr'` et `'gpt'`. La valeur par défaut est `'mbr'`.

`partitions` (par défaut : `'()'`)

Les partitions de l'image en tant que liste d'enregistrements `partition` (voir Section 16.1.1 [référence de partition], page 645).

`compression?` (par défaut : `#true`)

Indique si le contenu de l'image doit être compressé, en tant que booléen. Sa valeur par défaut est `#true` et ne s'applique qu'aux formats d'image `'iso9660'`.

`volatile-root?` (par défaut : `#true`)

Indique si la partition racine de l'image devrait être volatile, en tant que booléen.

Cela se fait en utilisant un système de fichiers en RAM (`overlayfs`) monté sur la partition racine par `l'initrd`. Sa valeur par défaut est `#true`. Lorsque la valeur est `#false`, la partition racine de l'image est montée en lecture-écriture par `l'initrd`.

`shared-store?` (par défaut : `#false`)

Indique si le dépôt de l'image devrait être partagé avec le système hôte, en tant que booléen. Cela peut être utile si vous créez des images dédiées à des machines virtuelles. Lorsque la valeur est `#false`, ce qui est la valeur par défaut, la clôture du `operating-system` de l'image est copiée dans l'image. Sinon, lorsque la valeur est `#true`, le dépôt de l'hôte est censé être disponible au démarrage avec un montage `9p` par exemple.

`shared-network?` (par défaut : `#false`)

Indique s'il faut utiliser les interfaces réseaux de l'hôte dans l'image, en tant que booléen. C'est utilisé uniquement pour le format d'image `'docker'`. Sa valeur par défaut est `#false`.

`substitutable?` (par défaut : `#true`)

Indique si la dérivation de l'image devrait être substituable, en tant que booléen. Sa valeur par défaut est `#true`.

### 16.1.1 Référence de partition

L'enregistrement `image` peut contenir des partitions.

`partition`

[Type de données]

C'est le type de donnée représentant une partition d'image.



- size** (par défaut : `'guess'`)  
La taille de la partition en octets ou `'guess'`. Le symbole `'guess'`, qui est la valeur par défaut, signifie que la taille de la partition sera inférée en fonction de son contenu.
- offset** (par défaut : `0`)  
La position du début de la partition en octets, relativement au début de l'image ou de la fin de la partition précédente. Sa valeur par défaut est `0` ce qui signifie qu'aucun décalage n'est appliqué.
- file-system** (par défaut : `"ext4"`)  
Le système de fichiers de la partition en tant que chaîne, par défaut `"ext4"`. Les valeurs prises en charge sont `"vfat"`, `"fat16"`, `"fat32"` et `"ext4"`.
- file-system-options** (par défaut : `'()`)  
Les options de création du système de fichiers de la partition qui seront passées à l'outil de création de partition, en tant que liste de chaînes. Ce n'est pris en charge que pour créer des partitions `"ext4"`.  
Voir la section `"extended-options"` dans la page de manuel de l'outil `"mke2fs"` pour une référence plus complète.
- label**  
L'étiquette de la partition en tant que chaîne obligatoire, par exemple `"ma-racine"`.
- uuid** (par défaut : `#false`)  
L'UUID de la partition en tant qu'enregistrement `uuid` (voir Section 12.3 [Systèmes de fichiers], page 250). Sa valeur par défaut est `#false`, ce qui signifie que l'outil de création de partition attribuera un UUID aléatoire à la partition.
- flags** (par défaut : `'()`)  
Les drapeaux de la partition en tant que liste de symboles. Les valeurs possibles sont `'boot` et `'esp`. Le drapeau `'boot` devrait être indiqué si vous voulez démarrer sur cette partition. Exactement une partition devrait avoir ce drapeau, typiquement la partition racine. Le drapeau `'esp` identifie une partition système pour UEFI.
- initializer** (par défaut : `#false`)  
La procédure d'initialisation de la partition en tant que `gexp`. Cette procédure est appelée pour remplir la partition. Si aucune procédure n'est passée, la procédure `initialize-root-partition` du module (`gnu build image`) sera utilisée.

## 16.2 Instancier une image

Imaginons que vous vouliez créer une image MBR avec trois partitions distinctes :

- L'ESP (partition système EFI), une partition de 40 Mio à la position 1024 Kio avec un système de fichiers `vfat`.
- une partition `ext4` de 50 Mio de données, et étiquetée « données ».

- une partition amorçable ext4 contenant le système d'exploitation %simple-os.

Vous écririez ensuite la définition d'image suivante dans un fichier `mon-image.scm` par exemple.

```
(use-modules (gnu)
             (gnu image)
             (gnu tests)
             (gnu system image)
             (guix gexp))

(define Mio (expt 2 20))

(image
 (format 'disk-image)
 (operating-system %simple-os)
 (partitions
 (list
 (partition
 (size (* 40 Mio))
 (offset (* 1024 1024))
 (label "GNU-ESP")
 (file-system "vfat")
 (flags '(esp))
 (initializer (gexp initialize-efi-partition)))
 (partition
 (size (* 50 Mio))
 (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))))))
```

Remarquez que la première et la troisième partition utilisent les procédures d'initialisation par défaut, respectivement `initialize-efi-partition` et `initialize-root-partition`. `Initialize-efi-partition` installe un chargeur EFI GRUB qui charge le chargeur d'amorçage GRUB situé sur la partition racine. `Initialize-root-partition` instancie un système complet défini par le système d'exploitation %simple-os.

Vous pouvez maintenant lancer :

```
guix system image mon-image.scm
```

pour instancier la définition de l'image. Cela produit une image disque qui a la structure attendue :

```
$ parted $(guix system image mon-image.scm) print
...
Model: (file)
Disk /gnu/store/yhylv1bp5b2ypb97pd3bbhz6jk5nbhxm-disk-image: 1714MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type    File system  Flags
  1      1049kB  43.0MB  41.9MB  primary fat16        esp
  2      43.0MB  95.4MB  52.4MB  primary ext4
  3      95.4MB  1714MB  1619MB  primary ext4        boot
```

La taille de la partition boot a été inférée à 1619 Mo pour qu'elle soit assez grande pour héberger le système d'exploitation `%simple-os`.

Vous pouvez aussi utiliser les définitions d'enregistrements `image` existantes et hériter d'elles pour simplifier la définition de l'image. Le module `(gnu system image)` fournit les variables de définition d'image suivantes.

`efi-disk-image` [Variable Scheme]

Une image disque MBR composée de deux partitions : une partition ESP 64 bits et une partition racine de démarrage. Cette image peut être utilisée sur la plupart des machines `x86_64` et `i686` qui prennent en charge l'amorçage via le BIOS ou l'UEFI.

`efi32-disk-image` [Variable Scheme]

Comme `efi-disk-image` mais avec une partition EFI 32 bits.

`iso9660-image` [Variable Scheme]

Une image ISO-9660 composée d'une seule partition amorçable. Cette image peut être utilisée sur la plupart des machines `x86_64` et `i686`.

`docker-image` [Variable Scheme]

Une image Docker qui peut être utilisée pour démarrer un conteneur Docker.

En utilisant `efi-disk-image` nous pouvons simplifier notre précédente déclaration d'image de cette manière :

```
(use-modules (gnu)
             (gnu image)
             (gnu tests)
             (gnu system image)
             (guix gexp)
             (ice-9 match))

(define Mio (expt 2 20))

(define data
  (partition
```

```

(size (* 50 Mio))
(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)))))

```

Cela donnera exactement la même instanciation d'image mas la déclaration de l'image est plus simple.

### 16.3 référence de image-type

La commande `guix system image` peut, comme nous l'avons vu plus haut, prendre un fichier contenant une déclaration d'image comme argument et produire une image disque à partir de celui-ci. Cette même commande peut aussi gérer un fichier contenant une déclaration `operating-system` comme argument. Dans ce cas, comment un `operating-system` est-il transformé en une image ?

C'est là que l'enregistrement `image-type` intervient. Cet enregistrement définit comment transformer un enregistrement `operating-system` en un enregistrement `image`.

**image-type** [Type de données]

C'est le type de donnée représentant un type d'image.

**name** Le nom du type d'image, en tant que symbole obligatoire, par exemple `'efi32-raw`.

**constructor** Le constructeur du type d'image, en tant que procédure obligatoire qui prend un enregistrement `operating-system` comme argument et renvoie un enregistrement `image`.

Il y a plusieurs enregistrements `image-type` fournis par les modules (`gnu system image`) et (`gnu system images ...`).

**efi-raw-image-type** [Variable Scheme]

Construit une image basée sur l'image `efi-disk-image`.

**efi32-raw-image-type** [Variable Scheme]

Construit une image basée sur l'image `efi32-disk-image`.

- qcow2-image-type** [Variable Scheme]  
Construit une image basée sur l'image `efi-disk-image` mais avec le format d'image `compressed-qcow2`.
- iso-image-type** [Variable Scheme]  
Construit une image compressée basée sur l'image `iso9660-image`.
- uncompressed-iso-image-type** [Variable Scheme]  
Construit une image basée sur l'image `iso9660-image` mais avec le champ `compression?` à `#false`.
- docker-image-type** [Variable Scheme]  
Construit une image basée sur l'image `docker-image`.
- raw-with-offset-image-type** [Variable Scheme]  
Construit une image MBR avec une seule partition commençant à l'adresse `1024 Kio`. C'est utile pour laisser la place pour installer un chargeur d'amorçage dans l'intervalle post-MBR.
- pinebook-pro-image-type** [Variable Scheme]  
Construit une image qui cible la machine Pinebook Pro. L'image MBR contient une seule partition commençant à l'adresse `9 Mio`. Le chargeur d'amorçage `u-boot-pinebook-pro-rk3399-bootloader` sera installé dans l'intervalle.
- rock64-image-type** [Variable Scheme]  
Construit une image qui cible la machine Rock64. L'image MBR contient une seule partition commençant à l'adresse `16 Mio`. Le chargeur d'amorçage `u-boot-rock64-rk3328-bootloader` sera installé dans l'intervalle.
- novena-image-type** [Variable Scheme]  
Construit une image qui cible la machine Novena. Elle a les mêmes caractéristiques que `raw-with-offset-image-type`.
- pine64-image-type** [Variable Scheme]  
Construit une image qui cible la machine Pine64. Elle a les mêmes caractéristiques que `raw-with-offset-image-type`.
- hurd-image-type** [Variable Scheme]  
Construit une image qui cible une machine `i386` qui exécute le noyau Hurd. L'image MBR contient une seule partition `ext2` avec des drapeaux `file-system-options` spécifiques.
- hurd-qcow2-image-type** [Variable Scheme]  
Construit une image similaire à celle construit par `hurd-image-type` mais avec le format `'compressed-qcow2`.
- wsl2-image-type** [Variable Scheme]  
Construit une image pour le WSL2 (sous-système Linux de Windows 2). Elle peut être importée en exécutant :
- ```
wsl --import Guix ./guix ./wsl2-image.tar.gz
wsl -d Guix
```

Donc, revenons à la commande `guix system image` qui prend une déclaration `operating-system` comme argument. Par défaut, `efi-raw-image-type` est utilisé pour transformer l'`operating-system` fourni en une image amorçable.

Pour utiliser un `image-type` différent, vous pouvez utiliser l'option `--image-type`. L'option `--list-image-types` listera tous les types d'images pris en charge. C'est une liste textuelle de toutes les variables `image-type` décrites plus haut (voir Section 12.15 [Invoquer `guix system`], page 584).

## 16.4 Modules des images

Prenons l'exemple du Pine64, une machine ARM. Pour pouvoir produire une image ciblant cette carte, nous avons besoin des éléments suivants :

- Un enregistrement `operating-system` contenant au moins un noyau approprié (`linux-libre-arm64-generic`) et un chargeur d'amorçage (`u-boot-pine64-lts-bootloader`) pour le Pine64.
- Éventuellement, un enregistrement `image-type` fournissant une manière de transformer l'enregistrement `operating-system` en un enregistrement `image` approprié pour le Pine64.
- Une `image` qui peut être instanciée avec la commande `guix system image`.

Le module (`gnu system images pine64`) fournit tous ces éléments : `pine64-barebones-os`, `pine64-image-type` et `pine64-barebones-raw-image`, respectivement.

Le module renvoie `pine64-barebones-raw-image` pour pouvoir exécuter :

```
guix system image gnu/system/images/pine64.scm
```

Maintenant, grâce à l'enregistrement `pine64-image-type` qui déclare l'`image-type` `'pine64-raw`, on peut aussi préparer un fichier `my-pine.scm` avec le contenu suivant :

```
(use-modules (gnu system images pine64))
(operating-system
 (inherit pine64-barebones-os)
 (timezone "Europe/Athens"))
```

pour personnaliser le `pine64-barebones-os`, puis lancer :

```
$ guix system image --image-type=pine64-raw my-pine.scm
```

Remarquez qu'il y a d'autres modules dans le répertoire `gnu/system/images` qui ciblent les machines Novena, Pine64, PinebookPro et Rock64.

## 17 Installer les fichiers de débogage

Les binaires des programmes, produits par les compilateurs GCC par exemple, sont typiquement écrits au format ELF, avec une section contenant des *informations de débogage*. Les informations de débogage sont ce qui permet au débogueur, GDB, de relier le code binaire et le code source ; elles sont requises pour déboguer un programme compilé dans de bonnes conditions.

Ce chapitre explique comment utiliser les informations de débogage séparées quand un paquet ne les fournit pas, et comment reconstruire un paquet avec les informations de débogage manquantes.

### 17.1 Informations de débogage séparées

Le problème avec les informations de débogage est qu'elles prennent pas mal de place sur le disque. Par exemple, les informations de débogage de la bibliothèque C de GNU prend plus de 60 Mo. Ainsi, en tant qu'utilisateur, garder toutes les informations de débogage de tous les programmes installés n'est souvent pas une possibilité. Cependant, l'économie d'espace ne devrait pas empêcher le débogage — en particulier, dans le système GNU, qui devrait faciliter pour ses utilisateurs l'exercice de leurs libertés (voir Section 1.2 [Distribution GNU], page 2).

Heureusement, les utilitaires binaires de GNU (Binutils) et GDB fournissent un mécanisme qui permet aux utilisateurs d'avoir le meilleur des deux mondes : les informations de débogage peuvent être nettoyées des binaires et stockées dans des fichiers séparés. GDB peut ensuite charger les informations de débogage depuis ces fichiers, lorsqu'elles sont disponibles (voir Section “Separate Debug Files” dans *Debugging with GDB*).

La distribution GNU se sert de cela pour stocker les informations de débogage dans le sous-répertoire `lib/debug` d'une sortie séparée du paquet appelée sans grande imagination `debug` (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59). Les utilisateurs peuvent choisir d'installer la sortie `debug` d'un paquet lorsqu'ils en ont besoin. Par exemple, la commande suivante installe les informations de débogage pour la bibliothèque C de GNU et pour GNU Guile :

```
guix install glibc:debug guile:debug
```

On doit ensuite dire à GDB de chercher les fichiers de débogage dans le profil de l'utilisateur, en remplissant la variable `debug-file-directory` (vous pourriez aussi l'instancier depuis le fichier `~/.gdbinit`, voir Section “Startup” dans *Debugging with GDB*) :

```
(gdb) set debug-file-directory ~/.guix-profile/lib/debug
```

À partir de là, GDB récupérera les informations de débogage des fichiers `.debug` sous `~/.guix-profile/lib/debug`.

Voici un script GDB alternatif qui est utile pour travailler avec d'autres profils. Il s'appuie sur l'intégration Guile facultative dans GDB. Ce bout de code est inclus par défaut sur le système Guix dans le fichier `~/.gdbinit`.

```
guile
(use-modules (gdb))
```

```
(execute (string-append "set debug-file-directory "
                        (or (getenv "GDB_DEBUG_FILE_DIRECTORY")
                            "~/guix-profile/lib/debug")))
end
```

EN plus, vous voudrez sans doute que GDB puisse montrer le code source débogué. Pour cela, vous devrez désarchiver le code source du paquet qui vous intéresse (obtenu via `guix build --source`, voir Section 10.1 [Invoquer `guix build`], page 180) et pointer GDB vers ce répertoire des sources avec la commande `directory` (voir Section “Source Path” dans *Debugging with GDB*).

Le mécanisme de sortie `debug` dans Guix est implémenté par le `gnu-build-system` (voir Section 9.5 [Systèmes de construction], page 124). Actuellement, ce n’est pas obligatoire — les informations de débogage sont disponibles uniquement si les définitions déclarent explicitement une sortie `debug`. Cela pourrait être modifié tout en permettant aux paquets de s’en passer dans le futur si nos serveurs de construction peuvent tenir la charge. Pour vérifier si un paquet a une sortie `debug`, utilisez `guix package --list-available` (voir Section 6.2 [Invoquer `guix package`], page 44).

Continuez à lire pour apprendre comment gérer les paquets qui n’ont pas de sortie `debug`.

## 17.2 Reconstruire les informations de débogage

Comme nous l’avons montré, certains paquets, mais pas tous, fournissent des informations de débogage dans une sortie `debug`. Que faire lorsque les informations de débogage sont manquantes ? L’option `--with-debug-info` fournit une solution à cela : il vous permet de reconstruire les paquets pour lesquels les informations de débogages sont manquantes — et seulement ceux-là — et les greffe sur les applications que vous déboguez. Ainsi, bien que ce ne soit pas aussi rapide qu’installer la sortie `debug`, ça demande relativement peu de choses.

Illustrons cela. Supposons que vous rencontriez un bogue dans Inkscape et que vous souhaitiez voir ce qui se passe dans GLib, une bibliothèque assez loin dans le graphe de dépendances. Il se trouve que GLib n’a pas de sortie `debug` et la trace de débogage de GDB est toute triste :

```
(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=0x7fffffffefe8) at dl-init.c:72
#3 0x00007ffff7fe2866 in call_init (env=0x7fffffffefe8, argv=0x7fffffffefd8, argc=1,
    at dl-init.c:118
```

Pour corriger cela, vous installez Inkscape lié à une variante de GLib qui contient les informations de débogage :

```
guix install inkscape --with-debug-info=glib
```

Cette fois-ci, le débogage a bien plus d’informations :

```
$ 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=0x7ffffd0a8) at dl-init.c:72
...
```

Bien mieux !

Remarquez qu'il y a des paquets pour lesquels `--with-debug-info` n'a pas l'effet désiré. Voir Section 10.1.2 [Options de transformation de paquets], page 183, pour plus d'informations.

## 18 Utiliser T<sub>E</sub>X et L<sup>A</sup>T<sub>E</sub>X

Guix fournit des paquets pour les systèmes de composition T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, ConTeXt, LuaTeX etc, qui font parti de la distribution T<sub>E</sub>X Live (<https://www.tug.org/texlive/>). Cependant, comme T<sub>E</sub>X Live est énorme et comme s’y retrouver est difficile, nous pensons que vous, chère utilisatrice, cher utilisateur, apprécierez un peu d’aide sur la manière de déployer les paquets utiles pour que vous puissiez compiler vos documents T<sub>E</sub>X et L<sup>A</sup>T<sub>E</sub>X.

T<sub>E</sub>X Live est actuellement proposé dans deux variantes dans Guix :

- Le paquet `texlive` « monolithique » : il possède *tous les paquets T<sub>E</sub>X Live* (plus de 7 000), mais il est énorme (plus de 4 Gio pour un seul paquet !).
- Les paquets `texlive-` « modulaires » : vous installez `texlive-base`, qui fournit les fonctionnalités de base et les commandes principales — `pdflatex`, `dvips`, `luatex`, `mf`, etc. — avec les paquets individuels qui fournissent juste les fonctionnalités dont vous avez besoin — `texlive-listings` pour le paquet `listings`, `texlive-hyperref` pour `hyperref`, `texlive-beamer` pour Beamer, `texlive-pgf` pour PGF/TikZ, etc.

Nous recommandons d’utiliser l’ensemble de paquets modulaires car il demande beaucoup moins de ressources. Pour construire vos documents, vous utiliserez des commandes comme :

```
guix shell texlive-base texlive-wrapfig \
  texlive-hyperref texlive-cm-super -- pdflatex doc.tex
```

Vous pouvez rapidement vous retrouver avec des commandes assez longues cependant. La solution est d’écrire un manifeste à la place, par exemple comme celui-ci :

```
(specifications->manifest
  ("rubber"

   "texlive-base"
   "texlive-wrapfig"

   "texlive-microtype"
   "texlive-listings" "texlive-hyperref"

   ;; PGF/TikZ
   "texlive-pgf"

   ;; Polices supplémentaires.
   "texlive-cm-super" "texlive-amsfonts"
   "texlive-times" "texlive-helvetica" "texlive-courier"))
```

Vous pouvez ensuite le passer à n’importe quelle commande avec l’option `-m` :

```
guix shell -m manifest.scm -- pdflatex doc.tex
```

Voir Section 9.4 [Écrire un manifeste], page 120, pour plus d’informations sur les manifestes. Dans le futur, nous prévoyons de fournir des paquets pour les *collections* de T<sub>E</sub>X Live — des « méta-paquets » comme `fontsfreemodern`, `humanities` ou `langarabic` qui fournissent l’ensemble de paquets requis dans ce domaine spécifique. Cela vous permettra de lister moins de paquets.

La principale difficulté ici est qu'utiliser l'ensemble de paquets modulaires vous force à choisir précisément les paquets dont vous avez besoin. Vous pouvez utiliser `guix search`, mais trouver le bon paquet peut être compliqué. Lorsqu'un paquet manque, `pdflatex` et les commandes similaires échouent avec un message obscur de ce type :

```
doc.tex: File `tikz.sty' not found.
doc.tex:7: Emergency stop.
```

ou, pour une police manquante :

```
kpathsea: Running mktexmf phvr7t
! I can't find file `phvr7t'.
```

Comment déterminer le paquet manquant ? Dans le premier cas, vous trouverez la réponse avec :

```
$ guix search texlive tikz
name: texlive-pgf
version: 59745
...
```

Dans le second cas, `guix search` ne renvoie rien. À la place, vous pouvez chercher dans la base de données des paquets T<sub>E</sub>X Live avec la commande `tlmgh` :

```
$ 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
```

Le fichier est disponible dans le paquet T<sub>E</sub>X Live `helvetic`, qui est connu dans Guix sous le nom `texlive-helvetic`. Quelle aventure !

Il y a une autre limite importante cependant : Guix fournit actuellement un sous-ensemble des paquets T<sub>E</sub>X Live. Si vous trouvez un paquet manquant, vous pouvez essayer de l'importer (voir Section 10.5 [Invoquer `guix import`], page 197) :

```
guix import texlive paquet
```

Les options supplémentaires comprennent :

```
--recursive
```

```
-r          Traverse le graphe des dépendances du paquet amont donné et génère les expressions de paquets de tous ceux qui ne sont pas déjà dans Guix.
```

**Remarque:** La gestion des paquets T<sub>E</sub>X Live est encore en chantier, mais vous pouvez nous aider ! Voir Chapitre 22 [Contribuer], page 668, pour plus d'informations.

## 19 Mises à jour de sécurité

Parfois, des vulnérabilités importantes sont découvertes dans les paquets logiciels et doivent être corrigées. Les développeur·euses de Guix essayent de suivre les vulnérabilités connues et d'appliquer des correctifs aussi vite que possible dans la branche `master` de Guix (nous n'avons pas encore de branche « stable » contenant seulement des mises à jour de sécurité). L'outil `guix lint` aide à trouver les versions vulnérables des paquets logiciels dans la distribution :

```
$ guix lint -c cve
gnu/packages/base.scm:652:2 : glibc@2.21 : probablement vulnérable à CVE-2015-1781, CVE-2015-7547
gnu/packages/gcc.scm:334:2 : gcc@4.9.3 : probablement vulnérable à CVE-2015-5276
gnu/packages/image.scm:312:2 : openjpeg@2.1.0 : probablement vulnérable à CVE-2016-1923, CVE-2016-1924
...
```

Voir Section 10.8 [Invoquer `guix lint`], page 213, pour plus d'informations.

Guix suit une discipline de gestion de paquets fonctionnelle (voir Chapitre 1 [Introduction], page 1), ce qui implique que lorsqu'un paquet change, *tous les paquets qui en dépendent* doivent être reconstruits. Cela peut grandement ralentir le déploiement de corrections dans les paquets du cœur comme `libc` ou `bash` comme presque toute la distribution aurait besoin d'être reconstruite. Cela aide d'utiliser des binaires pré-construits (voir Section 6.3 [Substituts], page 54), mais le déploiement peut toujours prendre plus de temps de souhaité.

Pour corriger cela, Guix implémente les *greffes*, un mécanisme qui permet un déploiement rapide des mises à jour de sécurité critiques sans le coût associé à une reconstruction complète de la distribution. L'idée est de reconstruire uniquement le paquet qui doit être corrigé puis de le « greffer » sur les paquets qui sont explicitement installés par l'utilisateur et qui se référaient avant au paquet d'origine. Le coût d'une greffe est typiquement très bas, et plusieurs ordres de grandeurs moins élevé que de reconstruire tout la chaîne de dépendance.

Par exemple, supposons qu'une mise à jour de sécurité doive être appliquée à Bash. Les développeurs de Guix fourniront une définition de paquet pour le Bash « corrigé », disons *bash-fixed*, de la manière habituelle (voir Section 9.2 [Définition des paquets], page 105). Ensuite, la définition originale du paquet est augmentée avec un champ `replacement` qui pointe vers le paquet contenant la correction :

```
(define bash
  (package
    (name "bash")
    ;; ...
    (replacement bash-fixed)))
```

À partir de maintenant, tout paquet dépendant directement ou indirectement de Bash — rapporté par `guix gc --requisites` (voir Section 6.5 [Invoquer `guix gc`], page 60) — installé est automatiquement « réécrit » pour se référer à *bash-fixed* au lieu de *bash*. Ce processus de greffe prend du temps en proportion de la taille du paquet, typiquement moins d'une minute pour un paquet de taille « moyenne » sur une machine récente. La greffe est récursive : lorsqu'une dépendance indirecte a besoin d'être greffée, la greffe se « propage » jusqu'au paquet que l'utilisateur installe.

Actuellement la longueur du nom et la version de la greffe et du paquet qu'il remplace (*bash-fixed* et *bash* dans l'exemple ci-dessus) doivent être identiques. Cette restriction vient surtout du fait que la greffe fonctionne en corrigeant les fichiers, dont des fichiers binaires, directement. D'autres restrictions peuvent apparaître : par exemple, si vous ajoutez une greffe à un paquet fournissant une bibliothèque partagée, la bibliothèque partagée originale et son remplacement doivent avoir le même **SONAME** et être compatibles au niveau binaire.

L'option en ligne de commande `--no-grafts` vous permet d'éviter les greffes (voir Section 10.1.1 [Options de construction communes], page 180). Donc la commande :

```
guix build bash --no-grafts
```

renvoie le nom de fichier dans les dépôt du Bash original, alors que :

```
guix build bash
```

renvoie le nom de fichier du Bash « corrigé » de remplacement. Cela vous permet de distinguer les deux variantes de Bash.

Pour vérifier à quel Bash votre profil se réfère, vous pouvez lancer (voir Section 6.5 [Invoquer `guix gc`], page 60) :

```
guix gc -R $(readlink -f ~/.guix-profile) | grep bash
```

... et comparer les noms de fichiers que vous obtenez avec ceux du dessus. De la même manière pour une génération du système Guix :

```
guix gc -R $(guix system build my-config.scm) | grep bash
```

Enfin, pour vérifier quelles processus Bash lancés vous utilisez, vous pouvez utiliser la commande `lsuf` :

```
lsuf | grep /gnu/store/*.bash
```

## 20 Bootstrapping

Dans notre contexte, le bootstrap se réfère à la manière dont la distribution est construite « à partir de rien ». Rappelez-vous que l’environnement de construction d’une dérivation ne contient rien d’autre que les entrées déclarées (voir Chapitre 1 [Introduction], page 1). Donc il y a un problème évident de poule et d’œuf : comment le premier paquet est-il construit ? Comment le premier compilateur est-il construit ?

Il est tentant de penser que seuls les bidouilleurs les plus enthousiastes se soucient de cette question. Cependant, bien que la réponse à cette question soit technique par nature, ses implications sont très vastes. La manière dont une distribution est amorcée définit l’étendue de ce que nous, individuellement et collectivement en tant qu’utilisateurs, utilisatrices, bidouilleuses et bidouilleurs, pouvons faire confiance aux logiciels que nous lançons. C’est une préoccupation centrale du point de vue de la *sécurité* et de *liberté des utilisateurs*.

Le système GNU est surtout fait de code C, avec la libc en son cœur. Le système de construction GNU lui-même suppose la disponibilité d’un shell Bourne et d’outils en ligne de commande fournis par GNU Coreutils, Awk, Findutils, sed et grep. En plus, les programmes de construction — les programmes qui exécutent `./configure`, `make` etc — sont écrits en Guile Scheme (voir Section 9.10 [Dérivations], page 159). En conséquence, pour pouvoir construire quoi que ce soit, de zéro, Guix a besoin de binaire pré-construits de Guile, GCC, Binutils, la libc et des autres paquets mentionnés plus haut — les *binaires de bootstrap*.

Ces binaires de bootstrap sont pris comme des acquis, bien qu’on puisse les recréer (ça arrive plus tard) si besoin (voir Section 20.2 [Se préparer à utiliser les binaires de bootstrap], page 662).

### 20.1 Le bootstrap avec les sources binaires réduites

Guix — comme les autres distributions GNU/Linux — est traditionnellement bootstrappé à partir d’un ensemble de binaires de bootstrap : Bourne shell, des outils en ligne de commande fournis par GNU Coreutils, Awk, Findutils, « sed » et « grep » et Guile, GCC, Binutils et la bibliothèque C de GNU (voir Chapitre 20 [Bootstrapping], page 659). Habituellement, ces binaires de bootstrap sont « pris pour acquis ».

Prendre les binaires de bootstrap pour acquis signifie que nous les considérons comme des « sources » corrects et de confiance pour la construction d’un système complet. Là se trouve le problème : la taille de ces binaires de bootstrap est d’environ 250 Mo (voir Section “Bootstrappable Builds” dans *GNU Mes*). Auditer ou même inspecter ces binaires est presque impossible.

Pour `i686-linux` et `x86_64-linux`, Guix propose maintenant un bootstrap avec « source binaire réduite »<sup>1</sup>.

Le bootstrap avec les sources binaires réduites supprime les outils les plus critiques — du point de vue de la confiance — des binaires de bootstrap : GCC, Binutils et la bibliothèque C de GNU sont remplacées par : `bootstrap-mescc-tools` (un petit assembleur et éditeur des liens) et `bootstrap-mes` (un petit interpréteur Scheme et un compilateur C écrit en Scheme et la bibliothèque C de Mes, construite pour TinyCC et pour GCC).

<sup>1</sup> On aimerait pouvoir dire « Bootstrap Source Complet » et même si on travail vers ce but, utiliser ce terme serait une hyperbole pour ce qu’on fait actuellement.

À partir de ces nouvelles sources binaires, les Binutils, GCC et la bibliothèque C de GNU sont construits à partir des sources. À partir de là, les processus de bootstrap plus traditionnel continue. Cette approche a réduit la taille des binaires de bootstrap à environ 145 Mo dans Guix v1.1.

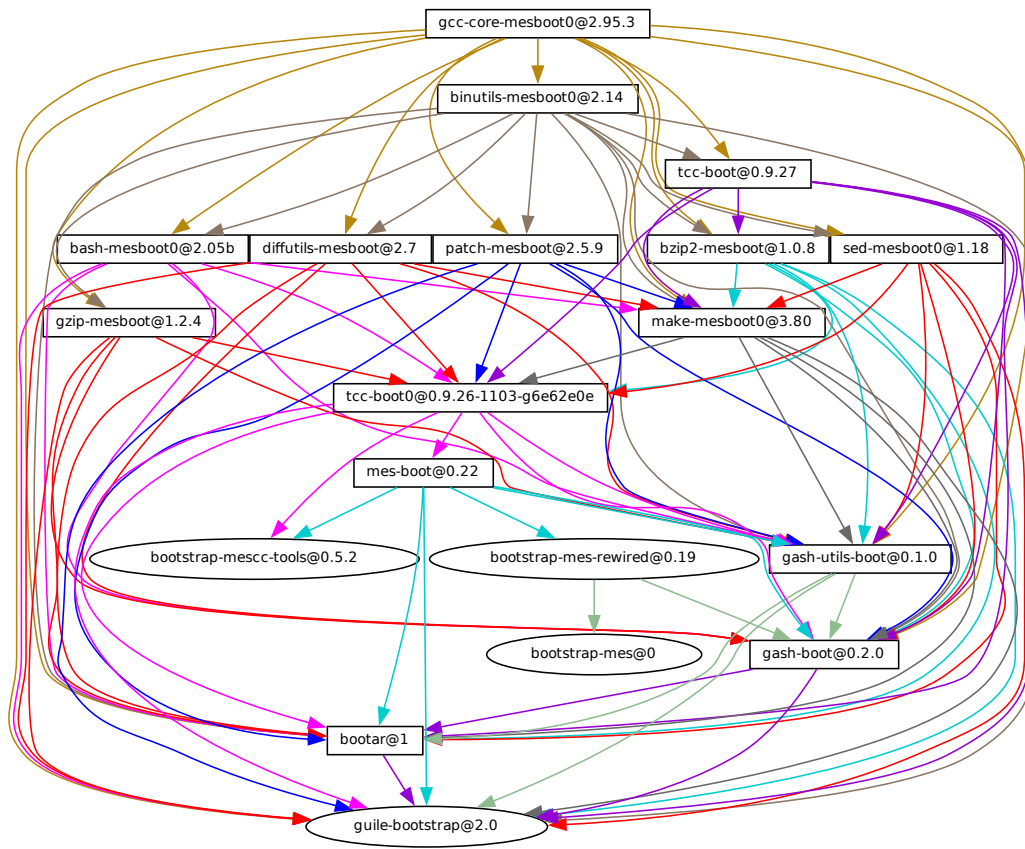
La prochaine étape qui Guix prépare est de remplacer le shell et tous ses utilitaires avec des implémentations en Guile Scheme, le *bootstrap Scheme-seulement*. Gash (voir Section “Gash” dans *The Gash manual*) est un shell compatible avec POSIX qui remplace Bash, et il propose Gash Utils qui a des remplaçants minimalistes pour Awk, les utilitaires GNU Core Utilities, Grep, Gzip, Sed et Tar. Le reste des binaires de bootstrap que nous avons enlevés sont maintenant construits à partir des sources.

Construire le Système GNU à partir des sources est actuellement seulement possible en ajoutant certains paquets GNU historiques en tant qu’étapes intermédiaires<sup>2</sup>. Tandis que Gash et Gash Utils avancent, et que les paquets GNU deviennent de nouveau plus bootstrappables (p. ex. les nouvelles versions de GNU Sed proposeront de nouveau des archives gzip, en alternative à la compression `xz` difficile à bootstrapper), on espère pouvoir réduire de nouveau cet ensemble de paquets supplémentaires.

Le graphe ci-dessous montre le graphe de dépendance résultant pour `gcc-core-mesboot`, le compilateur de bootstrap utilisé pour le bootstrap traditionnel du reste du système Guix.

---

<sup>2</sup> Les paquets comme `gcc-2.95.3`, `binutils-2.14`, `glibc-2.2.5`, `gzip-1.2.4`, `tar-1.22` et certains autres. Pour les détails, voir `gnu/packages/commencement.scm`.



La seule source binaire restant<sup>3</sup> sont un interpréteur Scheme et un compilateur Scheme : GNU Mes et GNU Guile<sup>4</sup>.

Cette réduction supplémentaire a réduit la taille du bootstrap à environ 60 Mo pour `i686-linux` et `x86_64-linux`.

Le travail continue pour supprimer tous les blobs binaires de notre pile de bootstrap libre, en essayant d'atteindre un bootstrap complet par les sources. Le travail continue aussi pour apporter ces bootstraps aux architectures `arm-linux` et `aarch64-linux` et au Hurd.

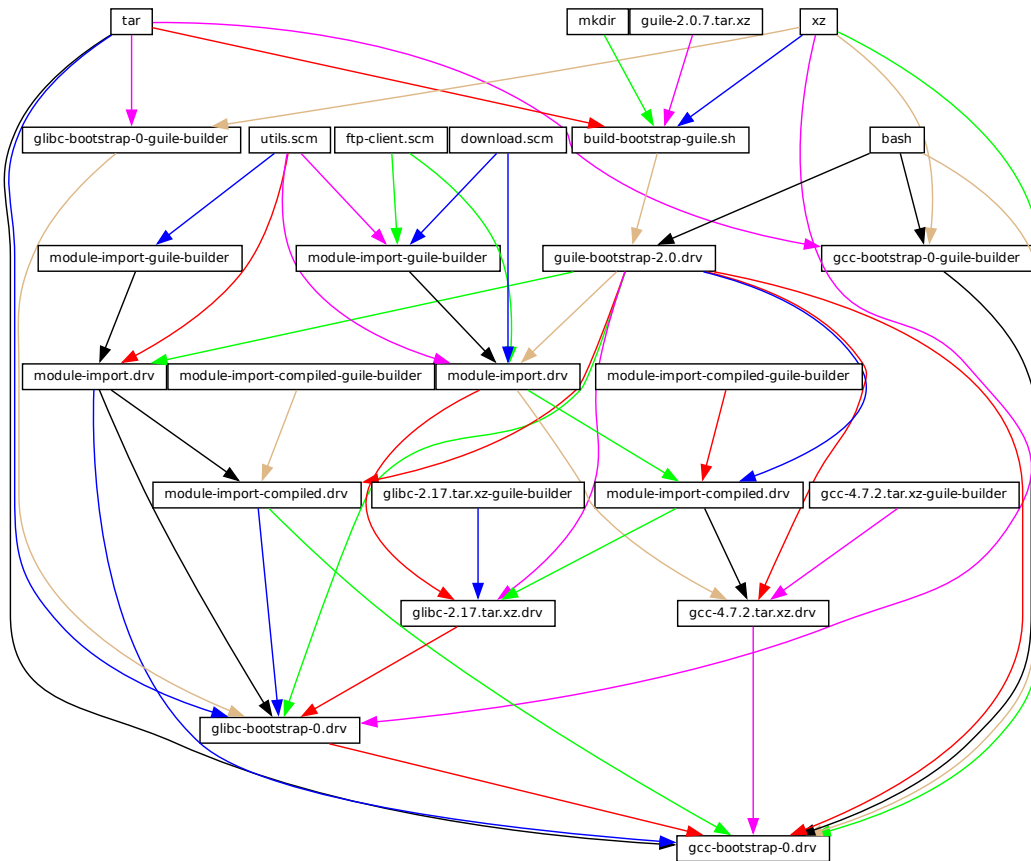
Si vous êtes intéressé-e, rejoignez-nous sur '#bootstrappable' sur le réseau IRC Freenode ou discutez-en sur `bug-mes@gnu.org` ou `gash-devel@nongnu.org`.

<sup>3</sup> En ignorant `mescc-tools` qui fait 68 Ko ; qui sera supprimée plus tard, avec `mes`.

<sup>4</sup> Ce graphe ne montre pas les binaires statiques pour `bash`, `tar` et `xz` utilisés pour pouvoir lancer Guile.



## 20.2 Se préparer à utiliser les binaires de bootstrap



La figure ci-dessus montre le tout début du graphe de dépendances de la distribution, correspondant aux définitions des paquets du module (`gnu packages bootstrap`). Une figure similaire peut être générée avec `guix graph` (voir Section 10.10 [Invoquer `guix graph`], page 218), de cette manière :

```
guix graph -t derivation \
  -e '((@ (gnu packages bootstrap) %bootstrap-gcc) ' \
  | dot -Tps > gcc.ps
```

ou, pour le bootstrap avec les sources binaires plus réduites

```
guix graph -t derivation \
  -e '((@ (gnu packages bootstrap) %bootstrap-mes) ' \
  | dot -Tps > mes.ps
```

À ce niveau de détails, les choses sont légèrement complexes. Tout d'abord, Guile lui-même consiste en un exécutable ELF, avec plusieurs fichiers Scheme sources et compilés qui sont chargés dynamiquement quand il est exécuté. Cela est stocké dans l'archive `guile-`

2.0.7.`tar.xz` montrée dans ce graphe. Cette archive fait parti de la distribution « source » de Guix, et est insérée dans le dépôt avec `add-to-store` (voir Section 9.9 [Le dépôt], page 157).

Mais comment écrire une dérivation qui décompresse cette archive et l'ajoute au dépôt ? Pour résoudre ce problème, la dérivation `guile-bootstrap-2.0.drv` — la première qui est construite — utilise `bash` comme constructeur, qui lance `build-bootstrap-guile.sh`, qui à son tour appelle `tar` pour décompresser l'archive. Ainsi, `bash`, `tar`, `xz` et `mkdir` sont des binaires liés statiquement, qui font aussi partie de la distribution source de Guix, dont le seul but est de permettre à l'archive de Guile d'être décompressée.

Une fois que `guile-bootstrap-2.0.drv` est construit, nous avons un Guile fonctionnel qui peut être utilisé pour exécuter les programmes de construction suivants. Sa première tâche consiste à télécharger les archives contenant les autres binaires pré-construits — c'est ce que la dérivation `.tar.xz.drv` accomplit. Les modules Guix comme `ftp-client.scm` sont utilisés pour cela. Les dérivations `module-import.drv` importent ces modules dans un répertoire dans le dépôt, en utilisant la disposition d'origine. Les dérivations `module-import-compiled.drv` compilent ces modules, et les écrivent dans un répertoire de sortie avec le bon agencement. Cela correspond à l'argument `#:modules` de `build-expression->derivation` (voir Section 9.10 [Dérivations], page 159).

Enfin, les diverses archives sont décompressées par les dérivations `gcc-bootstrap-0.drv`, `glibc-bootstrap-0.drv`, ou `bootstrap-mes-0.drv` et `bootstrap-mescc-tools-0.drv`, à ce stade, nous avons une chaîne d'outils C qui fonctionne.

## Construire les outils de construction

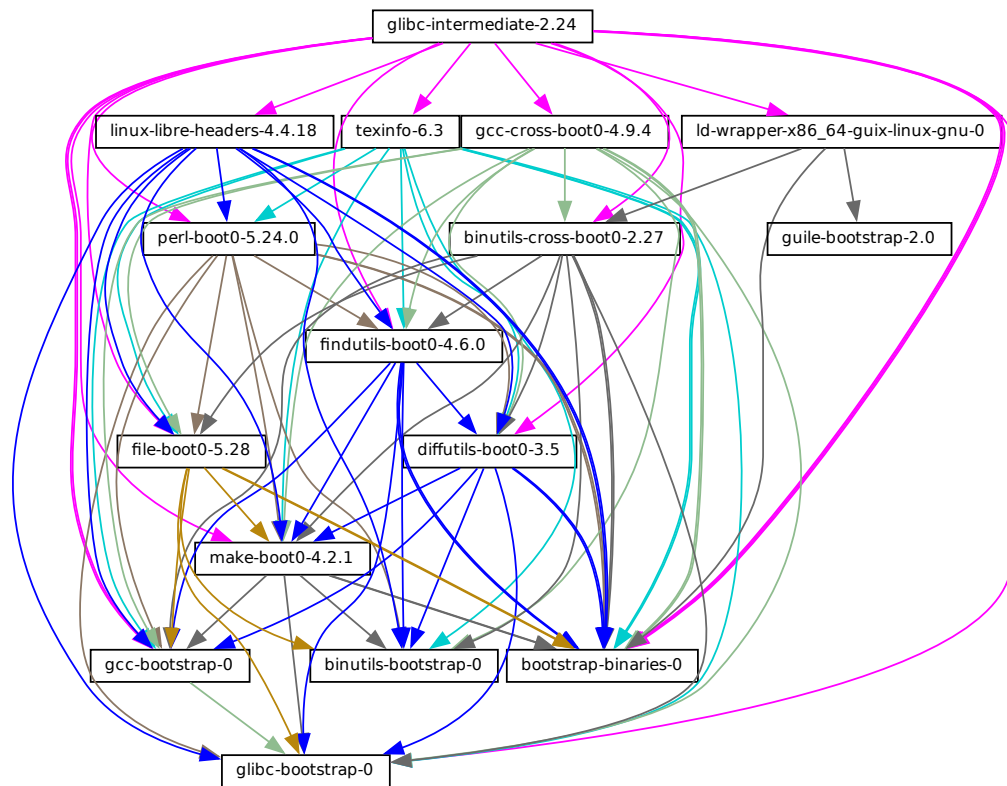
Le bootstrap est complet lorsque nous avons une chaîne d'outils complète qui ne dépend pas des outils de bootstrap pré-construits dont on vient de parler. Ce pré-requis d'indépendance est vérifié en s'assurant que les fichiers de la chaîne d'outil finale ne contiennent pas de référence vers les répertoires `/gnu/store` des entrées de bootstrap. Le processus qui mène à cette chaîne d'outils « finale » est décrit par les définitions de paquets qui se trouvent dans le module (`gnu packages commencement`).

La commande `guix graph` nous permet de « dézoomer » comparé au graphe précédent, en regardant au niveau des objets de paquets plutôt que des dérivations individuelles — rappelez-vous qu'un paquet peut se traduire en plusieurs dérivations, typiquement une dérivation pour télécharger ses sources, une pour les modules Guile dont il a besoin et une pour effectivement compiler le paquet depuis les sources. La commande :

```
guix graph -t bag \
  -e '(@@ (gnu packages commencement)
        glibc-final-with-bootstrap-bash)' | dot -Tps > t.ps
```

produit le graphe de dépendances qui mène à la bibliothèque C « finale »<sup>5</sup>, représentée ci-dessous.

<sup>5</sup> Vous remarquerez qu'elle s'appelle `glibc-intermediate`, ce qui suggère qu'elle n'est pas *tout à fait* finale, mais c'est une bonne approximation tout de même.



Le premier outil construit avec les binaires de bootstrap est GNU Make — appelé `make-boot0` ci-dessus — qui est un prérequis de tous les paquets suivants. Ensuite, Findutils et Diffutils sont construits.

Ensuite vient la première passe de Binutils et GCC, construits comme des pseudo outils croisés — c.-à-d. dont `--target` est égal à `--host`. Ils sont utilisés pour construire la libc. Grâce à cette astuce de compilation croisée, cette libc est garantie de ne contenir aucune référence à la chaîne d'outils initiale.

À partir de là, les Binutils finaux et GCC (non visibles ci-dessus) sont construits. GCC utilise `ld` du Binutils final et lie le programme avec la libc qui vient d'être construite. Cette chaîne d'outils est utilisée pour construire les autres paquets utilisés par Guix et par le système de construction de GNU : Guile, Bash, Coreutils, etc.

Et voilà ! À partir de là nous avons l'ensemble complet des outils auxquels s'attend le système de construction GNU. Ils sont dans la variable `%final-inputs` du module (`gnu packages commencement`) et sont implicitement utilisés par tous les paquets qui utilisent le `gnu-build-system` (voir Section 9.5 [Systèmes de construction], page 124).

## Construire les binaires de bootstrap

Comme la chaîne d'outils finale ne dépend pas des binaires de bootstrap, ils ont rarement besoin d'être mis à jour. Cependant, il est utile d'avoir une manière de faire cela automatiquement, dans le cas d'une mise à jour et c'est ce que le module (`gnu packages make-bootstrap`) fournit.

La commande suivante construit les archives contenant les binaires bootstrap (Binutils, GCC, glibc, pour le bootstrap traditionnel et les linux-libre-headers, bootstrap-mescc-tools, bootstrap-mes pour le bootstrap Reduced Binary Seed, et Guile, et une archive contenant un mélange de Coreutils et d'autres outils de base de la ligne de commande) :

```
guix build bootstrap-tarballs
```

Les archives générées sont celles qui devraient être référencées dans le module (`gnu packages bootstrap`) au début de cette section.

Vous êtes toujours là ? Alors peut-être que maintenant vous vous demandez, quand est-ce qu'on atteint un point fixe ? C'est une question intéressante ! La réponse est inconnue, mais si vous voulez enquêter plus profondément (et que vous avez les ressources en puissance de calcul et en capacité de stockage pour cela), dites-le nous.

## Réduire l'ensemble des binaires de bootstrap

Nos binaires de bootstrap incluent actuellement GCC, GNU Libc, Guile, etc. C'est beaucoup de code binaire ! Pourquoi est-ce un problème ? C'est un problème parce que ces gros morceaux de code binaire sont en pratique impossibles à auditer, ce qui fait qu'il est difficile d'établir quel code source les a produit. Chaque binaire non vérifiable nous rend également vulnérables aux portes dérobées des compilateurs, comme le décrit Ken Thompson dans le document *Reflections on Trusting Trust* de 1984.

Cela est rendu moins inquiétant par le fait que les binaires de bootstrap ont été générés par une révision antérieure de Guix. Cependant, il leur manque le niveau de transparence que l'on obtient avec le reste des paquets du graphe de dépendance, où Guix nous donne toujours une correspondance source-binaire. Ainsi, notre but est de réduire l'ensemble des binaires de bootstrap au minimum.

Le site web [Bootstrappable.org](http://bootstrappable.org) (<http://bootstrappable.org>) liste les projets en cours à ce sujet. L'un d'entre eux parle de remplacer le GCC de bootstrap par une série d'assembleurs, d'interpréteurs et de compilateurs d'une complexité croissante, qui pourraient être construits à partir des sources à partir d'un assembleur simple et auditable. Votre aide est (bien sûr !) la bienvenue.

Notre premier grand succès est le remplacement de GCC, de la bibliothèque C de GNU et de Binutils par MesCC-Tools (un éditeur de lien hexadécimal et un macro assembleur) et Mes (voir *GNU Mes*, un interpréteur Scheme et un compilateur C écrit en Scheme). Ni MesCC-Tools ni Mes ne peuvent encore être bootstrappés et donc nous les injectons comme des sources binaires. Nous appelons cela le bootstrap avec les sources binaires réduites, comme cela a réduit de moitié la taille de nos binaires de bootstrap ! Aussi, cela a éliminé le compilateur C binaire ; les paquets `i686-linux` et `x86_64-linux` de Guix sont maintenant bootstrappés sans aucun compilateur C binaire.

Le travail continue pour rendre MesCC-Tools et Mes complètement bootstrappables et nous cherchons aussi comment remplacer tout autre binaire de bootstrap. votre aide est la bienvenue !

## 21 Porter vers une nouvelle plateforme

Comme nous en avons discuté plus haut, la distribution GNU est auto-contenue, et cela est possible en se basant sur des « binaires de bootstrap » pré-construits (voir Chapitre 20 [Bootstrapping], page 659). Ces binaires sont spécifiques au noyau de système d’exploitation, à l’architecture CPU et à l’interface applicative binaire (ABI). Ainsi, pour porter la distribution sur une plateforme qui n’est pas encore supportée, on doit construire ces binaires de bootstrap et mettre à jour le module (`gnu packages bootstrap`) pour les utiliser sur cette plateforme.

Heureusement, Guix peut effectuer une *compilation croisée* de ces binaires de bootstrap. Lorsque tout va bien, et en supposant que la chaîne d’outils GNU supporte la plateforme cible, cela peut être aussi simple que de lancer une commande comme ceci :

```
guix build --target=armv5tel-linux-gnueabi bootstrap-tarballs
```

Pour que cela fonctionne, vous devez d’abord enregistrer une nouvelle plateforme définie dans le module (`guix platform`). Une plateforme connecte un triplet GNU (voir Section “Specifying Target Triplets” dans *Autoconf*), l’équivalent du *système* en notation Nix, le nom du *glibc-dynamic-linker* et le nom de l’architecture Linux correspondante si applicable (voir Chapitre 15 [Plateformes], page 642)..

Une fois que les archives de bootstrap sont construites, le module (`gnu packages bootstrap`) doit être mis à jour pour se référer à ces binaires sur la plateforme cible. C’est à dire que les hashes et les URL des archives de bootstrap pour la nouvelle plateforme doivent être ajoutés avec ceux des plateformes actuellement supportées. L’archive de bootstrap de Guile est traitée séparément : elle doit être disponible localement, et `gnu/local.mk` a une règle pour la télécharger pour les architectures supportées ; vous devez également ajouter une règle pour la nouvelle plateforme.

En pratique, il peut y avoir des complications. Déjà, il se peut que le triplet GNU étendu qui spécifie l’ABI (comme le suffixe `eabi` ci-dessus) ne soit pas reconnu par tous les outils GNU. Typiquement, la *glibc* en reconnaît certains, alors que *GCC* utilise un drapeau de configuration `--with-abi` supplémentaire (voir `gcc.scm` pour trouver des exemples où ce cas est géré). Ensuite, certains des paquets requis pourraient échouer à se construire pour cette plateforme. Enfin, les binaires générés pourraient être cassé pour une raison ou une autre.

## 22 Contribuer

Ce projet est un effort coopératif et nous avons besoin de votre aide pour le faire grandir ! Contactez-nous sur [guix-devel@gnu.org](mailto:guix-devel@gnu.org) et [#guix](https://libera.chat) sur le réseau IRC Libera Chat. Nous accueillons les idées, les rapports de bogues, les correctifs et tout ce qui pourrait aider le projet. Nous apprécions particulièrement toute aide sur la création de paquets (voir Section 22.4 [Consignes d’empaquetage], page 673).

Nous souhaitons fournir un environnement chaleureux, amical et sans harcèlement pour que tout le monde puisse contribuer au mieux de ses capacités. Pour cela notre projet a une « Convention de contribution » adaptée de <http://contributor-covenant.org/>. Vous pouvez trouver une version locale dans le fichier `CODE-OF-CONDUCT` dans l’arborescence des sources.

Les contributeur·rices n’ont pas besoin d’utiliser leur nom légal dans leurs correctifs et leurs communications en ligne ; elles et ils peuvent utiliser n’importe quel nom ou pseudonyme de leur choix.

### 22.1 Construire depuis Git

Si vous souhaitez travailler sur Guix lui-même, il est recommandé d’utiliser la dernière version du dépôt Git :

```
git clone https://git.savannah.gnu.org/git/guix.git
```

Comment vous assurer que vous avez obtenu une copie authentique du dépôt ? Pour cela, exécutez `guix git authenticate`, en lui transmettant le commit et l’empreinte OpenPGP de l’*canal introduction* (voir Section 8.5 [Invoquer `guix git authenticate`], page 102) :

```
git fetch origin keyring:keyring
guix git authenticate 9edb3f66fd807b096b48283debdccdcfea34bad \
  "BBB0 2DDF 2CEA F6A8 0D1D E643 A2A0 6DF2 A33A 54FA"
```

Cette commande se termine avec le code de sortie zéro en cas de succès ; elle affiche un message d’erreur et quitte avec un code non zéro dans le cas contraire.

Comme vous pouvez le voir, il y a un problème de poule et d’œuf : il faut d’abord avoir installé Guix. En général, vous installez Guix System (voir Chapitre 3 [Installation du système], page 26) ou Guix sur une autre distribution (voir Section 2.1 [Installation binaire], page 5) ; dans les deux cas, vous vérifiez la signature OpenPGP sur le support d’installation. Cela "amorç" la chaîne de confiance.

La manière la plus simple de configurer un environnement de développement pour Guix est, bien sûr, d’utiliser Guix ! La commande suivante démarre un nouveau shell où toutes les dépendances et les variables d’environnements appropriées sont configurés pour travailler sur Guix :

```
guix shell -D guix --pure
```

Voir Section 8.1 [Invoquer `guix shell`], page 83, pour plus d’informations sur cette commande.

Si vous ne pouvez pas utiliser Guix pour la construction de Guix depuis un extrait, les paquets suivants sont requis en plus de ceux mentionnés dans les instructions d’installation (voir Section 2.2 [Prérequis], page 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 (facultatif) (<https://www.gnu.org/software/help2man/>).

Avec Guix, vous pouvez ajouter des dépendances supplémentaires en lançant `guix shell` :

```
guix shell -D guix help2man git strace --pure
```

Vous pouvez maintenant générer l’infrastructure du système de construction avec Autoconf et Automake :

```
./bootstrap
```

Si vous obtenez une erreur de ce type :

```
configure.ac:46: error: possibly undefined macro: PKG_CHECK_MODULES
```

cela signifie probablement qu’Autoconf n’a pas pu trouver `pkg.m4` qui est fourni par `pkg-config`. Assurez-vous que `pkg.m4` est disponible. C’est aussi vrai pour l’ensemble de macros de `guile.m4` fournies par Guile. Par exemple, si vous avez installé Automake dans `/usr/local`, il ne cherchera pas les fichiers `.m4` dans `/usr/share`. Dans ce cas vous devez invoquer la commande suivante :

```
export ACLOCAL_PATH=/usr/share/aclocal
```

Voir Section “Macro Search Path” dans *The GNU Automake Manual*, pour plus d’information.

Ensuite lancez :

```
./configure --localstatedir=/var
```

... où `-var` est la valeur `localstatedir` habituelle (voir Section 9.9 [Le dépôt], page 157, pour plus d’informations à ce propos). Remarquez que vous n’allez probablement pas lancer `make install` à la fin (vous n’avez pas besoin de la faire) mais il est toujours important de passer le bon `localstatedir`.

Enfin, vous pouvez construire Guix et, si vous le souhaitez, lancer les tests (voir Section 2.3 [Lancer la suite de tests], page 9) :

```
make
make check
```

Si quelque chose échoue, jetez un œil aux instructions d’installation (voir Chapitre 2 [Installation], page 5) ou envoyez un message à la liste `guix-devel@gnu.org`.

À partir de maintenant, vous pouvez authentifier tous les commits de votre extrait en lançant :

```
make authenticate
```

La première exécution prend quelques minutes, mais les exécutions suivantes seront plus rapides.

Ou bien, si la configuration de votre dépôt Git local ne correspond pas à la configuration par défaut, vous pouvez fournir la référence pour la branche `keyring` dans la variable `GUIX_GIT_KEYRING`. L’exemple suivant suppose que vous avez un dépôt distant nommé ‘`myremote`’ qui pointe vers le dépôt officiel :

```
make authenticate GUIX_GIT_KEYRING=myremote/keyring
```



**Remarque:** Nous vous conseillons d'exécuter `make authenticate` après chaque invocation de `git pull`. Cela vous garantit de continuer à recevoir des modifications valables dans le dépôt.

Après la mise à jour du dépôt, `make` peut échouer avec une erreur semblable à celle de l'exemple suivant :

```
error: failed to load 'gnu/packages/dunst.scm':
ice-9/eval.scm:293:34: In procedure abi-check: #<record-type <origin>>: record ABI mis
```

Cela signifie qu'un des types d'enregistrement que Guix définit (dans cet exemple, l'enregistrement `origin`) a changé, et que l'entièreté de guix doit être recompilé pour prendre ce changement en compte. Pour ce faire, lancez `make clean-go` suivi de `make`.

## 22.2 Lancer Guix avant qu'il ne soit installé

Pour garder un environnement de travail sain, il est utile de tester les changements localement sans les installer pour de vrai. Pour pouvoir distinguer votre rôle « d'utilisateur-riche final-e » de celui parfois haut en couleur de « développeur-euse ».

Pour cela, tous les outils en ligne de commande sont utilisables même sans avoir lancé `make install`. Pour faire cela, vous devez d'abord avoir un environnement avec toutes les dépendances disponibles (voir Section 22.1 [Construire depuis Git], page 668), et puis préfixer simplement chaque commande avec `./pre-inst-env` (le script `pre-inst-env` se trouve dans le répertoire de plus haut niveau de l'arborescence des sources de Guix ; il est généré en lançant `./bootstrap` suivi de `./configure`). À titre d'exemple, voici comment construire le paquet `hello` tel que défini dans votre arborescence de travail (cela suppose que `guix-daemon` fonctionne déjà sur votre système ; c'est OK si c'est une version différente) :

```
$ ./pre-inst-env guix build hello
```

De même, pour une session Guile qui utilise les modules Guix :

```
$ ./pre-inst-env guile -c '(use-modules (guix utils)) (pk (%current-system))'
;;; ("x86_64-linux")
```

... et pour une REPL (voir Section 9.14 [Utiliser Guix de manière interactive], page 177)

:

```
$ ./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
```

Si vous travaillez sur le démon et son code supporté ou si `guix-daemon` n'est pas déjà lancé sur votre système, vous pouvez le lancer directement depuis l'arborescence de construction<sup>1</sup> :

```
$ sudo -E ./pre-inst-env guix-daemon --build-users-group=guixbuild
```

Le script `pre-inst-env` paramètre toutes les variables d'environnement nécessaires, dont `PATH` et `GUILE_LOAD_PATH`.

Remarquez que `./pre-inst-env guix pull` ne met *pas* à jour l'arborescence des sources locale ; cela met seulement à jour le lien symbolique de `~/.config/guix/current` (voir Section 6.6 [Invoquer `guix pull`], page 63). Lancez `git pull` à la place si vous voulez mettre à jour votre arborescence des source locale.

Parfois, surtout si vous avez récemment mis à jour votre dépôt, lancer `./pre-inst-env` affichera des messages semblables à l'exemple suivant :

```
;;; note: source file /home/user/projects/guix/guix/progress.scm
;;;      newer than compiled /home/user/projects/guix/guix/progress.go
```

Ce n'est qu'une remarque et vous pouvez l'ignorer sans problème. Vous pouvez vous débarrasser de ce message en lançant `make -j4`. Jusqu'à ce que vous fassiez cela, Guile sera un tout petit peu plus lent parce qu'il doit interpréter le code au lieu d'utiliser les fichiers objets Guile (`.go`) précompilés.

Vous pouvez lancer `make` automatiquement quand vous travaillez avec `watchexec` du paquet `watchexec`. Par exemple, pour construire de nouveau à chaque fois que vous modifiez un fichier de paquet lancez `'watchexec -w gnu/packages -- make -j4'`.

## 22.3 La configuration parfaite

La configuration parfaite pour travailler sur Guix est simplement la configuration parfaite pour travailler en Guile (voir Section "Using Guile in Emacs" dans *Guile Reference Manual*). Tout d'abord, vous avez besoin de mieux qu'un éditeur de texte, vous avez besoin de Emacs (<http://www.gnu.org/software/emacs>), amélioré par le superbe Geiser (<http://nongnu.org/geiser/>). Pour paramétrer cela, lancez :

```
guix install emacs guile emacs-geiser emacs-geiser-guile
```

Geiser permet le développement interactif et incrémental depuis Emacs : la compilation du code et son évaluation depuis les tampons, l'accès à la documentation en ligne (docs-strings), la complétion sensible au contexte, `M-.` pour sauter à la définition d'un objet, un REPL pour tester votre code, et bien plus (voir Section "Introduction" dans *Geiser User Manual*). Pour travailler confortablement sur Guix, assurez-vous de modifier le chemin de chargement de Guile pour qu'il trouve les fichiers source de votre dépôt :

```
;; Si l'extrait est dans ~/src/guix.
(with-eval-after-load 'geiser-guile
  (add-to-list 'geiser-guile-load-path "~/src/guix"))
```

Pour effectivement éditer le code, Emacs a déjà un très bon mode Scheme. Mais en plus de ça, vous ne devez pas rater Paredit (<http://www.emacswiki.org/emacs/ParEdit>). Il

<sup>1</sup> Le drapeau `-E` de `sudo` garantit que `GUILE_LOAD_PATH` est correctement configuré de sorte que `guix-daemon` et les outils qu'il utilise puissent trouver les modules Guile dont ils ont besoin.

fournit des fonctionnalités pour opérer directement sur l'arbre de syntaxe, comme relever une s-expression ou l'envelopper, absorber ou rejeter la s-expression suivante, etc.

Nous fournissons aussi des modèles pour les messages de commit git courants et les définitions de paquets dans le répertoire `etc/snippets`. Ces modèles s'utilisent pour étendre de courtes phrases d'amorce en des bouts de texte interactifs. Si vous utilisez YASnippet (<http://joaotavora.github.io/yasnippet/>), vous voudrez peut-être ajouter le répertoire des modèles, `etc/snippets/yas`, dans la variable d'environnement `yas-snippet-dirs`. Si vous utilisez Tempel (<https://github.com/minad/tempel/>), vous voudrez peut-être ajouter le chemin `etc/snippets/tempel/*` dans la variable `tempel-path` d'Emacs.

```
;; En supposant que le dépôt git de Guix est dans ~/src/guix.
;; Configuration Yasnippet
(with-eval-after-load 'yasnippet
  (add-to-list 'yas-snippet-dirs "~/src/guix/etc/snippets/yas"))
;; Configuration Tempel
(with-eval-after-load 'tempel
  ;; S'assure que tempel-path est une liste --- elle peut aussi être une chaîne.
  (unless (listp 'tempel-path)
    (setq tempel-path (list tempel-path)))
  (add-to-list 'tempel-path "~/src/guix/etc/snippets/tempel/*"))
```

Les extraits de messages de commit dépendent de Magit (<https://magit.vc/>) pour afficher les fichiers sélectionnés. Lors de la modification d'un message de commit, tapez `add` suivi de `TAB` pour insérer un modèle de message de commit pour ajouter un paquet ; tapez `update` suivi de `TAB` pour insérer un modèle pour la mise à jour d'un paquet ; tapez `https` suivi de `TAB` pour insérer un modèle pour le changement à HTTPS de l'URI de la page d'accueil.

L'extrait principal pour `scheme-mode` est lancé en tapant `package...` suivi par `TAB`. Cet extrait insère aussi la chaîne de déclenchement `origin...`, qui peut aussi être étendue. L'extrait `origin` lui-même peut aussi insérer des chaînes de déclenchement qui finissent sur `...`, qui peuvent aussi être étendues.

De plus, nous fournissons l'insertion et la mise à jour automatique d'une ligne de copyright dans `etc/copyright.el`. Vous voudrez sans doute indiquer votre nom complet, adresse de courriel et charger le fichier.

```
(setq user-full-name "Alice Doe")
(setq user-mail-address "alice@mail.org")
;; En supposant que le dépôt Guix est dans ~/src/guix.
(load-file "~/src/guix/etc/copyright.el")
```

Pour insérer une ligne de copyright à la ligne actuelle invoquez `M-x guix-copyright`.

Pour mettre à jour une ligne de copyright vous devez spécifier un `copyright-names-regexp`.

```
(setq copyright-names-regexp
  (format "%s <%s>" user-full-name user-mail-address))
```

Vous pouvez vérifier si votre ligne de copyright est à jour en évaluant `M-x copyright-update`. Si vous voulez le faire automatiquement après chaque sauvegarde de tampon ajoutez `(add-hook 'after-save-hook 'copyright-update)` dans Emacs.

## 22.4 Consignes d’empaquetage

La distribution GNU est jeune et vos paquets préférés peuvent manquer. Cette section décrit comment vous pouvez aider à agrandir la distribution.

Les paquets de logiciels libres sont habituellement distribués sous forme *d’archives de sources* — typiquement des fichiers `.tar.gz` contenant tous les fichiers sources. Ajouter un paquet à la distribution signifie en substance deux choses : ajouter une *recette* qui décrit comment construire le paquet, avec une liste d’autres paquets requis pour le construire, et ajouter des *métadonnées de paquet* avec la recette, comme une description et une licence.

Dans Guix, toutes ces informations sont incorporées dans les *définitions de paquets*. Les définitions de paquets fournissent une vue de haut-niveau du paquet. Elles sont écrites avec la syntaxe du langage de programmation Scheme ; en fait, pour chaque paquet nous définissons une variable liée à la définition et exportons cette variable à partir d’un module (voir Section 9.1 [Modules de paquets], page 104). Cependant, il n’est *pas* nécessaire d’avoir une connaissance approfondie du Scheme pour créer des paquets. Pour plus d’informations sur les définitions des paquets, voir Section 9.2 [Définition des paquets], page 105.

Une fois une définition de paquet en place, stocké dans un fichier de l’arborescence des sources de Guix, il peut être testé avec la commande `guix build` (voir Section 10.1 [Invoquer `guix build`], page 180). Par exemple, en supposant que le nouveau paquet s’appelle `gnew`, vous pouvez lancer cette commande depuis l’arborescence de construction de Guix (voir Section 22.2 [Lancer Guix avant qu’il ne soit installé], page 670) :

```
./pre-inst-env guix build gnew --keep-failed
```

Utiliser `--keep-failed` rend facile le débogage des échecs car il fournit l’accès à l’arborescence de construction qui a échoué. Une autre sous-commande utile pour le débogage est `--log-file`, pour accéder au journal de construction.

Si le paquet n’est pas connu de la commande `guix`, il se peut que le fichier source ait une erreur de syntaxe, ou qu’il manque une clause `define-public` pour exporter la variable du paquet. Pour comprendre cela, vous pouvez charger le module depuis Guile pour avoir plus d’informations sur la véritable erreur :

```
./pre-inst-env guile -c '(use-modules (gnu packages gnew))'
```

Une fois que votre paquet est correctement construit, envoyez-nous un correctif (voir Chapitre 22 [Contribuer], page 668). Enfin, si vous avez besoin d’aide, nous serons ravis de vous aider aussi. Une fois que le correctif soumis est commité dans le dépôt Guix, le nouveau paquet est automatiquement construit sur les plate-formes supportées par notre système d’intégration continue (<https://ci.guix.gnu.org>).

On peut obtenir la nouvelle définition du paquet simplement en lançant `guix pull` (voir Section 6.6 [Invoquer `guix pull`], page 63). Lorsque [ci.guix.gnu.org](https://ci.guix.gnu.org) a fini de construire le paquet, l’installation du paquet y télécharge automatiquement les binaires (voir Section 6.3 [Substituts], page 54). La seule intervention humaine requise est pendant la revue et l’application du correctif.

### 22.4.1 Liberté logiciel

Le système d’exploitation GNU a été développé pour que les utilisatrices et utilisateurs puissent utiliser leur ordinateur en toute liberté. GNU est un *logiciel libre*, ce qui signifie que chaque personne l’utilisant bénéficie des quatre libertés essentielles (<https://www.gnu.org/licenses/olfs-fdl.html>).

[gnu.org/philosophy/free-sw.fr.html](http://gnu.org/philosophy/free-sw.fr.html)) : exécuter le programme, étudier et modifier le programme sous sa forme source, redistribuer des copies exactes et distribuer les versions modifiées. Les paquets qui se trouvent dans la distribution GNU ne fournissent que des logiciels qui respectent ces quatre libertés.

En plus, la distribution GNU suit les recommandations pour les distributions systèmes libres (<https://www.gnu.org/distros/free-system-distribution-guidelines.html>). Entre autres choses, ces recommandations rejettent les micrologiciels non libres, les recommandations de logiciels non libres et discute des façon de gérer les marques et les brevets.

Certaines sources amont autrement parfaitement libres contiennent une petite partie facultative qui viole les recommandations ci-dessus, par exemple car cette partie est du code non-libre. Lorsque cela arrive, les éléments en question sont supprimés avec des correctifs ou des bouts de codes appropriés dans la forme `origin` du paquet (voir Section 9.2 [Définition des paquets], page 105). De cette manière, `guix build --source` renvoie la source « libérée » plutôt que la source amont sans modification.

### 22.4.2 Conventions de nommage

Un paquet a en fait deux noms qui lui sont associés. Premièrement il y a le nom de la *variable Scheme*, celui qui suit `define-public`. Par ce nom, le paquet peut se faire connaître par le code Scheme, par exemple comme entrée d'un autre paquet. Deuxièmement, il y a la chaîne dans le champ `name` d'une définition de paquet. Ce nom est utilisé par les commandes de gestion des paquets comme `guix package` et `guix build`.

Les deux sont habituellement les mêmes et correspondent à la conversion en minuscule du nom du projet choisi en amont, où les underscores sont remplacés par des tirets. Par exemple, GNUnet est disponible en tant que `gnunet` et `SDL_net` en tant que `sdl-net`.

Une exception notable à cette règle est lorsque le nom du projet consiste en un unique caractère, ou si un projet maintenu plus ancien avec le même nom existe déjà — indépendamment de s'il a été empaqueté dans Guix on nom. Utilisez votre bon sens pour éviter de rendre ces noms ambigües et de leur enlever leur sens. Par exemple, le paquet Guix pour le shell nommé « s » en amont est `s-shell` et *pas s*. N'hésitez pas à demander si vous êtes en panne d'inspiration.

Nous n'ajoutons pas de préfixe `lib` aux bibliothèques de paquets, à moins qu'il ne fasse partie du nom officiel du projet. Mais voir Section 22.4.7 [Modules Python], page 678, et Section 22.4.8 [Modules Perl], page 679, pour des règles spéciales concernant les modules pour les langages Python et Perl.

Les noms de paquets de polices sont gérés différemment, voir Section 22.4.12 [Polices de caractères], page 682.

### 22.4.3 Numéros de version

Nous n'incluons en général que la dernière version d'un projet de logiciel libre donné. Mais parfois, par exemple pour des versions incompatibles de bibliothèques, deux (ou plus) versions du même paquet sont requises. Elles ont besoin d'un nom de variable Scheme différent. Nous utilisons le nom défini dans Section 22.4.2 [Conventions de nommage], page 674, pour la version la plus récente ; les versions précédentes utilisent le même nom, suffixé par `-` et le plus petit préfixe du numéro de version qui permet de distinguer deux versions.

Le nom dans la définition du paquet est le même pour toutes les versions d'un paquet et ne contient pas de numéro de version.

Par exemple, les version 2.24.20 et 3.9.12 de GTK+ peuvent être inclus de cette manière :

```
(define-public gtk+
  (package
    (name "gtk+")
    (version "3.9.12")
    ...))
(define-public gtk+-2
  (package
    (name "gtk+")
    (version "2.24.20")
    ...))
```

Si nous voulons aussi GTK+ 3.8.2, cela serait inclus de cette manière

```
(define-public gtk+-3.8
  (package
    (name "gtk+")
    (version "3.8.2")
    ...))
```

Parfois, nous incluons des paquets provenant d'instantanés de systèmes de contrôle de version (VCS) au lieu de versions publiées formellement. Cela devrait rester exceptionnel, car c'est le rôle des développeurs amont de spécifier quel est la version stable. Cependant, c'est parfois nécessaire. Donc, que faut-il mettre dans le champ `version` ?

Clairement, nous devons rendre l'identifiant de commit de l'instantané du VCS visible dans la version, mais nous devons aussi nous assurer que la version augmente de manière monotone pour que `guix package --upgrade` puisse déterminer quelle version est la plus récente. Comme les identifiants de commits, notamment avec Git, n'augmentent pas, nous ajoutons un numéro de révision qui nous augmentons à chaque fois que nous mettons à jour vers un nouvel instantané. La chaîne qui en résulte ressemble à cela :

```
2.0.11-3.cabba9e
  ^      ^      ^
  |      |      |  `-- ID du commit en amont
  |      |
  |      |  `--- révision du paquet Guix
  |
dernière version en amont
```

C'est une bonne idée de tronquer les identifiants dans le champ `version` à disons 7 caractères. Cela évite un problème esthétique (en supposant que l'esthétique ait un rôle à jouer ici) et des problèmes avec les limites de l'OS comme la longueur maximale d'un shebang (127 octets pour le noyau Linux). Il y a des fonctions auxiliaires pour faire cela avec les paquets qui utilisent `git-fetch` ou `hg-fetch` (voir plus bas). Il vaut mieux cependant utiliser l'identifiant de commit complet dans `origin`, pour éviter les ambiguïtés. Une définition de paquet peut ressembler à ceci :

```
(define my-package
```

```
(let ((commit "c3f29bc928d5900971f65965feaae59e1272a3f7")
      (revision "1")) ;révision du paquet Guix
  (package
    (version (git-version "0.9" revision commit))
    (source (origin
              (method git-fetch)
              (uri (git-reference
                    (url "git://example.org/my-package.git")
                    (commit commit)))
              (sha256 (base32 "1mbikn..."))
              (file-name (git-file-name name version))))
    ;; ...
  )))
```

`git-version` *VERSION REVISION COMMIT* [Procédure Scheme]

Renvoie la chaîne de version pour les paquets qui utilisent `git-fetch`.

```
(git-version "0.2.3" "0" "93818c936ee7e2f1ba1b315578bde363a7d43d05")
⇒ "0.2.3-0.93818c9"
```

`hg-version` *VERSION REVISION CHANGESET* [Procédure Scheme]

Renvoie la chaîne de version pour les paquets qui utilisent `hg-fetch`. Cela fonctionne de la même manière que `git-fetch`.

#### 22.4.4 Synopsis et descriptions

Comme nous l'avons vu avant, chaque paquet dans GNU Guix contient un résumé et une description (voir Section 9.2 [Définition des paquets], page 105). Les résumés et les descriptions sont importants : ce sont eux que recherche `guix package --search`, et c'est une source d'informations cruciale pour aider les utilisateur·rices à déterminer si un paquet donné correspond à leurs besoins. En conséquence, il convient de prêter attention à leur contenu lorsqu'on travaille sur un paquet.

Les résumés doivent commencer par une lettre capitale et ne doit pas finir par un point. Ils ne doivent pas commencer par « a » ou « the » (« un » ou « le/la »), ce qui n'apporte généralement rien ; par exemple, préférez « File-frobbing tool » (« Outil de frobage de fichier ») à « A tool that frobs file » (« Un outil qui frobe les fichiers »). Le résumé devrait dire ce que le paquet est — p. ex. « Utilitaire du cœur de GNU (fichier, text, shell) » — ou ce à quoi il sert — p. ex. le résumé de `grep` est « Affiche des lignes correspondant à un motif ».

Gardez à l'esprit que le résumé doit avoir du sens pour un large public. Par exemple « Manipulation d'alignements au format SAM » peut avoir du sens pour un bioinformaticien chevronné, mais n'aidera pas ou pourra perdre une audience de non-spécialistes. C'est une bonne idée de créer un résumé qui donne une idée du domaine d'application du paquet. Dans cet exemple, cela donnerait « Manipulation d'alignements de séquences de nucléotides », ce qui devrait donner une meilleure idée à la personne qui le lit pour savoir si c'est ce qu'elle recherche.

Les descriptions devraient faire entre cinq et dix lignes. Utilisez des phrases complètes, et évitez d'utiliser des acronymes sans les introduire d'abord. Évitez les expressions commerciales comme « world-leading », « industrial-strength » et « next-generation » et évitez

les superlatifs comme « the most advanced » — ils ne sont pas utiles aux personnes qui cherchent un paquet et semblent même un peu suspects. À la place, essayez d’être factuels, en mentionnant les cas d’utilisation et les fonctionnalités.

Les descriptions peuvent inclure du balisage Texinfo, ce qui est utile pour introduire des ornements comme `@code` ou `@dfn`, des listes à points ou des hyperliens (voir Section “Overview” dans *GNU Texinfo*). Cependant soyez prudents lorsque vous utilisez certains symboles, par exemple ‘@’ et les accolades qui sont les caractères spéciaux de base en Texinfo (voir Section “Special Characters” dans *GNU Texinfo*). Les commandes et outils comme `guix show` prennent en charge le rendu.

Les résumés et les descriptions sont traduits par des volontaires sur Weblate (<https://translate.fedoraproject.org/projects/guix/packages>) pour que le plus de personnes possible puissent les lire dans leur langue natale. Les interfaces les recherchent et les affichent dans la langue spécifiée par le paramètre de régionalisation actuel.

Pour permettre à `xgettext` de les extraire comme des chaînes traduisibles, les résumés et les descriptions *doivent être des chaînes littérales*. Cela signifie que vous ne pouvez pas utiliser `string-append` ou `format` pour construire ces chaînes :

```
(package
  ;; ...
  (synopsis "Ceci est traduisible")
  (description (string-append "Ceci n'est " "*pas*" " traduisible.")))
```

La traduction demande beaucoup de travail, faites donc d’autant plus attention à vos résumés et descriptions lorsque vous développez un paquet car chaque changement peut demander du de travail de la part des traducteur·rices. Pour les aider, il est possible de donner des recommandations ou des instructions qu’ils et elles pourront voir en insérant des commentaires spéciaux comme ceci (voir Section “xgettext Invocation” dans *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 Substituts ou Phases

La différence entre l’utilisation d’un bout de code (snippet) dans `origin` et une phase de construction pour modifier les sources d’un paquet peut être particulièrement subtile. Les bouts de code dans `origin` sont en général utilisés pour supprimer des fichiers indésirables, comme des bibliothèques incluses, des sources non libres ou pour appliquer de simples substitutions. La source dérivée d’un objet `origin` devrait produire une source qui peut être utilisée pour construire le paquet sur n’importe quel système pris en charge par le paquet en amont (c.-à-d., être la source correspondante). En particulier, les bouts de code dans `origin` ne doivent pas inclure d’éléments du dépôt dans les sources ; de telles corrections devraient plutôt être faites avec une phase de construction. Référez-vous à la documentation sur l’enregistrement `origin` pour plus d’information (voir Section 9.2.2 [référence de `origin`], page 113).

### 22.4.6 Paquets Emacs

Les paquets Emacs devraient plutôt utiliser le système de construction Emacs (voir [emacs-build-system], page 141), pour l’uniformité et les bénéfices apportés par ses phases de



construction, comme la génération automatique du fichier autoloade et la compilation des sources. Comme il n'y a pas de manière standardisée de lancer une suite de tests pour les paquets Emacs, les tests sont désactivés par défaut. Lorsqu'une suite de tests est disponible, elle devrait être activée en mettant l'argument `#:tests?` à `#true`. Par défaut, la commande pour lancer les tests est `make check`, mais on peut spécifier une autre commande via l'argument `#:test-command`. L'argument `#:test-command` attend une liste contenant une commande et ses arguments, à invoquer pendant la phase `check`.

Les dépendances Elisp des paquets Emacs sont en général fournies dans `propagated-inputs` lorsqu'elles sont requises à l'exécution. Comme pour les autres paquets, les dépendances de test et de construction doivent être spécifiées dans `native-inputs`.

Les paquets Emacs dépendent parfois de répertoires de ressources qui devraient être installés avec les fichiers Elisp. L'argument `#:include` peut être utilisé pour cela, en spécifiant une liste d'expressions régulières. La bonne pratique pour utiliser l'argument `#:include` est d'étendre plutôt que de remplacer sa valeur par défaut (accessible via la variable `%default-include`). Par exemple, un paquet d'extension `yasnippet` inclut en général un répertoire `snippets`, qui pourrait être copié vers le répertoire d'installation avec :

```
#:include (cons "~snippets/" %default-include)
```

Lorsque vous rencontrez des problèmes, il est utile de vérifier la présence de l'en-tête d'extension `Package-Requires` dans le fichier source principal du paquet, et si les dépendances et leurs versions qui y sont listées sont satisfaites.

### 22.4.7 Modules Python

Nous incluons actuellement Python 2 et Python 3, sous les noms de variables Scheme `python-2` et `python` comme expliqué dans Section 22.4.3 [Numéros de version], page 674. Pour éviter la confusion et les problèmes de noms avec d'autres langages de programmation, il semble désirable que le nom d'un paquet pour un module Python contienne le mot `python`.

Certains modules ne sont compatibles qu'avec une version de Python, d'autres avec les deux. Si le paquet `Toto` ne compile qu'avec Python 3, on le nomme `python-toto`. S'il est compilé avec Python 2, on le nomme `python2-toto`. Les paquets ne devraient être ajoutés que lorsqu'ils sont nécessaires ; nous n'ajoutons pas les variantes Python 2 du paquet à moins qu'on ne les utilise ensuite.

Si un projet contient déjà le mot `python`, on l'enlève, par exemple le module `python-dateutil` est packagé sous les noms `python-dateutil` et `python2-dateutil`. Si le nom du projet commence par `py` (p. ex. `pytz`), on le garde et on le préfixe comme décrit ci-dessus.

**Remarque:** Il y a actuellement deux systèmes de construction différents pour les paquets Python dans Guix : `python-build-system` et `pyproject-build-system`. Pendant longtemps, les paquets Python étaient construits à partir d'un fichier `setup.py` à la spécification informelle. Cela fonctionnait très bien, étant donné le succès de Python, mais il était difficile de construire des outils par dessus. En conséquence, de nombreuses alternatives ont émergé et la communauté a finalement retenu un standard formel (<https://peps.python.org/pep-0517>) pour spécifier les prérequis de construction. `pyproject-build-system` est l'implémentation de ce standard par Guix. Il est considéré comme « expérimental » dans le sens où il ne prend pas encore en charge les divers *modeurs de construction* de PEP-517, mais nous vous encourageons à l'essayer pour

les nouveaux paquets Python et à rapporter tout problème que vous rencontrez. Il finira par être rendu obsolète et fusionné dans *python-build-system*.

### 22.4.7.1 Spécifier les dépendances

Les informations de dépendances pour les paquets Python se trouvent généralement dans l'arborescence des source du paquet, avec plus ou moins de précision : dans le fichier `pyproject.toml`, le fichier `setup.py`, dans `requirements.txt` ou dans `tox.ini` (ce dernier surtout pour les dépendances de test).

Votre mission, lorsque vous écrivez une recette pour un paquet Python, est de faire correspondre ces dépendances au bon type « d'entrée » (voir Section 9.2.1 [référence de package], page 108). Bien que l'importeur `pypi` fasse du bon boulot (voir Section 10.5 [Invoquer `guix import`], page 197), vous devriez vérifier la liste suivant pour déterminer où va telle dépendance.

- Nous empaquetons Python avec `setuptools` et `pip` installé par défaut. Cela va changer, et nous vous encourageons à utiliser `python-toolchain` si vous voulez un environnement de construction pour Python.

`guix lint` vous avertira si `setuptools` ou `pip` sont ajoutés en entrées natives car ils ne sont en général pas nécessaires.

- Les dépendances Python requises à l'exécutions vont dans `propagated-inputs`. Elles sont typiquement définies dans le mot-clef `install_requires` dans `setup.py` ou dans le fichier `requirements.txt`.
- Les paquets Python requis uniquement à la construction — p. ex. ceux listés dans `build-system.requires` dans `pyproject.toml` ou dans le mot-clef `setup_requires` de `setup.py` — ou les dépendances seulement pour les tests — p. ex. ceux dans `tests_require` — vont dans `native-inputs`. La raison est qu'ils n'ont pas besoin d'être propagés car ils ne sont pas requis à l'exécution et dans le cas d'une compilation croisée, c'est l'entrée « native » qu'il nous faut.

Les cadriciels de tests `pytest`, `mock` et `nose` sont des exemples. Bien sûr si l'un de ces paquets est aussi requis à l'exécution, il doit aller dans `propagated-inputs`.

- Tout ce qui ne tombe pas dans les catégories précédentes va dans `inputs`, par exemple des programmes pour des bibliothèques C requises pour construire des paquets Python avec des extensions C.
- Si un paquet Python a des dépendances facultatives (`extras_require`), c'est à vous de décider de les ajouter ou non, en fonction du ratio entre utilité et complexité (voir Section 22.6 [Envoyer des correctifs], page 683).

### 22.4.8 Modules Perl

Les programmes Perl utiles en soit sont nommés comme les autres paquets, avec le nom amont en minuscule. Pour les paquets Perl contenant une seule classe, nous utilisons le nom de la classe en minuscule, en remplaçant les occurrences de `::` par des tirets et en préfixant le tout par `perl-`. Donc la classe `XML::Parser` devient `perl-xml-parser`. Les modules contenant plusieurs classes gardent leur nom amont en minuscule et sont aussi préfixés par `perl-`. Ces modules tendent à avoir le mot `perl` quelque part dans leur nom, que nous supprimons en faveur du préfixe. Par exemple, `libwww-perl` devient `perl-libwww`.

### 22.4.9 Paquets Java

Les programmes Java utiles en soit sont nommés comme les autres paquets, avec le nom amont en minuscule.

Pour éviter les confusions et les problèmes de nom avec d'autres langages de programmation, il est désirable que le nom d'un paquet Java soit préfixé par `java-`. Si un projet contient déjà le mot `java`, nous le supprimons, par exemple le paquet `ngsjava` est empaqueté sous le nom `java-ngs`.

Pour les paquets java contenant une seule classe ou une petite hiérarchie de classes, nous utilisons le nom de la classe en minuscule, en remplaçant les occurrences de `.` par des tirets et en préfixant le tout par `java-`. Donc la classe `apache.commons.cli` devient `java-apache-commons-cli`.

### 22.4.10 Paquets Rust

Les programmes Rust utiles en soit sont nommés comme les autres paquets, avec le nom amont en minuscule.

Pour éviter des collisions de noms nous préfixons les autres paquets avec le préfixe `rust-`. Vous devrez changer le nom en minuscule et garder les tirets en place.

Dans l'écosystème rust il est courant d'avoir plusieurs versions incompatibles d'un paquet utilisées en même temps, donc toutes les définitions des paquets devraient avoir un suffixe de version. Le suffixe de version est le nombre le plus à gauche qui n'est pas un zéro (et tous les zéros précédents, évidemment). Cela suit les conventions de version « caret » de Cargo. Par exemple `rust-clap-2`, `rust-rand-0.6`.

À cause de la difficulté à réutiliser des paquets rust en entrées précompilées pour d'autres paquets, le système de construction Cargo (voir Section 9.5 [Systèmes de construction], page 124) présente les mots-clés `#:cargo-inputs` et `cargo-development-inputs` en argument du système de construction. Vous pouvez y penser comme les équivalents de `propagated-inputs` et `native-inputs`. Les `dependencies` et `build-dependencies` de Rust devraient aller dans `#:cargo-inputs` et `dev-dependencies` dans `#:cargo-development-inputs`. Si un paquet Rust se lie à d'autres bibliothèques alors vous devriez utiliser l'emplacement `inputs` standard et compagnie.

Vous devriez faire attention à vous assurer que la bonne version des dépendances est utilisée ; pour cela nous essayons d'éviter de passer les tests ou d'utiliser `#:skip-build?` lorsque c'est possible. Bien sûr ce n'est pas toujours possible, comme le paquet peut être développé sur un autre système d'exploitation, dépendre d'une fonctionnalité du compilateur Rust Nightly ou la suite de test atrophiée depuis la sortie.

### 22.4.11 Paquets elm

Les applications Elm peuvent être nommée comme les autres logiciels : leur nom n'a pas besoin de mentionner Elm.

Les paquets au sens de Elm (voir `elm-build-system` dans Section 9.5 [Systèmes de construction], page 124) doivent utiliser un nom au format `auteur/projet`, où à la fois l'*auteur* et le *projet* peuvent contenir des tirets, et *auteur* contient parfois des lettres majuscules.

Pour former le nom de paquet Guix à partir du nom en amont, nous suivons une convention similaire à celle des paquets Python (voir Section 22.4.7 [Modules Python], page 678), en ajoutant le préfixe `elm-` à moins que le nom ne commence déjà par `elm-`.

Dans la plupart des cas on peut reconstruire le nom d'un paquet Elm en amont avec une heuristique, mais, comme la conversion vers le nom Guix implique une perte d'information, cela n'est pas toujours possible. Vous devrez faire attention à ajouter la propriété `'upstream-name` si nécessaire pour que `'guix import elm` fonctionne correctement (voir Section 10.5 [Invoquer `guix import`], page 197). Les scénarios les plus importants où il faut spécifier explicitement le nom en amont sont :

1. Lorsque l'*auteur* est `elm` et que le *projet* contient une ou plusieurs tirets, comme `elm/virtual-dom` et
2. Lorsque l'*auteur* contient des tirets ou des lettres majuscule, comme `Elm-Canvas/raster-shapes` — à moins que l'*auteur* ne soit `elm-explorations`, qui est géré à part, donc un paquet comme `elm-explorations/markdown` n'a *pas* besoin d'utiliser la propriété `'upstream-name`.

Le module (`guix build-system elm`) fournit les utilitaires suivants pour travailler avec les noms et les conventions associées :

`elm-package-origin` *elm-name* *version* *hash* [Procédure Scheme]

Renvoie une origine Git en utilisant le nom et les tags du dépôt, requis pour publier un paquet Elm avec le nom amont *elm-name* à la version *version* avec la somme de contrôle sha256 *hash*.

Par exemple :

```
(package
  (name "elm-html")
  (version "1.0.0")
  (source
    (elm-package-origin
      "elm/html"
      version
      (base32 "15k1679ja57vvlpinpv06znmrxy091bhzfkdzc89i01qa8c4gb4a"))))
```

`elm->package-name` *elm-name* [Procédure Scheme]

Renvoie le nom de paquet Guix pour un paquet Elm avec le nom amont *elm-name*.

Remarquez qu'il y a plus d'un *elm-name* possible pour un résultat de `elm->package-name` donné.

`guix-package->elm-name` *paquet* [Procédure Scheme]

Étant donné un *paquet* Elm, renvoie le nom amont éventuellement inféré, ou `#f` si le nom amont n'est pas spécifié via la propriété `'upstream-name` et qu'il ne peut pas être deviné par `infer-elm-package-name`.

`inferior-elm-package-name` *guix-name* [Procédure Scheme]

Étant donné le *guix-name* d'un paquet Elm, renvoie le nom amont inféré, ou `#f` si le nom amont ne peut pas être inféré. Si le résultat n'est pas `#f`, l'envoyer à `elm->package-name` produirait *guix-name*.

### 22.4.12 Polices de caractères

Pour les polices qui ne sont en général pas installées pour être utilisées pour du traitement de texte, ou qui sont distribuées en tant que partie d'un paquet logiciel plus gros, nous nous appuyons sur les règles générales pour les logiciels ; par exemple, cela s'applique aux polices livrées avec le système X.Org ou les polices qui font partie de TeX Live.

Pour rendre plus facile la recherche par l'utilisatrice ou l'utilisateur, les noms des autres paquets contenant seulement des polices sont construits ainsi, indépendamment du nom du paquet en amont.

Le nom d'un paquet contenant une unique famille de polices commence par `font-` ; il est suivi du nom du fondeur et d'un tiret - si le fondeur est connu, et du nom de la police, dont les espaces sont remplacés par des tirets (et comme d'habitude, toutes les lettres majuscules sont transformées en minuscules). Par exemple, la famille de polices Gentium de SIL est empaqueté sous le nom `font-sil-gentium`.

Pour un paquet contenant plusieurs familles de polices, le nom de la collection est utilisée à la place du nom de la famille. Par exemple les polices Liberation consistent en trois familles, Liberation Sans, Liberation Serif et Liberation Mono. Elles pourraient être empaquetées séparément sous les noms `font-liberation-sans` etc, mais comme elles sont distribuées ensemble sous un nom commun, nous préférons les empaqueter ensemble en tant que `font-liberation`.

Dans le cas où plusieurs formats de la même famille ou collection sont empaquetés séparément, une forme courte du format, préfixé d'un tiret est ajouté au nom du paquet. Nous utilisons `-ttf` pour les polices TrueType, `-otf` pour les polices OpenType et `-type1` pour les polices Type 1 de PostScript.

## 22.5 Style de code

En général notre code suit le Standard de Code GNU (voir *GNU Coding Standards*). Cependant, il ne parle pas beaucoup de Scheme, donc voici quelques règles supplémentaires.

### 22.5.1 Paradigme de programmation

Le code Scheme dans Guix est écrit dans un style purement fonctionnel. Le code qui s'occupe des entrées-sorties est une exception ainsi que les procédures qui implémentent des concepts bas-niveau comme la procédure `memoize`.

### 22.5.2 Modules

Les modules Guile qui sont sensés être utilisés du côté de la construction doivent se trouver dans l'espace de nom (`guix build ...`). Ils ne doivent pas se référer à d'autres modules Guix ou GNU. Cependant il est correct pour un module « côté hôte » de dépendre d'un module coté construction.

Les modules qui s'occupent du système GNU général devraient se trouver dans l'espace de nom (`gnu ...`) plutôt que (`guix ...`).

### 22.5.3 Types de données et reconnaissance de motif

La tendance en Lisp classique est d'utiliser des listes pour tout représenter et de naviguer dedans « à la main » (avec `car`, `cdr`, `cadr` et `compagnie`). Il y a plusieurs problèmes avec ce

style, notamment le fait qu'il soit dur à lire, source d'erreur et un obstacle aux rapports d'erreur bien typés.

Le code de Guix devrait définir des types de données appropriées (par exemple, avec `define-record-type*`) plutôt que d'abuser des listes. En plus, il devrait utiliser la recherche de motifs, via le module Guile (`ice-9 match`), surtout pour rechercher dans des listes (voir Section "Pattern Matching" dans *GNU Guile Reference Manual*) : la reconnaissance de motif pour les enregistrements se fait mieux avec `match-record` de (`guix records`), qui contrairement à `match`, vérifie le nom des champs à l'expansion des macros.

### 22.5.4 Formatage du code

Lorsque nous écrivons du code Scheme, nous suivons la sagesse commune aux programmeurs Scheme. En général, nous suivons les règles de style de Riastradh (<https://mumble.net/~campbell/scheme/style.txt>). Ce document décrit aussi les conventions utilisées dans le code de Guile. Il est bien pensé et bien écrit, alors n'hésitez pas à le lire.

Certaines formes spéciales introduites dans Guix comme la macro `substitute*` ont des règles d'indentation spécifiques. Elles sont définies dans le fichier `.dir-locals.el` qu'Emacs utilise automatiquement. Remarquez aussi qu'Emacs-Guix fournit le mode `guix-devel-mode` qui indente et colore le code Guix correctement (voir Section "Development" dans *The Emacs-Guix Reference Manual*).

Si vous n'utilisez pas Emacs, assurez-vous que votre éditeur connaisse ces règles. Pour indenter automatiquement une définition de paquet, vous pouvez aussi lancer :

```
./pre-inst-env guix style paquet
```

Voir Section 10.7 [Invoquer `guix style`], page 210, pour plus d'informations.

Si vous éditez du code avec Vim, nous recommandons de lancer `:set autoindent` pour que votre code soit automatiquement indenté au moment où vous l'entrez. En plus, `paredit.vim` ([https://www.vim.org/scripts/script.php?script\\_id=3998](https://www.vim.org/scripts/script.php?script_id=3998)) peut vous aider à gérer toutes ces parenthèses.

Nous demandons que toutes les procédures de premier niveau contiennent une chaîne de documentation. Ce prérequis peut être relâché pour les procédures privées simples dans l'espace de nom (`guix build ...`) cependant.

Les procédures ne devraient pas avoir plus de quatre paramètres positionnés. Utilisez des paramètres nommés pour les procédures qui prennent plus de quatre paramètres.

## 22.6 Envoyer des correctifs

Le développement se fait avec le système de contrôle de version Git. Ainsi, l'accès au dépôt n'est pas strictement nécessaire. Nous accueillons les contributions sous forme de correctifs produits par `git format-patch` envoyés sur la liste de diffusion `guix-patches@gnu.org` (voir Section "Submitting patches to a project" dans *Git User Manual*). Si vous voulez contribuer, nous vous encourageons à prendre un moment pour changer quelques options du dépôt Git pour améliorer la lisibilité des correctifs avant de commencer (voir Section 22.6.1 [Configurer Git], page 687). Si vous avez bien expérimenté la contribution à Guix, vous pouvez aussi regarder la section sur les accès en commit (voir Section 22.8 [Accès en commit], page 691).

Cette liste de diffusion est gérée par une instance Debbugs, qui nous permet de suivre les soumissions. Chaque message envoyé à cette liste se voit attribuer un nouveau numéro de suivi ; les gens peuvent ensuite répondre à cette soumission en envoyant un courriel à `NUMÉRO_PROBLÈME@debbugs.gnu.org`, où `NUMÉRO_PROBLÈME` est le numéro de suivi (voir Section 22.6.2 [Envoyer une série de correctifs], page 687).

Veillez écrire les messages de commit dans le format ChangeLog (voir Section “Change Logs” dans *GNU Coding Standards*) ; vous pouvez regarder l’historique des commits pour trouver des exemples.

Avant de soumettre un correctif qui ajoute ou modifie la définition d’un paquet, veuillez vérifier cette check-list :

1. Si les auteurs ou autrices du paquet logiciel fournissent une signature cryptographique pour l’archive, faites un effort pour vérifier l’authenticité de l’archive. Pour un fichier de signature GPG détaché, cela se fait avec la commande `gpg --verify`.
2. Prenez un peu de temps pour fournir un synopsis et une description adéquats pour le paquet. Voir Section 22.4.4 [Synopsis et descriptions], page 676, pour quelques lignes directrices.
3. Lancez `guix lint paquet`, où `paquet` est le nom du nouveau paquet ou du paquet modifié, et corrigez les erreurs qu’il rapporte (voir Section 10.8 [Invoquer guix lint], page 213).
4. Lancez `guix style paquet` pour formater la nouvelle définition du paquet en suivant les conventions du projet (voir Section 10.7 [Invoquer guix style], page 210).
5. Assurez-vous que le paquet se construise sur votre plate-forme avec `guix build paquet`.
6. Nous vous recommandons aussi d’essayer de construire le paquet sur les autres plate-formes prises en charge. Comme vous n’avez pas forcément accès aux plate-formes matérielles, nous vous recommandons d’utiliser le `qemu-binfmt-service-type` pour les émuler. Pour cela, ajoutez le module de service `virtualization` et le service suivant à la liste des services dans votre configuration de système d’exploitation :

```
(service qemu-binfmt-service-type
 (qemu-binfmt-configuration
  (platforms (lookup-qemu-platforms "arm" "aarch64"))))
```

Puis reconfigurez votre système.

Vous pourrez ensuite construire les paquets pour différentes plate-formes en spécifiant l’option `--system`. Par exemple pour construire le paquet « hello » pour les architectures `armhf` ou `aarch64`, vous devrez lancer les commandes suivantes, respectivement :

```
guix build --system=armhf-linux --rounds=2 hello
guix build --system=aarch64-linux --rounds=2 hello
```

7. Assurez-vous que le paquet n’utilise pas de copie groupée d’un logiciel déjà disponible dans un paquet séparé.

Parfois, les paquets incluent des copie du code source de leurs dépendances pour le confort des utilisateur·rices. Cependant, en tant que distribution, nous voulons nous assurer que ces paquets utilisent bien les copient que nous avons déjà dans la distribution si elles existent. Cela améliore l’utilisation des ressources (la dépendance n’est

construite et stockée qu'une seule fois) et permet à la distribution de faire des changements transversaux comme appliquer des correctifs de sécurité pour un paquet donné depuis un unique emplacement et qu'ils affectent tout le système, ce qu'empêchent les copies groupées.

8. Regardez le profil rapporté par `guix size` (voir Section 10.9 [Invoquer `guix size`], page 215). Cela vous permettra de remarquer des références à d'autres paquets qui ont été retenus sans que vous vous y attendiez. Il peut aussi aider à déterminer s'il faut découper le paquet (voir Section 6.4 [Des paquets avec plusieurs résultats], page 59) et quelles dépendances facultatives utiliser. En particulier, évitez d'ajouter `texlive` en dépendance : à cause de sa taille extrême, utilisez `texlive-tiny` ou `texlive-union` à la place.
9. Pour les changements important, vérifiez que les paquets qui en dépendent (s'ils existent) ne sont pas affectés par le changement ; `guix refresh --list-dependant paquet` vous aidera (voir Section 10.6 [Invoquer `guix refresh`], page 205).

Suivant le nombre de paquets dépendants et donc le nombre de reconstruction induites, les commits vont vers des branches différentes, suivant ces principes :

300 paquets dépendants ou moins

branche `master` (changements non-disruptifs).

entre 300 et 1 800 paquets dépendants

branche `staging` (changements non-disruptifs). Cette branche devrait être fusionnées dans `master` toutes les 6 semaines. Les changements par thèmes (par exemple une mise à jour de la pile GNOME) peuvent aller dans une branche spécifique (disons, `gnome-updates`). Il n'est pas forcément possible de construire ni utiliser cette branche jusque tard dans le processus de développement.

plus de 1 800 paquets dépendants

la branche `core-updates` (peut inclure des changements majeurs et potentiellement disruptifs). Cette branche devrait être fusionnée dans `master` tous les 6 mois environ. Il n'est pas forcément possible de construire ni utiliser cette branche jusque tard dans le processus de développement.

Toutes ces branches sont gérées par notre ferme de construction (<https://ci.guix.gnu.org>) et fusionnées dans `master` une fois que tout a été construit correctement. Cela nous permet de corriger des problèmes avant qu'ils n'atteignent les utilisateur·rices et réduit la fenêtre pendant laquelle les binaires pré-construits ne sont pas disponibles.

Lorsqu'on décide de commencer à construire la branche `staging` ou `core-updates`, on les fork et on les renomme avec le suffixe `-frozen`, et à partir de ce moment, seuls les corrections de bogues sont poussées sur les branches gelées. Les branches `core-updates` et `staging` restent ouvertes et acceptent des correctifs pour le prochain cycle. Demandez sur la liste de diffusion ou sur IRC si vous ne savez pas où envoyer un correctif.

10. Vérifiez si le processus de construction du paquet est déterministe. Cela signifie typiquement vérifier qu'une construction indépendante du paquet renvoie exactement le même résultat que vous avez obtenu, bit à bit.



Une manière simple de le faire est de reconstruire le paquet plusieurs fois à la suite sur votre machine (voir Section 10.1 [Invoquer `guix build`], page 180) :

```
guix build --rounds=2 mon-paquet
```

Cela est suffisant pour trouver une classe de non-déterminisme commune, comme l'horodatage ou des sorties générées aléatoirement dans le résultat de la construction.

Une autre option consiste à utiliser `guix challenge` (voir Section 10.12 [Invoquer `guix challenge`], page 226). Vous pouvez lancer la commande une fois que les paquets ont été committés et construits par `ci.guix.gnu.org` pour vérifier s'il obtient le même résultat que vous. Mieux encore : trouvez une autre machine qui peut le construire et lancez `guix publish`. Puisque la machine distante est sûrement différente de la vôtre, cela peut trouver des problèmes de non-déterminisme liés au matériel — par exemple utiliser une extension du jeu d'instruction — ou du noyau du système d'exploitation — par exemple se reposer sur `uname` ou les fichiers de `/proc`.

11. Lorsque vous écrivez de la documentation, utilisez une formulation au genre neutre lorsque vous vous référez à des personnes, comme le “they”, “their”, “them” singulier ([https://fr.wikipedia.org/wiki/They\\_singulier](https://fr.wikipedia.org/wiki/They_singulier)) (en anglais).
12. Vérifiez que votre correctif contienne seulement un ensemble de changements liés. Grouper des changements non liés ensemble rend la revue plus difficile et plus lente. Ajouter plusieurs paquet ou une mise à jour d'un paquet avec des corrections dans ce paquet sont des exemples de changements sans rapport.
13. Suivez nos règles de formatage de code, éventuellement en lançant le script `guix style` pour le faire automatiquement (voir Section 22.5.4 [Formatage du code], page 683).
14. Si possible, utilisez des miroirs dans l'URL des sources (voir Section 10.3 [Invoquer `guix download`], page 195). Utilisez des URL stable, pas des URL générées. Par exemple, les archives GitHub ne sont pas nécessairement identiques d'une génération à la suivante, donc il vaut mieux dans ce cas cloner le dépôt. N'utilisez pas le champ `name` dans l'URL : ce n'est pas très utile et si le nom change, l'URL sera probablement erronée.
15. Vérifiez si Guix compile (voir Section 22.1 [Construire depuis Git], page 668) et corrigez les avertissements, surtout ceux à propos de symboles manquants.
16. Assurez-vous que vos changements ne cassent pas Guix et simulez un `guix pull` avec :

```
guix pull --url=/path/to/your/checkout --profile=/tmp/guix.master
```

Lorsque vous envoyez un correctif à la liste de diffusion, utilisez ‘[PATCH] ...’ comme objet. Si votre correctif doit être appliqué à une autre branche que `master`, comme `core-updates`, spécifiez-le dans l'objet comme dans ‘[PATCH core-updates] ...’.

Vous pouvez utiliser votre client de messagerie ou la commande `git send-email` (voir Section 22.6.2 [Envoyer une série de correctifs], page 687). Nous préférons recevoir les correctifs dans des messages en texte clair, soit en ligne, soit sous forme de pièces jointes MIME. Il vous est conseillé de faire attention si votre client de messagerie modifie quoi que ce soit, comme des sauts de ligne ou des indentations, qui pourraient potentiellement casser les correctifs.

Attendez-vous à un délai entre le moment où vous envoyez votre tout premier correctif à `guix-patches@gnu.org` et le moment où le message sera reçu. Vous devez attendre de recevoir un message de confirmation avec le numéro de suivi assigné à votre correctif. Les confirmations suivantes ne seront pas retardées.

Lorsqu'un bogue est résolu, veuillez fermer le fil en envoyant un courriel à `NUMÉRO_PROBLÈME-done@debbugs.gnu.org`.

### 22.6.1 Configurer Git

Si vous ne l'avez pas encore fait, vous devriez indiquer un nom et une adresse de courriel qui seront associés à vos commits (voir Section “Telling Git your name” dans *Git User Manual*). Si vous souhaitez utiliser un nom ou une adresse différent pour les commits de ce dépôt, vous pouvez utiliser `git config --local`, ou modifier `.git/config` dans le dépôt au lieu de `~/.gitconfig`.

Nous fournissons quelques paramètres par défaut dans `etc/git/gitconfig` qui modifient la manière de générer des correctifs, pour les rendre plus lisibles et plus faciles à appliquer. Vous pouvez appliquer ces paramètres manuellement en les copiant dans `.git/config` dans votre dépôt, ou en disant à Git d'inclure le fichier complet :

```
git config --local include.path ../etc/git/gitconfig
```

À partir de maintenant, tout changement de `etc/git/gitconfig` prendra automatiquement effet.

Comme le premier correctif d'une série doit être envoyé séparément (voir Section 22.6.2 [Envoyer une série de correctifs], page 687), il peut être intéressant de dire à `git format-patch` de se charger de la création d'un fil de discussion à la place de `git send-email` :

```
git config --local format.thread shallow
git config --local sendemail.thread no
```

### 22.6.2 Envoyer une série de correctifs

#### Correctifs isolés

La commande `git send-email` est la meilleure manière d'envoyer aussi bien des correctifs isolés que des séries de correctifs (voir [Plusieurs correctifs], page 688) à la liste de diffusion de Guix. Envoyer des correctifs en pièce-jointe peut les rendre plus difficile à revoir dans certains clients de courriel et `git diff` ne stocke pas les métadonnées des commits.

**Remarque:** La commande `git send-email` est fournie par la sortie `send-email` du paquet `git`, c.-à-d. `git:send-email`.

La commande suivante créera un courriel de correctif à partir du dernier commit, l'ouvrira dans *EDITOR* ou *VISUAL* pour que vous puissiez le modifier et l'enverra à la liste de diffusion de Guix pour qu'il soit revu et fusionné :

```
$ git send-email -1 -a --base=auto --to=guix-patches@gnu.org
```

**Astuce:** Pour ajouter un préfixe à l'objet de votre correctif, vous pouvez utiliser l'option `--subject-prefix`. Le projet Guix utilise cela pour spécifier si un correctif est prévu pour une autre branche ou un autre dépôt que la branche `master` de `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
```

Le courriel de correctif contient une ligne de séparation à trois tirets après le message de commit. Vous pouvez « annoter » le correctif avec un texte explicatif en l'ajoutant sous

cette ligne. Si vous ne voulez pas annoter le courriel, vous pouvez enlever le drapeau `-a` (qui est le raccourci pour `--annotate`).

Le drapeau `--base=auto` ajoute automatiquement une note à la fin du correctif indiquant le commit sur lequel il est basé, ce qui facilite le rebasage et la fusion de votre correctif pour les mainteneurs.

Si vous voulez envoyer un correctif modifié, ne le renvoyez pas de cette manière et n'envoyez pas un correctif de « correction » à appliquer sur l'ancien ; à la place, utilisez `git commit -a` ou `git rebase` (<https://git-rebase.io>) pour modifier le commit, et utilisez l'adresse `NUMÉRO_PROBLÈME@debbugs.gnu.org` et le drapeau `-v` avec `git send-email`.

```
$ git commit -a
$ git send-email -1 -a --base=auto -v RÉVISION \
  --to=NUMÉRO_PROBLÈME@debbugs.gnu.org
```

Vous pouvez trouver `NUMÉRO_PROBLÈME` soit en cherchant sur l'interface mumi sur [issues.guix.gnu.org](https://issues.guix.gnu.org) le nom de votre correctif ou en lisant l'accusé de réception envoyé automatiquement par Debbugs en réponse aux bogues et correctifs entrants, qui contient le numéro de suivi.

## Notifier les équipes

Le script `etc/teams.scm` peut être utilisé pour notifier toutes les personnes qui sont intéressées par votre correctif (voir Section 22.6.3 [Équipes], page 689). Utilisez `etc/teams.scm list-teams` pour afficher toutes les équipes, pour décider quelles équipes votre correctif intéressera et utilisez `etc/teams.scm cc` pour afficher les drapeaux `git send-email` qui notifieront les membres des équipes appropriées ou utilisez `etc/teams.scm cc-members` pour automatiquement détecter les équipes appropriées.

## Plusieurs correctifs

Alors que `git send-email` seul suffira pour un unique correctif, un problème malheureux de Debbugs fait que vous devrez faire plus attention pour envoyer plusieurs correctifs : si vous les envoyez tous à l'adresse `guix-patches@gnu.org`, un nouveau ticket sera créé pour chaque correctif !

Lorsque vous envoyez une série de correctifs, il vaut mieux envoyer d'abord un « résumé » Git pour donner un aperçu de votre série de correctifs aux relecteur-ices. Vous pouvez créer un répertoire nommé `sortant` contenant à la fois votre série de correctifs et un résumé appelé `0000-cover-letter.patch` avec `git format-patch`.

```
$ git format-patch -NOMBRE_COMMITS -o sortant \
  --cover-letter --base=auto
```

Vous pouvez maintenant envoyer *seulement* le résumé à l'adresse `guix-patches@gnu.org` ce qui créera un ticket auquel vous pourrez envoyer le reste des correctifs.

```
$ git send-email outgoing/0000-cover-letter.patch -a \
  --to=guix-patches@debbugs.gnu.org \
  $(etc/teams.scm cc-members ...)
```

```
$ rm sortant/0000-cover-letter.patch # on ne veut pas le renvoyer !
```

Assurez-vous de modifier le courriel pour ajouter une ligne d'objet appropriée et un petit texte avant de l'envoyer. Remarquez le journal court et les statistiques de changements générés automatiquement sous votre texte.

Une fois que le logiciel de Debbugs a répondu à votre courriel de résumé, vous pouvez envoyer les correctifs à l'adresse de suivi nouvellement créée.

```
$ git send-email sortant/*.patch \
  --to=NUMÉRO_PROBLÈME@debbugs.gnu.org \
  $(etc/teams.scm cc-members ...)
$ rm -rf sortant # on n'en a plus besoin
```

Heureusement, cette danse avec `git format-patch` n'est pas nécessaire pour envoyer une série modifiée car un ticket existe déjà pour la suite de correctifs.

```
$ git send-email -NOMBRE_COMMITS \
  -vRÉVISION --base=auto \
  --to NUMÉRO_PROBLÈME@debbugs.gnu.org
```

Si besoin, vous pouvez utiliser `--cover-letter -a` pour envoyer un autre résumé, p. ex. pour expliquer ce qui a changé par rapport à l'ancienne révision et que ces changements sont nécessaires.

### 22.6.3 Équipes

Il y a plusieurs équipes qui prennent en charge différentes parties du code source de Guix. Pour lister toutes ces équipes, vous pouvez exécuter depuis un clone de Guix :

```
$ ./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>
...
```

Vous pouvez exécuter la commande suivante pour ajouter l'équipe Mentors en CC de votre série de correctifs :

```
$ git send-email --to NUMÉRO_PROBLÈME@debbugs.gnu.org $(./etc/teams.scm cc mentors) *
```

L'équipe ou les équipes appropriées peuvent aussi être inférées à partir des fichiers modifiés. Par exemple, si vous voulez envoyer les deux derniers commits du dépôt Git actuel à la revue, vous pouvez exécuter :

```
$ guix shell -D guix
[env]$ git send-email --to NUMÉRO_PROBLÈME@debbugs.gnu.org $(./etc/teams.scm cc-member
```

## 22.7 Suivi des bogues et des correctifs

Cette section décrit comme le projet Guix suit les rapports de bogues et les soumissions de correctifs.

### 22.7.1 L'outil de gestion des défauts

Les rapports de bogue et les correctifs soumis sont actuellement suivis par l'instance Debbugs <https://bugs.gnu.org>. Les rapports de bogue sont rempli pour le « paquet » `guix`

(dans le vocabulaire de Debbugs), en envoyant un courriel à `bug-guix@gnu.org`, tandis que les correctifs sont soumis pour le paquet `guix-patches` en envoyant un courriel à `guix-patches@gnu.org` (voir Section 22.6 [Envoyer des correctifs], page 683).

### 22.7.2 Interfaces utilisateurs à Debbugs

Une interface web (en fait *deux* interfaces web !) sont disponibles pour naviguer dans les tickets :

- <https://issues.guix.gnu.org> fournit une interface agréable<sup>2</sup> pour consulter les rapports de bogues et les correctifs, et pour participer aux discussions ;
- <https://bugs.gnu.org/guix> liste les rapports de bogues ;
- <https://bugs.gnu.org/guix-patches> liste les tickets soumis.

Pour consulter les discussions relatives au numéro d'édition *n*, rendez-vous sur '<https://issues.guix.gnu.org/n>' ou '<https://bugs.gnu.org/n>'.

Si vous utilisez Emacs, vous pourriez trouver plus confortable d'interagir avec les tickets en utilisant `debbugs.el`, que vous pouvez installer avec :

```
guix install emacs-debbugs
```

Par exemple, pour lister tous les tickets ouverts sur `guix-patches`, tapez :

```
C-u M-x debbugs-gnu RET RET guix-patches RET n y
```

Voir *Debbugs User Guide*, pour plus d'info sur cet outil pratique !

### 22.7.3 Étiquettes personnalisées de Debbugs

Debbugs fournit une fonctionnalité nommée *étiquettes personnalisées* (`usertags`) qui permettent à tous d'étiqueter un bogue avec une étiquette arbitraire. On peut rechercher les bogues par étiquette, ce qui les rend pratiques pour organiser les bogues<sup>3</sup>.

Par exemple, pour voir tous les rapports de bogues (ou les correctifs, dans le cas de `guix-patches` avec l'étiquette personnalisée `powerpc64le-linux` pour l'utilisateur `guix`, ouvrez cette URL dans un navigateur web : <https://debbugs.gnu.org/cgi-bin/pkgreport.cgi?tag=powerpc64le-linux;users=guix>.

Pour plus d'informations sur le fonctionnement des étiquettes personnalisées, référez-vous à la documentation de Debbugs ou à la documentation de n'importe quel outil que vous utilisez pour interagir avec Debbugs.

Dans Guix, nous expérimentons les étiquettes personnalisées pour suivre les problèmes spécifiques aux différentes architectures. Pour faciliter la collaboration, toutes les étiquettes personnalisées sont associées à un unique utilisateur `guix`. Les étiquettes suivantes existent pour cet utilisateur :

`powerpc64le-linux`

Le but de cet étiquettes est de faciliter la découverte des problèmes qui sont les plus importants pour le type de système `powerpc64le-linux`. Assignez cette étiquette aux bogues et aux correctifs qui affectent `powerpc64le-linux`

<sup>2</sup> L'interface web à <https://issues.guix.gnu.org> est alimentée par Mumi, un logiciel très cool écrit en Guile, votre aide est la bienvenue ! Voir <https://git.elephly.net/gitweb.cgi?p=software/mumi.git>.

<sup>3</sup> La liste des étiquettes personnalisées est publique, et n'importe qui peut modifier la liste des étiquettes de n'importe qui, donc pensez-y si vous utilisez cette fonctionnalité.

mais pas les autres types de systèmes. En plus, vous pouvez l'utiliser pour identifier les problèmes particulièrement importants pour le type de système `powerpc64le-linux`, même si le problème affecte d'autres types de systèmes aussi.

### reproductibilité

Pour les problèmes de reproductibilité. Par exemple, ce serait approprié d'assigner cette étiquette à un rapport de bogue pour un paquet qui ne se construit pas de manière reproductible.

Si vous avez les droits en commit et que vous souhaitez ajouter une étiquette personnalisée, utilisez-la simplement avec l'utilisateur `guix`. Si l'étiquette vous est utile, vous pourrez alors mettre à jour cette section du manuel pour que d'autres puissent savoir ce que votre étiquette signifie.

## 22.8 Accès en commit

Tout le monde peut contribuer à Guix même sans accès en commit (voir Section 22.6 [Envoyer des correctifs], page 683). Cependant, pour les contributeurs et contributrices fréquent·e·s, l'accès en écriture au dépôt peut être pratique. Le principe général est qu'un contributeur ou une contributrice devrait avoir accumulé cinquante commits revus pour demander les accès en commit et il ou elle doit avoir eu une activité soutenue dans le projet pendant au moins 6 mois. Cela permet de s'assurer qu'il y a eu suffisamment d'interactions avec le contributeur ou la contributrice, ce qui est nécessaire pour accompagner la personne et évaluer si elle est prête à recevoir les accès en commit. L'accès en commit n'est pas une médaille mais plutôt une responsabilité qu'un contributeur ou une contributrice accepte pour aider le projet.

Les sections suivantes expliquent comment recevoir des accès en commit, comment se préparer à pousser des commits et la politique et les attentes de la communauté pour les commits poussés en amont.

### 22.8.1 Candidater pour avoir accès en commit

Lorsque vous l'estimez nécessaire, pensez à candidater à l'accès en commit en suivant les étapes suivantes :

1. Trouvez trois personnes ayant les accès en commit et qui soutiendront votre candidature. Vous pouvez trouver la liste de ces personnes sur <https://savannah.gnu.org/project/memberlist.php?group=guix>. Chacune d'entre elles devra envoyer une déclaration à `guix-maintainers@gnu.org` (un alias privé pour le collectif des mainteneur·ses), signé de leur clef OpenPGP.

Les commiteurs devront avoir eu des interactions avec vous en tant que contributeur et pouvoir juger si vous êtes suffisamment familier avec les pratiques du projet. Ce n'est *pas* un jugement sur la valeur de votre travail, donc vous devriez plutôt interpréter un refus comme un « essayons un peu plus tard ».

2. Envoyez un message à `guix-maintainers@gnu.org` déclarant votre intention, listant les trois commiteu·ses qui supportent votre candidature, signez-le avec la clef OpenPGP que vous utiliserez pour signer vos commits et donnez son empreinte (voir plus bas). Voir <https://emailselfdefense.fsf.org/fr/>, pour une introduction à la cryptographie à clef publique avec GnuPG.

Configurer GnuPG de manière à ce qu'il n'utilise jamais l'algorithme de hachage SHA1 pour les signatures numériques, dont on sait qu'il est dangereux depuis 2019, par exemple en ajoutant la ligne suivante à `~/.gnupg/gpg.conf` (voir Section "GPG Esoteric Options" dans *The GNU Privacy Guard Manual*) :

```
digest-algo sha512
```

3. Les mainteneur·ses ont le dernier mot sur la décision de vous donner accès et suivent généralement l'avis de vos référents.
4. Si vous obtenez l'accès, envoyez un message à `guix-devel@gnu.org` pour le faire savoir, en signant de nouveau avec le clef OpenPGP que vous utiliserez pour signer les commits (faites-le avant votre premier commit). De cette manière, tout le monde peut s'en rendre compte et s'assurer que c'est bien vous qui contrôlez cette clef OpenPGP.

**Important:** Avant d'envoyer pour la première fois, les mainteneur·ses doivent :

1. ajoutez votre clé OpenPGP à la branche `keyring` ;
  2. ajoutez votre empreinte OpenPGP au fichier `.guix-authorizations` de la (des) branche(s) sur laquelle (lesquelles) vous vous engagez.
5. Assurez-vous de lire le reste de cette section et... profitez !

**Remarque:** Les mainteneur·ses se réjouissent de donner l'accès en commit aux personnes qui ont contribué depuis un certain temps et ont un historique — ne soyez pas timide et ne sous-estimez pas votre travail !

Cependant, remarquez que le projet travail sur un système de revu de correctifs et de fusion plus automatisé, qui, en conséquence, pourra nous faire réduire le nombre de personne ayant accès en commit au dépôt principal. Restez à l'écoute !

Tous les commits poussés sur le dépôt central sur Savannah doivent être signés avec une clef OpenPGP et la clef publique doit être chargée sur votre compte utilisateur dans Savannah et les serveurs de clef publique, comme `keys.opengpg.org`. Pour configurer Git pour signer automatiquement vos commits, lancez :

```
git config commit.gpgsign true
```

```
# Remplacez l'empreinte par celle de votre clé PGP publique
git config user.signingkey CABBA6EA1DC0FF33
```

Pour vérifier que les commits sont signés avec une clé correcte, utilisez :

```
make authenticate
```

Vous pouvez éviter de pousser accidentellement des commits non signés ou signés avec une mauvaise clé vers Savannah et utilisant le crochet Git de pré-envoi situé dans `etc/git/pre-push` :

```
cp etc/git/pre-push .git/hooks/pre-push
```

Il appelle aussi `make-check-channel-news` pour s'assurer que le fichier `news.scm` est correct.

## 22.8.2 Politique de commit

Si vous avez accès en commit, assurez-vous de respecter la politique ci-dessous (les discussions à propos de cette politique peuvent avoir lieu sur `guix-devel@gnu.org`).

Les correctifs non triviaux doivent toujours être postés sur `guix-patches@gnu.org` (les correctifs triviaux consistent en des correctifs de coquilles, etc). Cette liste de diffusion remplit la base de données de correctifs (voir Section 22.7 [Suivi des bogues et des correctifs], page 689).

Pour les correctifs qui ne font qu'ajouter un nouveau paquet, et un paquet simple, commiter directement est accepté si vous êtes en confiance (ce qui signifie que vous avez réussi à le construire dans un environnement chroot et avez fait un audit suffisant des copyrights et de la licence). Pareil pour les mises à jour de paquets, sauf pour les mises à jour qui occasionnent beaucoup de reconstruction (par exemple mettre à jour GnuTLS ou GLib). Nous avons une liste de diffusion pour les notifications de commit (`guix-commits@gnu.org`), pour que les gens puissent s'en rendre compte. Avant de pousser vos changements, assurez-vous d'avoir lancé `git pull --rebase`.

Lorsque vous poussez un commit pour le compte de quelqu'un d'autre, ajoutez une ligne `Signed-off-by` à la fin du message de commit — p. ex. avec `git am --signoff`. Cela améliore le suivi de qui fait quoi.

Quand vous ajoutez de nouvelles entrées au canal (voir Chapitre 7 [Canaux], page 75), assurez-vous qu'elles sont bien formées en exécutant la commande suivante juste avant d'envoyer :

```
make check-channel-news
```

Pour tout autre chose, envoyez un correctif à `guix-patches@gnu.org` et laissez un peu de temps pour la revue, sans rien commiter (voir Section 22.6 [Envoyer des correctifs], page 683). Si vous ne recevez pas de réponse dans les deux semaines et que vous avez confiance, vous pouvez commiter.

La dernière partie est sujète à ajustement, pour permettre à certains de commiter directement des changements non controversés sur des parties qui leurs sont familières.

## 22.8.3 Régler les problèmes

La revue par les pairs (voir Section 22.6 [Envoyer des correctifs], page 683) et les outils comme `guix lint` (voir Section 10.8 [Invoquer guix lint], page 213) et la suite de tests (voir Section 2.3 [Lancer la suite de tests], page 9) devraient trouver les problèmes avant qu'ils ne soient poussés. Pourtant, il arrive parfois que des commits qui « cassent » une fonctionnalité passent à travers les mailles du filet. Lorsque cela arrive, il y a deux priorités : minimiser l'impact et comprendre ce qui s'est passé, pour réduire les chances qu'un incident similaire se produise à nouveau. Les personnes impliquées ont la plus grande part de responsabilité dans ces deux tâches, mais comme pour tout le reste, c'est aussi un travail commun.

Certains problèmes peuvent directement affecter les utilisateurs et utilisatrices — par exemple si `guix pull` échoue ou qu'une fonctionnalité importante est cassée parce que des paquets importants sont cassés (à la construction ou à l'exécution), ou parce qu'ils introduisent des problèmes de sécurité connus.

Les personnes impliquées, auteurs, relecteurs et ayant poussé ces commits devraient avant tout réduire leur impact rapidement en poussant un nouveau commit qui le corrige



(si c'est possible) ou en inversant les commits pour laisser du temps avant de trouver un correctif satisfaisant, et en communiquant le problème aux autres développeurs.

Si ces personnes ne peuvent pas s'occuper du problème à temps, les autres commiteurs peuvent inverser les commits, en expliquant le problème dans le message de commit et sur la liste de diffusion, dans le but de donner du temps au commiteur, aux relecteurs et aux auteurs de proposer une solution.

Une fois que le problème a été réglé, il est de la responsabilité des personnes impliquées de s'assurer que la situation a été comprise. Si vous travaillez à comprendre ce qui est arrivé, efforcez-vous de récolter des informations plutôt que de désigner des coupables. Demandez aux gens impliqués de décrire ce qui est arrivé, ne leur demandez pas d'expliquer la situation – ce qui les mettrait implicitement dans une position de responsabilité et n'est d'aucune aide. La responsabilité vient une fois atteint un consensus sur le problème, qui permet d'apprendre de ce qui s'est passé et d'améliorer les processus afin qu'il soit moins à risque de se reproduire.

#### 22.8.4 Révocation des droits en commit

Pour réduire la probabilité de faire une erreur, celles et ceux qui peuvent commiter seront retiré·e·s du projet Guix sur Savannah et leur clé supprimée de `.guix-authorizations` après 12 mois d'inactivité ; ils et elles pourront retrouver leur accès en envoyant un courriel aux mainteneur·ses, sans avoir à passer par le processus de cooptation.

Les mainteneur·ses<sup>4</sup> peuvent aussi révoquer les droits en commit de quelqu'un, en dernier recours, si la coopération avec le reste de la communauté a créé trop de friction – même en restant dans le cadre du code de conduite (voir Chapitre 22 [Contribuer], page 668) du projet. Ils ne le feraient qu'après une discussion publique ou privée avec la personne concernée et un avertissement clair. Voici des exemples de comportements qui gênent la coopération et pourraient mener à une telle décision :

- enfreindre de manière répétée la politique de commit indiquée ci-dessus ;
- échouer de manière répétée à prendre en compte les critiques de la communauté ;
- briser la confiance suite à une série d'incidents graves.

Quand les mainteneurs recourent à une telle décision, ils en informent les développeurs sur `guix-devel@gnu.org` ; des questions peuvent être envoyées à `guix-maintainers@gnu.org`. Suivant la situation, les contributions de la personne peuvent rester bienvenues.

#### 22.8.5 Aider

Une dernière chose : le projet continue à avancer non seulement parce que les commiteurs poussent leurs propres changements, mais aussi parce qu'ils offrent de leur temps pour *revoir* et pousser les changements des autres personnes. En tant que commiteur, vous pouvez utiliser votre expertise et vos droits en commit pour aider d'autres contributeurs aussi !

---

<sup>4</sup> Voir <https://guix.gnu.org/fr/about> pour la liste à jour des mainteneur·ses. Vous pouvez leur envoyer un courriel en privé à l'adresse `guix-maintainers@gnu.org`.

## 22.9 Mettre à jour Guix

Il est quelquefois souhaitable de mettre à jour le paquet `guix` lui-même (le paquet défini dans `(gnu packages package-management)`), par exemple pour rendre de nouvelles caractéristiques disponibles à l'utilisation par le type de service `guix-service-type`. Afin de simplifier cette tâche, la commande suivante peut être utilisée :

```
make update-guix-package
```

Le make target `update-guix-package` utilisera le dernier *commit* connu correspondant à `HEAD` dans votre checkout Guix, calculera le hash des sources Guix correspondant à ce commit et mettra à jour la définition des paquets `commit`, `revision` et le hash du paquet `guix`.

Pour avoir la certitude que les hash du paquet `guix` mis à jour sont corrects et qu'il peut être construit avec succès, la commande suivante peut être lancée depuis le répertoire de votre checkout Guix :

```
./pre-inst-env guix build guix
```

Pour éviter de mettre à jour accidentellement le paquet `guix` en un commit auquel les autres ne peuvent pas se référer, on vérifie que le commit utilisé a déjà été placé dans le dépôt git de Guix hébergé par Savannah.

Cette vérification peut être désactivée, *à vos risques et périls*, en passant la variable d'environnement `GUIX_ALLOW_ME_TO_USE_PRIVATE_COMMIT`. Lorsque cette variable est initialisée, les source du paquet mis à jour est aussi ajouté au dépôt. Cela est utilisé dans le processus de publication de Guix.

## 22.10 Écrire de la documentation

Guix est documenté avec le système Texinfo. Si vous n'êtes pas à l'aise avec, nous acceptons les contributions pour la documentation dans la plupart des formats. Cela comprends le texte brut, Markdown, Org, etc.

Vous pouvez envoyer des contributions pour la documentation à `guix-patches@gnu.org`. Ajoutez `'[DOCUMENTATION]'` au début de l'objet.

Lorsque vous avez besoin de faire plus qu'un simple ajout à la documentation, nous préférons que vous envoyiez un correctif plutôt que d'envoyer un courriel comme précisé plus haut. Voir Section 22.6 [Envoyer des correctifs], page 683, pour en apprendre plus sur l'envoi de vos correctifs.

Pour modifier la documentation, vous devrez modifier `doc/guix.texi` et `doc/contributing.texi` (qui contient cette section de la documentation) ou `doc/guix-cookbook.texi` pour le livre de recettes. Si vous avez déjà compilé le dépôt Guix, vous aurez bien plus de fichiers `.texi` qui sont les traductions de ces documents. Ne les modifiez pas, les traductions sont gérées par Weblate (<https://translate.fedoraproject.org/projects/guix>). Voir Section 22.11 [Traduire Guix], page 696, pour plus de détails.

Pour générer la documentation, vous devez d'abord vous assurer d'avoir lancé `./configure` dans votre arborescence des sources (voir Section 22.2 [Lancer Guix avant qu'il ne soit installé], page 670). Ensuite, vous pourrez lancer l'une des commandes suivantes :

- `'make doc/guix.info'` pour compiler le manuel Info. Vous pouvez le consulter avec `info doc/guix.info`.

- ‘`make doc/guix.html`’ pour compiler la version HTML. Vous pouvez pointer votre navigateur vers le fichier qui vous intéresse dans le répertoire `doc/guix.html`.
- ‘`make doc/guix-cookbook.info`’ pour le manuel Info du livre de recettes.
- ‘`make doc/guix-cookbook.html`’ pour la version HTML du livre de recettes.

## 22.11 Traduire Guix

Écrire du code et des paquets n’est pas la seule manière possible de proposer une contribution intéressante à Guix. Traduire vers une langue que vous parlez est un autre exemple de contribution importante que vous pouvez faire. Cette section décrit le processus de traduction. Elle vous donner des conseils sur comment vous impliquer, ce qui peut être traduit, les erreurs à éviter et ce que vous pouvez faire pour nous aider !

Guix est un gros projet qui a plusieurs composants traduisibles. Nous coordonnons l’effort de traduction sur une instance Weblate (<https://translate.fedoraproject.org/projects/guix/>) hébergée par nos amis de Fedora. Vous aurez besoin d’un compte pour soumettre vos traductions.

Certains logiciels empaquetés par Guix contiennent aussi des traductions. Nous n’hébergeons pas de plateforme de traduction pour eux. Si vous voulez traduire un paquet fournit par Guix, vous devrez contacter ses développeurs ou trouver plus d’information sur son site web. Par exemple, vous pouvez trouver la page d’accueil du paquet `hello` en lançant `guix show hello`. Sur la ligne indiquant « homepage », vous verrez que la page d’accueil est <https://www.gnu.org/software/hello/>.

De nombreux paquets GNU et non-GNU peuvent être traduits sur le Projet de traduction (<https://translationproject.org>). Certains projets avec de multiples composants ont leur propre plateforme. Par exemple, GNOME a sa propre plateforme, Damned Lies (<https://110n.gnome.org/>).

Guix a cinq composants hébergés sur Weblate.

- `guix` contient toutes les chaînes du logiciel Guix (le guide d’installation du système, le gestionnaire de paquets, etc), en dehors des paquets.
- `packages` contient le synopsis (une description courte en une ligne) et la description (longue) des paquets de Guix.
- `website` contient le site web officiel de Guix, en dehors des billets de blog et du contenu multimédia.
- `documentation-manual` correspond à ce manuel.
- `documentation-cookbook` est le composant du livre de recettes.

### Instructions générales

Une fois que vous aurez un compte, vous pourrez choisir un composant dans projet guix (<https://translate.fedoraproject.org/projects/guix/>), et choisir une langue. Si votre langue n’apparaît pas dans la liste, allez en bas et cliquez sur le bouton « commencer une nouvelle traduction ». Choisissez la langue en laquelle vous voulez traduire dans la liste pour commencer votre nouvelle traduction.

Comme de nombreux autres logiciels libres, Guix utilise GNU Gettext (<https://www.gnu.org/software/gettext>) pour ses traductions. Grâce à lui, les chaînes traduisibles sont extraites du code source en des fichier dits PO.

Même si les fichiers PO sont des fichiers textes, vous devriez les modifier non pas avec un éditeur de texte mais avec un logiciel d'édition de PO. Weblate intègre une fonctionnalité d'édition de PO. Autrement, vous pouvez utiliser n'importe lequel des divers outils libres pour remplir les traductions, dont Poedit (<https://poedit.net/>) est un exemple, et (après vous être connecté) téléverser (<https://docs.weblate.org/en/latest/user/files.html>) le fichier modifié. Il y a aussi un mode d'édition de PO (<https://www.emacswiki.org/emacs/PoMode>) spécial pour GNU Emacs. Avec le temps vous trouverez les logiciels qui vous satisfont et les fonctionnalités dont vous avez besoin.

Sur Weblate, vous trouverez plusieurs liens vers l'éditeur, qui vous montreront divers sous-ensemble (ou la totalité) des chaînes. Explorez Weblate et consultez la documentation (<https://docs.weblate.org/fr/latest/>) pour vous familiariser avec la plateforme.

## Composants des traductions

Dans cette section, nous proposons un guide plus détaillé du processus de traduction, ainsi que des détails sur ce que vous devriez ou ne devriez pas faire. Si vous avez un doute, contactez-nous, nous serons ravis de vous aider !

**guix** Guix est écrit dans le langage de programmation Guile, et certaines chaînes contiennent du formatage spécial interprété par Guile. Ce formatage spécial devrait être mis en évidence par Weblate. Il commence par `~` suivi d'un ou plusieurs caractères.

Lors de l'affichage des chaînes, Guile remplace les symboles de formatage spéciaux avec les vrais valeurs. Par exemple, la chaîne `'ambiguous package specification ~a'` serait modifiée pour contenir la spécification du paquet en question au lieu de `~a`. Pour traduire cette chaîne correctement, vous devez garder le formatage dans vos traduction, même si vous pouvez le placer là où cela a le plus de sens dans votre langue. Par exemple, la traduction française dit `'spécification du paquet « ~a » ambiguë'` parce que l'adjectif doit être placé à la fin de la phrase.

S'il y a plusieurs symboles de formatage, assurez-vous de respecter l'ordre. Guile ne sait pas dans quel ordre vous voulez qu'il lise la chaîne, donc il remplacera les symboles dans le même ordre que la phrase anglaise.

Par exemple, vous ne pouvez pas traduire `'package '~a' has been superseded by '~a'` par `'« ~a » remplace le paquet « ~a »'`, parce que le sens serait renversé. Si *toto* est remplacé par *titi*, la traduction dirait `'« toto » remplace le paquet « titi »'`. Pour éviter ce problème, il est possible d'utiliser un formatage plus avancé pour désigner une donnée particulière, au lieu de suivre l'ordre anglais par défaut. Voir Section "Formatted Output" dans *GNU Guile Reference Manual*, pour plus d'information sur le formatage en Guile.

**paquets**

Les descriptions de paquets contiennent parfois du balisage Texinfo (voir Section 22.4.4 [Synopsis et descriptions], page 676). Le balisage Texinfo ressemble à `'@code{rm -rf}'`, `'@emph{important}'`, etc. Lors de la traduction, laissez le balisage tel quel.

Les caractères après `« @ »` forment le nom de la balise, et le texte entre `« { »` et `« } »` est son contenu. En général, vous ne devriez pas traduire le contenu des

balises comme `@code`, car elles contiennent littéralement du code qui ne change pas avec la langue. Vous pouvez traduire le contenu des balises de formatage comme `@emph`, `@i`, `@itemize`, `@item`. Cependant, ne traduisez pas le nom de la balise, ou elle ne sera plus reconnue. Ne traduisez pas le mot après `@end`, c'est le nom de la balise fermée à cet endroit (p. ex. `@itemize ... @end itemize`).

documentation-manual et documentation-cookbook

La première étape pour assurer la traduction du manuel est de trouver et traduire les chaînes suivantes *en premier* :

- `version.texi` : Traduisez cette chaîne en `version-xx.texi`, où `xx` est le code de votre langue (celui affiché dans l'URL sur `weblate`).
- `contributing.texi` : Traduisez cette chaîne en `contributing.xx.texi`, où `xx` est le même code de langue.
- `Top` : Ne traduisez pas cette chaîne, c'est important pour Texinfo. Si vous la traduisez, le document sera vide (il manquera le nœud `Top`). Cherchez-là et utilisez `Top` comme traduction.

Traduire ces chaînes en premier s'assure que nous pouvez inclure vos traduction dans le dépôt `guix` sans casser le processus `make` ni la machinerie derrière `guix pull`.

Le manuel et le livre de recettes utilisent tous les deux Texinfo. Comme pour `packages`, gardez le balisage Texinfo intact. Il y a plus de types de balises possibles dans le manuel que dans les descriptions de paquets. En général, ne traduisez pas le contenu de `@code`, `@file`, `@var`, `@value`, etc. Vous devriez traduire le contenu des balises de formatage comme `@emph`, `@i`, etc.

Le manuel contient des sections auxquelles on peut se référer par leur nom avec `@ref`, `@xref` et `@pxref`. Nous avons mis en place un mécanisme pour que vous n'ayez pas besoin de traduire leur contenu. Si vous gardez le titre anglais, nous le remplacerons automatiquement par votre traduction de ce titre. Cela s'assure que Texinfo sera toujours capable de trouver le nœud. Si vous décidez de changer la traduction du titre, les références seront automatiquement mises à jour et vous n'aurez pas à les mettre à jour vous-même.

Lorsque vous traduisez des références de livre de recettes vers le manuel, vous devrez remplacer le nom du manuel et le nom de la section. Par exemple, pour traduire `@pxref{Defining Packages,,, guix, GNU Guix Reference Manual}`, vous devrez remplacer `Defining Packages` par le titre de cette section dans le manuel traduit *seulement* si ce titre est traduit. Si le titre n'est pas encore traduit dans votre langue, ne le traduisez pas ici, ou le lien sera cassé. Remplacez `guix` par `guix.xx` où `xx` est le code de votre langue. `GNU Guix Reference Manual` est le texte du lien. Vous pouvez le traduire comme bon vous semble.

website

Les pages du site web sont écrites en SXML, une manière d'écrire du HTML, le langage de base du web, avec des `s`-expressions. Nous avons un processus pour extraire les chaînes traduisibles dans les sources, et pour remplacer les `s`-expressions complexes avec du balisage XML plus familier, où chaque balise

est numérotée. Vous pouvez changer l'ordre comme bon vous semble, comme dans l'exemple suivant.

```
#. 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>
```

Remarquez que vous devez inclure les même balises. Vous ne devez pas en oublier.

Si vous faites une erreur, le composant peut échouer à construire correctement dans votre langue, voir casser guix pull. Pour éviter cela, nous avons mis en place un processus pour vérifier le contenu des fichiers avant de les pousser dans notre dépôt. Nous ne pourrions alors pas mettre à jour la traduction en votre langue dans Guix, donc nous vous notifierons (à travers weblate ou par courriel) pour que vous puissiez corriger le problème.

## En dehors de Weblate

Actuellement, certaines parties de Guix ne peuvent pas être traduites sur Weblate, nous avons besoin d'aide !

- Les nouvelles de `guix pull` peuvent être traduites dans `news.scm`, mais ne sont pas disponibles sur Weblate. Si vous voulez fournir une traduction, vous pouvez préparer un correctif comme décrit plus haut, ou simplement nous envoyer votre traduction avec le nom de l'entrée que vous avez traduite et votre langue. Voir Section 7.11 [Écrire des nouveautés de canaux], page 81, pour plus d'information sur les nouvelles de canaux.
- Les billets de blog de Guix ne peuvent pas encore être traduits.
- Le script d'installation (pour les distributions externes) est entièrement en anglais.
- Certaines bibliothèques que Guix utilise ne peuvent pas être traduites ou leurs traductions sont en dehors du projet Guix. Guile lui-même n'est pas internationalisé.
- Les liens vers les autres manuels ici ou dans le livre de recettes ne sont pas forcément traduits.

## Conditions d'inclusion

Il n'y a pas de condition particulière pour ajouter de nouvelles traductions aux composants `guix` et `guix-packages`, en dehors du fait qu'elles doivent avoir au moins une chaîne traduite. Les nouvelles langues seront ajoutées à Guix le plus tôt possible. Les fichiers peuvent être supprimés s'ils ne sont plus synchronisés et qu'ils n'ont plus aucune chaîne traduite.

Étant donné que le site web est prévu pour les nouveaux utilisateurs, nous souhaitons que ses traductions soient les plus complètes possible avant de les inclure dans le menu des langues. Pour qu'une nouvelle langue soit ajoutée, elle doit atteindre au moins les 80% de complétion. Lorsqu'une langue est incluse, elle peut être supprimée dans le futur si elle reste désynchronisée et que sa complétion descend en dessous des 60%.

Le manuel et le livre de recettes sont automatiquement ajoutés à la cible de compilation par défaut. Chaque fois que nous synchronisons les traductions, les développeurs et développeuses doivent recompiler tous les manuels et les livres de recettes traduits. C'est inutile s'il s'agit en grande partie du manuel ou du livre de recettes en anglais. De ce fait,

nous n'incluons les nouvelles langues que si elles atteignent 10% de complétion dans le composant en question. Lorsqu'une langue est ajoutée, elle pourra être supprimée dans le futur si elle reste désynchronisée et que son taux de complétion descend en dessous des 5%.

## Infrastructure de traduction

Weblate s'appuie sur un dépôt git à partir duquel il découvre les nouvelles chaînes à traduire et vers lequel il pousse les nouvelles traductions. Normalement, il suffirait de lui donner accès en commit à nos dépôts. Cependant, nous avons préféré utiliser un dépôt séparé pour deux raisons. Tout d'abord, nous aurions dû donner à Weblate l'accès en commit et autoriser sa clé de signature, mais nous ne lui faisons pas autant confiance qu'aux développeurs de guix, d'autant plus que nous ne gérons pas l'instance nous-même. Ensuite, si les traducteurs et les traductrices font une erreur, cela peut casser la génération du site web ou guix pull pour tout le monde, quelle que soit la langue.

Pour ces raisons, nous utilisons un dépôt dédié pour héberger les traductions, et nous le synchronisons avec nos dépôts guix et artwork après avoir vérifié qu'aucun problème n'est introduit par les traductions.

vous pouvez télécharger les derniers fichiers PO de weblate dans le dépôt Guix en lançant la commande `make download-po`. Cela télécharge automatiquement les derniers fichiers de weblate, les reformate vers une forme canonique, et vérifie qu'ils ne contiennent pas de problème. Le manuel doit être reconstruit pour vérifier qu'il n'y a pas de problème supplémentaire qui ferait planter Texinfo.

Avant de pousser de nouveaux fichiers de traduction, qui n'existaient pas avant, vous devez les ajouter à la machinerie make pour que les traductions soit effectivement disponibles. Le processus diffère en fonction des composants.

- Les nouveaux fichiers po pour les composants `guix` et `packages` doivent être enregistrés en ajoutant la nouvelle langue dans `po/guix/LINGUAS` ou `po/packages/LINGUAS`.
- Les nouveaux fichiers pour le composant `documentation-manuel` doivent être enregistrés en ajoutant le nom de fichier à `DOC_PO_FILES` dans `po/doc/local.mk`, le manuel généré `%D%/guix.xx.texi` à `info_TEXINFOS` dans `doc/local.mk` et les fichiers `%D%/guix.xx.texi` et `%D%/contributing.xx.texi` à `TRANSLATED_INFO` aussi dans `doc/local.mk`.
- Les nouveaux fichiers po pour le composant `documentation-cookbook` doivent être enregistrés en ajoutant le nom de fichier à `DOC_COOKBOOK_PO_FILES` dans `po/doc/local.mk`, le manuel généré `%D%/guix-cookbook.xx.texi` à `info_TEXINFOS` dans `doc/local.mk` et le fichier généré `%D%/guix-cookbook.xx.texi` à `TRANSLATED_INFO` aussi dans `doc/local.mk`.
- Les nouveaux fichiers po pour le composant `website` doivent être ajoutés dans le dépôt `guix-artwork`, dans `website/po/`. Vous devez mettre à jour `website/po/LINGUAS` et `website/po/ietf-tags.scm` en fonction (voir `website/i18n-howto.txt` pour plus d'information sur le processus).

## 23 Remerciements

Guix se base sur le <https://nixos.org/nix/> gestionnaire de paquets Nix conçu et implémenté par Eelco Dolstra, avec des contributions d'autres personnes (voir le fichier `nix/AUTHORS` dans Guix). Nix a inventé la gestion de paquet fonctionnelle et promu des fonctionnalités sans précédents comme les mises à jour de paquets transactionnelles et les retours en arrière, les profils par utilisateurs et les processus de constructions transparents pour les références. Sans ce travail, Guix n'existerait pas.

Les distributions logicielles basées sur Nix, Nixpkgs et NixOS, ont aussi été une inspiration pour Guix.

GNU Guix lui-même est un travail collectif avec des contributions d'un grand nombre de personnes. Voyez le fichier `AUTHORS` dans Guix pour plus d'information sur ces personnes de qualité. Le fichier `THANKS` liste les personnes qui ont aidé en rapportant des bogues, en prenant soin de l'infrastructure, en fournissant des images et des thèmes, en faisant des suggestions et bien plus. Merci !



# Annexe A La licence GNU Free Documentation

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## 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. Secondly, 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.



## Index des concepts

- 
- .local, résolution de nom d'hôte ..... 573
- /
- /bin/sh ..... 268
- /usr/bin/env ..... 268

## É

- Étiquettes personnalisées pour Debbugs ..... 690
- échecs de construction, débogage ..... 193
- échelonnage, du code ..... 146, 166
- écran tactile, pour Xorg ..... 561
- éléments du dépôt ..... 157
- éléments du dépôt invalides ..... 158
- émulation ..... 513, 514
- épinglage, canaux ..... 66, 76
- épinglage, des révisions des canaux d'un profil .. 54
- équipes ..... 688, 689
- étendre la collection de paquets (canaux) ..... 75
- étiquettes personnalisées, pour debbugs ..... 690

## A

- AArch64, chargeurs d'amorçage ..... 578
- abaissement, des objets
  - haut-niveau dans les gexps ..... 167, 175
- accès au scanner ..... 354
- accès distant au démon ..... 21, 157
- accès en commit, pour les développeurs ..... 691
- Accès SSH au démon de construction ..... 158
- accent grave (quasiquote) ..... 106
- ACL (liste de contrôle d'accès),
  - pour les substituts ..... 55
- action, des services Shepherd ..... 609
- activation par socket, pour **guix publish** ..... 223
- activation par socket, pour **guix-daemon** ..... 17
- agate ..... 460
- agent de transfert de courriel (MTA) ..... 394
- ajout de racines au ramasse-miettes ..... 192
- alias de courriel ..... 399
- alias, de **guix package** ..... 45
- alias, pour les adresses de courriel ..... 399
- ALSA ..... 364
- amont, dernière version ..... 187
- analyse de style, style du code ..... 210
- ancien système de démarrage, sur
  - les machines Intel ..... 239
- architectures externes ..... 235
- archivage de code source, Software Heritage ... 213
- archive ..... 71
- archive normalisée (nar) ..... 72

- ARM, chargeurs d'amorçage ..... 578
- Audit ..... 564
- authenticité de code obtenue avec **guix pull** ... 63
- authentification d'un extrait de Guix ..... 66, 77
- authentification, d'un extrait de Guix ..... 668
- autorisation, archives ..... 73
- AutoSSH ..... 315

## B

- Balilage texinfo, dans les
  - descriptions de paquets ..... 677
- bash ..... 625, 636
- binaires de bootstrap ..... 659, 665
- binaires pré-construits ..... 54
- binaires repositionnables ..... 98
- binaires repositionnables, avec **guix pack** ..... 96
- binfmt\_misc** ..... 513
- Bioconductor ..... 199
- BIOS, chargeur d'amorçage ..... 578
- black-list, des modules du noyau ..... 577
- bootloader ..... 578
- bootstrap ..... 659
- but ..... 1

## C

- cache de polices ..... 24
- cache, dans **guix shell** ..... 89
- cache, des profils ..... 89
- CalDAV ..... 400
- canaux ..... 75
- canaux, pour des paquets personnels ..... 78
- CardDAV ..... 400
- Cargo (système de construction Rust) ..... 128
- cat-avatar-generator ..... 458
- certificats ..... 91
- Certificats TLS ..... 461
- Certificats X.509 ..... 572
- chaîne d'outils, changer la chaîne
  - d'outils d'un paquet ..... 186
- chaîne d'outils, choisir la chaîne
  - d'outils d'un paquet ..... 112
- chaîne d'outils, pour le développement en C ... 101
- chaîne d'outils, pour le
  - développement en Fortran ..... 101
- channels.scm**, fichier de configuration ..... 64, 75
- chargeur d'amorçage ..... 578
- chargeur de module du noyau ..... 556
- chemin de dérivation ..... 159
- chemin de recherche ..... 154
- chemin de recherche des modules de paquets .. 104
- chemin le plus court entre paquets ..... 221
- chemins dans le dépôt ..... 157

- chemins de recherche ..... 45, 49
  - chercher de la documentation ..... 641
  - chercher des paquets ..... 51
  - chiffrement de l'espace d'échange ..... 258
  - chiffrement du disque ..... 33, 241, 257
  - childhurd ..... 515
  - childhurd, déchargement ..... 517
  - chroot ..... 12, 17
  - chroot, système guix ..... 38
  - chrooter, système guix ..... 38
  - clefs autorisées, SSH ..... 313
  - client de shell sécurisé, configuration ..... 631
  - client SSH, configuration ..... 631
  - Clojure (langage de programmation) ..... 130
  - closure ..... 62, 215, 216
  - closure de module ..... 168
  - code échelonné ..... 146, 166
  - code de conduite, des contributeur-rices ..... 668
  - collisions, dans un profil ..... 50
  - compilation croisée ..... 100, 108, 168, 192
  - compilation croisée,
    - dépendances des paquets ..... 109
  - composition de révisions de Guix ..... 68
  - comptes ..... 260
  - comptes de construction ..... 11
  - comptes utilisateurs ..... 260
  - confiance, en des binaires pré-construits ..... 58
  - configuration de **guix pull** ..... 75
  - configuration du dossier personnel ..... 619
  - configuration du système ..... 238
  - configuration git ..... 687
  - configuration, action des services Shepherd ..... 268
  - configurations complexes ..... 612
  - Connexion X11 ..... 328
  - Connman ..... 297
  - construction de paquets ..... 180
  - construction groupée ..... 684
  - construction reproductibles, vérification ..... 685
  - construction, dépendances ..... 159
  - constructions distantes ..... 492
  - constructions reproductibles ..... 17, 44, 57, 226
  - constructions vérifiables ..... 226
  - conteneur ..... 87, 91, 94, 230
  - conteneur, environnement de construction ..... 17
  - conteneur, pour **guix home** ..... 621, 636
  - ContentDB ..... 198
  - contrôle d'accès obligatoire, SELinux ..... 16
  - contrôleur d'interface réseau (NIC) ..... 293
  - convention de contribution ..... 668
  - Conventions de formatage ..... 210
  - conversion de paquets ..... 197
  - copier des éléments du dépôt par SSH ..... 229
  - correctifs ..... 106
  - correction des problèmes du système guix ..... 37
  - correction des problèmes, guix system ..... 37
  - correspondance de nom ..... 487
  - corruption, récupérer de ..... 62, 192
  - courriel ..... 372
  - CPAN ..... 199
  - créer des images de machines virtuelles ..... 589
  - Créer des images systèmes sous
    - différents formats ..... 588
  - CRAN ..... 199
  - crate ..... 202
  - crochet de construction ..... 12
  - cron ..... 287, 628
  - CTAN ..... 199
  - CVE, Common Vulnerabilities
    - and Exposures ..... 214
- ## D
- déchargement ..... 12, 18
  - déclaration de profil ..... 48
  - déduplication ..... 20, 62
  - défaire des transactions ..... 49, 65
  - défi ..... 226
  - définition de paquets, modification ..... 194
  - définition des paramètres linguistiques ..... 265
  - définition des types de service,
    - modification ..... 584, 637
  - démon ..... 11
  - démon de construction ..... 1
  - démon de gestion du remplissage de la
    - mémoire précoce ..... 555
  - démon, accès distant ..... 21, 157
  - démon, paramètres de grappes ..... 21, 157
  - démons ..... 599
  - dépôt ..... 2, 157
  - dépendances à l'exécution ..... 159
  - dépendances à la construction ..... 159
  - dépendances des paquets ..... 61, 218
  - dépendances, canaux ..... 79
  - dérivation ..... 62, 104
  - dérivations ..... 159
  - dérivations à sortie fixe ..... 159
  - dérivations à sortie fixe, pour télécharger ..... 113
  - déterminisme, du processus de construction ... 685
  - déterminisme, vérification ..... 192
  - développement logiciel ..... 83
  - Démarrage BIOS, sur les machines Intel ..... 239
  - Démarrage EFI ..... 239
  - Démarrage UEFI ..... 239
  - Démon de fiabilité de la plateforme, de
    - disponibilité et de serviabilité ..... 557
  - Démon IMAP4 GNU Mailutils ..... 399
  - DAG ..... 218
  - darkstat ..... 428
  - database ..... 366
  - ddclient ..... 476
  - Debbugs, interface web Mumi ..... 453
  - Debbugs, système de suivi de tickets ..... 689
  - dernier commit, construction ..... 186
  - des variantes de paquets (canaux) ..... 75
  - descripteurs de fichiers ouverts ..... 282
  - description du paquet ..... 676

DHCP ..... 31  
 DHCP, service réseau ..... 295  
 dhtproxy, à utiliser avec jami ..... 306  
 dictionnaire ..... 562  
 disponibilité des substituts ..... 231  
 disposition clavier ..... 263  
 disposition des touches, Xorg ..... 333  
 disposition du clavier ..... 31, 263  
 disposition du clavier, configuration ..... 264  
 disposition du clavier, définition ..... 263  
 disposition du clavier, pour le  
   chargeur d'amorçage ..... 580  
 disposition du clavier, pour Xorg ..... 333  
 disque de RAM initial ..... 247, 575, 577  
 Distribution android ..... 127  
 Distribution Système Guix,  
   maintenant le système Guix ..... 1  
 distro externe ..... 5, 22  
 DNS (domain name system) ..... 464  
 Docker ..... 563  
 Docker, construire une image avec guix pack ... 96  
 docker-image, créer des images docker ..... 589  
 documentation ..... 59, 695  
 documentation, recherche ..... 641  
 domain name system (DNS) ..... 464  
 domaine, kerberos ..... 434  
 dual boot ..... 581

## E

earlyoom ..... 555  
 EFI, chargeur d'amorçage ..... 578  
 EFI, installation ..... 32  
 egg ..... 204  
 elips, création de paquets ..... 677  
 Elm ..... 680  
 elm ..... 203  
 elpa ..... 202  
 emacs ..... 24  
 emacs, création de paquets ..... 677  
 email ..... 372  
 empreinte digitale ..... 559  
 entrée tablette, pour Xorg ..... 561  
 entrées de développement, d'un paquet ..... 112  
 entrées implicites, d'un paquet ..... 112  
 entrées propagées ..... 46  
 entrées, des paquets ..... 109  
 entrées, pour les paquets Python ..... 679  
 env, dans `/usr/bin` ..... 268  
 envelopper des programmes ..... 152  
 enveloppes de programmes ..... 152  
 environnement de construction ..... 11, 17  
 environnement de construction de paquets ..... 83  
 environnement de développement ..... 83  
 environnement persistant ..... 89, 92  
 environnements de construction  
   reproductibles ..... 83  
 ESP, partition système EFI ..... 32

espace d'échange ..... 258  
 espace d'échange compressé ..... 558  
 espace disque ..... 60  
 exécution, dépendances ..... 159  
 exporter des éléments du dépôt ..... 71  
 exporter des fichiers du dépôt ..... 71  
 extensibilité de la distribution ..... 1  
 extensions de service ..... 599  
 extensions, des gexps ..... 169  
 extraits de code ..... 672

## F

Fail2Ban ..... 567  
 faire des collisions de paquets dans des profils .. 50  
 fastcgi ..... 453  
 fc-cache ..... 24  
 fcgiwrap ..... 453  
 ferme de construction ..... 54  
 FHS (standard de la hiérarchie des  
   systèmes de fichiers) ..... 88  
 fichier de configuration pour les canaux ..... 64, 75  
 fichier de configuration, d'un  
   service Shepherd ..... 268  
 fichier de configuration, des  
   services Shepherd ..... 611  
 fichier hosts ..... 248  
 fichier sudoers ..... 250  
 fichier, chercher ..... 149  
 fichiers de débogage ..... 652  
 fichiers de verrouillage ..... 76  
 firmware ..... 248  
 fonctions monadiques ..... 162  
 Format d'image de CD ..... 590  
 Format d'image de DVD ..... 590  
 Format ISO-9660 ..... 590  
 formatage, du code ..... 683  
 formatage, du style du code ..... 210  
 formater le code ..... 683

## G

générations ..... 49, 52, 65, 586  
 générations du dossier personnel ..... 638  
 G-expression ..... 166  
 gagner de la place ..... 586, 638  
 ganeti ..... 518  
 GCC ..... 101  
 GDM ..... 325  
 gem ..... 198  
 gestion de l'énergie ..... 629  
 gestion de l'énergie avec TLP ..... 495  
 Gestion de la fréquence du CPU  
   avec thermald ..... 502  
 gestion de paquet fonctionnelle ..... 1  
 gestionnaire d'affichage, lightdm ..... 330  
 gestionnaire de connexion ..... 325, 328  
 gestionnaire de fenêtre ..... 325

gestionnaire du remplissage de la mémoire . . . . . 555  
**git format-patch** . . . . . 687  
**git send-email** . . . . . 687  
 Git, forge . . . . . 545  
 Git, hébergement . . . . . 543  
 Git, interface web . . . . . 531  
 Git, utiliser le dernier commit . . . . . 186  
 gmnisrv . . . . . 459  
 GNOME, gestionnaire de connexion . . . . . 325  
 go . . . . . 203  
 gpm . . . . . 280  
 graphe d'extension des services, pour  
   l'environnement personnel . . . . . 639  
 Graphe de dépendance du Shepherd, pour  
   l'environnement personnel . . . . . 640  
 grappes, paramètres du démon . . . . . 21, 157  
 greffes . . . . . 657  
 groupes . . . . . 261, 262  
 GSS . . . . . 487  
 GSSD . . . . . 487  
**guix archive** . . . . . 71  
**guix build** . . . . . 180  
**guix challenge** . . . . . 226  
**guix container** . . . . . 230  
**guix copy** . . . . . 229  
**guix deploy** . . . . . 593  
**guix describe** . . . . . 70  
**guix download** . . . . . 195  
**guix edit** . . . . . 194  
**guix environment** . . . . . 83, 90  
**guix gc** . . . . . 60  
**guix git authenticate** . . . . . 102  
**guix graph** . . . . . 218  
**guix hash** . . . . . 196  
**guix home** . . . . . 636  
**guix lint** . . . . . 213  
**guix pack** . . . . . 96  
**guix package** . . . . . 44  
**guix processes** . . . . . 233  
**guix publish** . . . . . 222  
**guix pull** . . . . . 63  
**guix pull** pour l'utilisateur-riche root, sur une  
   distribution externe . . . . . 25  
**guix pull**, fichier de configuration . . . . . 75  
**guix refresh** . . . . . 205  
**guix repl** . . . . . 176  
**guix shell** . . . . . 83  
**guix size** . . . . . 215  
**guix style** . . . . . 210  
**guix system** . . . . . 584  
 Guix System, installation . . . . . 26  
**guix time-machine** . . . . . 66  
**guix weather** . . . . . 231  
**guix-daemon** . . . . . 17  
 GuixSD, maintenant le système Guix . . . . . 1

## H

héritage, pour les définitions des paquets . . . . . 116  
 hackage . . . . . 200  
 HDPI . . . . . 583, 584  
 hexpm . . . . . 204  
 HiDPI . . . . . 583, 584  
 horloge . . . . . 302  
 hpcguix-web . . . . . 458  
 HTTP . . . . . 441  
 HTTP, HTTPS . . . . . 461  
 HTTPS, certificats . . . . . 572  
 hurd . . . . . 246, 515  
 Hurd, déchargement . . . . . 517

## I

i18n . . . . . 696  
 idmapd . . . . . 487  
 image d'installation . . . . . 36  
 image, créer des images disques . . . . . 588  
 Images système, création en divers formats . . . . . 588  
 images systèmes . . . . . 644  
 IMAP . . . . . 395  
 importer des fichiers dans le dépôt . . . . . 71  
 importer des paquets . . . . . 197  
 incompatibilité, des données linguistiques . . . . . 266  
 indentation, du code . . . . . 683  
 inetd . . . . . 304  
 inférieurs . . . . . 68, 176  
 Info, format de documentation . . . . . 641  
 informations de débogage, reconstruire . . . . . 185, 653  
 initrd . . . . . 247, 575, 577  
 inputattach . . . . . 561  
 insérer ou mettre à jour la ligne de copyright . . . . . 672  
 installation de paquets . . . . . 44  
 installer des paquets . . . . . 44  
 installer Guix . . . . . 5  
 installer Guix depuis les binaires . . . . . 5  
 installer Guix System . . . . . 26  
 installer par SSH . . . . . 32  
 intégration continue . . . . . 186, 490  
 intégration continue, statistiques . . . . . 231  
 intégrité, du dépôt . . . . . 62  
 interfaces utilisateur-riche . . . . . 1  
 introduction des canaux . . . . . 80  
 invalidation du cache, nscd . . . . . 274  
 inverser des comits . . . . . 693  
 invocation de programme, depuis Scheme . . . . . 150  
 invocation de programmes, depuis Scheme . . . . . 150  
 Invoquer **guix import** . . . . . 197  
 IPFS . . . . . 321  
 iptables . . . . . 301  
 IRC (Internet Relay Chat) . . . . . 407  
 isolation . . . . . 1

**J**

|                                                                               |          |
|-------------------------------------------------------------------------------|----------|
| jabber .....                                                                  | 400      |
| jackd .....                                                                   | 282      |
| jami, service .....                                                           | 408      |
| java .....                                                                    | 680      |
| jeu de caractère normalisé dans les noms de<br>paramètres linguistiques ..... | 266      |
| journalisation .....                                                          | 276, 289 |
| journalisation, anonymisation .....                                           | 291      |
| journaux de construction, accès .....                                         | 193      |
| journaux de construction, publication .....                                   | 223      |
| journaux de construction, verbosité .....                                     | 182      |
| journaux, rotation .....                                                      | 289      |
| JSON .....                                                                    | 71       |
| JSON, import .....                                                            | 200      |

**K**

|                  |     |
|------------------|-----|
| keepalived ..... | 322 |
| Kerberos .....   | 433 |

**L**

|                                                               |     |
|---------------------------------------------------------------|-----|
| l10n .....                                                    | 696 |
| Langage de programmation Rust .....                           | 128 |
| Langage de programmation Scheme, débiter ..                   | 107 |
| ld-wrapper .....                                              | 101 |
| LDAP .....                                                    | 435 |
| le Hurd .....                                                 | 515 |
| les autorisations des canaux .....                            | 80  |
| Let's Encrypt .....                                           | 461 |
| licence, des paquets .....                                    | 111 |
| licence, GNU Free Documentation License .....                 | 702 |
| liens symboliques, guix shell .....                           | 88  |
| lightdm, gestionnaire de<br>connexion graphique .....         | 330 |
| limites de session .....                                      | 282 |
| linker wrapper .....                                          | 101 |
| lirc .....                                                    | 561 |
| liste de contrôle d'accès (ACL),<br>pour les substituts ..... | 55  |
| logiciel libre .....                                          | 673 |
| lot .....                                                     | 96  |
| lot d'applications .....                                      | 96  |
| lot de logiciels .....                                        | 96  |
| LUKS .....                                                    | 257 |
| LVM, gestionnaire de volume logique .....                     | 257 |

**M**

|                                                                                |          |
|--------------------------------------------------------------------------------|----------|
| métadonnées, canaux .....                                                      | 79       |
| M-x copyright-update .....                                                     | 672      |
| M-x guix-copyright .....                                                       | 672      |
| machine virtuelle .....                                                        | 587, 597 |
| machine virtuelle, installation<br>de Guix System .....                        | 36       |
| man, pages de manuel .....                                                     | 641      |
| mandataire, pendant l'installation du système ..                               | 32       |
| manifest .....                                                                 | 120      |
| manifest de profil .....                                                       | 48       |
| manifeste, export .....                                                        | 53, 86   |
| mappage de périphériques .....                                                 | 256      |
| mcron .....                                                                    | 287, 628 |
| menu de démarrage .....                                                        | 581      |
| message du jour .....                                                          | 269      |
| messagerie instantanée .....                                                   | 400      |
| mettre à jour des paquets .....                                                | 48       |
| mettre à jour Guix .....                                                       | 63       |
| mettre à niveau Guix .....                                                     | 63       |
| minetest .....                                                                 | 198      |
| mise à niveau de Guix pour l'utilisateur root,<br>sur une distro externe ..... | 25       |
| Mise à niveau de Guix, sur une distro externe ..                               | 25       |
| mise à niveau du démon Guix, sur<br>une distro externe .....                   | 25       |
| mises à jour de sécurité .....                                                 | 657      |
| mises à jour non surveillées .....                                             | 322      |
| mises à jour, non surveillées .....                                            | 322      |
| modèles .....                                                                  | 672      |
| ModemManager .....                                                             | 298      |
| Modeswitching .....                                                            | 298      |
| modifications, des définitions de services ..                                  | 584, 637 |
| modprobe .....                                                                 | 556      |
| module, black-list .....                                                       | 577      |
| modules importés, pour les gexps .....                                         | 168      |
| monade .....                                                                   | 162      |
| monade d'état .....                                                            | 164      |
| mot de passe, pour les comptes utilisateurs .....                              | 262      |
| mpd .....                                                                      | 503      |
| MTA (agent de transfert de courriel) .....                                     | 394      |
| multi-versionnement des paquets .....                                          | 183      |
| Mumble .....                                                                   | 411      |
| Mumi, interface web pour debugs .....                                          | 453      |
| Murmur .....                                                                   | 411      |

**N**

|                                            |     |
|--------------------------------------------|-----|
| name service cache daemon .....            | 274 |
| name service caching daemon (nscd) .....   | 23  |
| name service switch .....                  | 573 |
| name service switch, glibc .....           | 23  |
| nar bundle, format d'archive .....         | 72  |
| nar, format d'archive .....                | 72  |
| Nettoyer la base de données du dépôt ..... | 63  |
| Network information service (NIS) .....    | 23  |
| NetworkManager .....                       | 296 |
| NFS .....                                  | 485 |

- NFS, serveur ..... 485  
nftables ..... 302  
NIC, contrôleur d'interface réseau ..... 293  
NIS (Network information service) ..... 23  
Nix ..... 566  
nofile ..... 282  
nom de paramètre linguistique ..... 266  
nom du paquet ..... 674  
nomenclature (manifestes) ..... 120  
non-déterminisme, dans les  
  constructions des paquets ..... 227  
nouveautés des canaux ..... 65  
nouveautés, canaux ..... 81  
nscd ..... 274  
nscd (name service caching daemon) ..... 23  
nscd, invalidation du cache ..... 274  
nslcd, service LDAP ..... 435  
nss-certs ..... 24, 572  
nss-mdns ..... 573  
NSS (name service switch), glibc ..... 23  
nsswitch.conf ..... 23  
NSS ..... 573  
NTP (Network Time Protocol), service ..... 302  
ntpd, service pour le démon  
  Network Time Protocol ..... 302  
numéro de version, pour les instantanés des  
  systèmes de contrôle de version ..... 675
- ## O
- objets simili-fichiers ..... 172  
OCaml ..... 203  
on-error ..... 591  
oom ..... 555  
OPAM ..... 203  
opendht, service réseau de table de  
  hash distribuée ..... 306  
OpenNTPD ..... 303  
OpenPGP, commits signés ..... 692  
optimisation, du code des paquets ..... 183
- ## P
- périphérique de rebouclage ..... 295  
périphériques d'espaces d'échange ..... 248  
périphériques mappés ..... 256  
Périphériques blocs compressés  
  basés sur la RAM ..... 558  
pack ..... 96  
pages de manuel ..... 641  
pam-krb5 ..... 434  
pam-mount ..... 548  
PAM ..... 250  
paquet, taille ..... 215  
paquets ..... 43  
paquets avec plusieurs résultats ..... 59  
paquets importés ..... 197  
paquets inférieurs ..... 68, 69  
paquets L<sup>A</sup>T<sub>E</sub>X ..... 655  
paquets personnels (canaux) ..... 78  
paquets réglables ..... 183  
paquets T<sub>E</sub>X ..... 655  
paquets, chercher des erreurs ..... 213  
paquets, création ..... 673  
paramètres linguistiques ..... 265  
partager des éléments du dépôt entre  
  plusieurs machines ..... 229  
Passerelle IRC ..... 407  
Patchwork ..... 450  
pcscd ..... 560  
performance, réglage du code ..... 183  
perl ..... 679  
personnalisation des paquets ..... 116  
personnalisation des services ..... 241  
personnalisation, des paquets ..... 1, 104  
phases de construction ..... 125, 142, 151  
phases de construction, modifier ..... 151  
phases de construction, personnalisation ..... 145  
phases de construction, pour les paquets ..... 143  
php-fpm ..... 454  
PID 1 ..... 607  
pipefs ..... 487  
pluggable authentication modules ..... 250  
point d'entrée, pour les images Docker ..... 99  
Points d'accès Wi-Fi, service hostapd ..... 300  
polices ..... 24, 682  
POP ..... 395  
postgis ..... 367  
postgresql extension-packages ..... 367  
principale URL, canaux ..... 81  
priorités ..... 282  
prise en charge de SIMD ..... 183  
prise en charge des langues ..... 696  
prise en charge du matériel sur Guix System ..... 26  
profil ..... 40, 44, 45  
profil, choix du ..... 50  
profil, collisions ..... 50  
programmes setgid ..... 571  
programmes setuid ..... 571  
prometheus-node-exporter ..... 429  
protection contre les attaques dites  
  de "downgrade" ..... 66  
Proxy HTTP, pour guix-daemon ..... 278  
proxy, pour l'accès guix-daemon HTTP ..... 278  
pull ..... 63  
PulseAudio, support du son ..... 364  
pypi ..... 197  
python ..... 678
- ## Q
- QEMU ..... 597  
QEMU, réseau ..... 295  
quote ..... 106  
quoting ..... 106  
quoting du code de construction ..... 166

**R**

règles de style ..... 210  
 réécriture d'entrées ..... 119  
 réécriture de l'arbre des dépendances ..... 119  
 réduire la quantité de code commun ..... 672  
 régionalisation, en dehors de Guix System ..... 22  
 réglage, du code des paquets ..... 183  
 réparer GRUB, par un chroot ..... 38  
 réparer le dépôt ..... 62  
 réparer les éléments du dépôt ..... 192  
 répertoire d'état ..... 9  
 répertoires liés aux distro externes ..... 5  
 réplication, des environnements logiciels ..... 44  
 répliquer Guix ..... 66, 70, 76  
 réseau privé virtuel (VPN) ..... 479  
 réseau, avec QEMU ..... 295  
 résumé ..... 688  
 résumé du paquet ..... 676  
 racine du ramasse-miettes, pour les  
   environnements ..... 89, 92  
 Racines du GC ..... 20, 60  
 Racines du GC, ajout ..... 192  
 racines du ramasse-miettes ..... 20, 60  
 racines du ramasse-miettes, pour les packs .... 101  
 ramasse-miettes ..... 60  
 rapports de bogue, suivi ..... 689  
 rasdaemon ..... 557  
 read-eval-print loop ..... 670  
 reconnaissance de motif ..... 683  
 references ..... 159  
 remplacement de paquet, pour les greffes ..... 657  
 REPL ..... 670  
 REPL, boucle de  
   lecture-évaluation-impression, script ..... 176  
 REPL, read-eval-print loop ..... 177  
 reproductibilité ..... 44, 70  
 reproductibilité, de Guix ..... 66, 76  
 reproductibilité, vérification ..... 192  
 resolution ..... 583, 584  
 revenir en arrière ..... 49, 65, 586, 638  
 revenir en arrière dans la  
   configuration du système ..... 246  
 rottlog ..... 289  
 rpc\_pipefs ..... 487  
 rpcbind ..... 486  
 rshiny ..... 565  
 RUNPATH, validation ..... 126, 144  
 rust ..... 680  
 RYF, Respects Your Freedom ..... 26

**S**

sécurité ..... 55  
 sécurité, guix pull ..... 63  
 sécurité, guix-daemon ..... 16  
 série de correctifs ..... 687  
 sac (représentation à  
   bas-niveau des paquets) ..... 124  
 Samba ..... 488  
 sans-fil ..... 31  
 script d'installation ..... 5  
 secrets, service Knot ..... 472  
 SELinux, installation de la politique ..... 16  
 SELinux, limites ..... 16  
 SELinux, politique du démon ..... 16  
 serveur privé virtuel (VPS) ..... 36  
 Serveur SSH ..... 310, 311, 598  
 Serveur VoIP ..... 411  
 serveurs de substituts, en ajouter ..... 56  
 service caché ..... 308  
 Service cgit ..... 531  
 service Gitile ..... 545  
 Service Gitolite ..... 543  
 service hostapd, pour les points  
   d'accès Wi-Fi ..... 300  
 service système ..... 600  
 services ..... 240, 599  
 services essentiels ..... 249  
 services personnels ..... 622  
 services ponctuels, pour le Shepherd ..... 608  
 services shepherd ..... 607  
 services shepherd, pour les  
   utilisateurs et utilisatrices ..... 630  
 services systèmes ..... 267  
 sh, dans /bin ..... 268  
 shell ..... 625, 636  
 shell de connexion ..... 625  
 shell interactif ..... 625  
 shell-profile ..... 636  
 signature, archives ..... 73  
 signatures numériques ..... 57  
 Singularity, construire une image  
   avec guix pack ..... 97  
 Singularity, service de conteneurs ..... 564  
 site officiel ..... 8  
 SMB ..... 488  
 SMTP ..... 394  
 Software Heritage, archive de code source ..... 213  
 sorties ..... 59  
 sorties de paquets ..... 59  
 soumission de correctifs, suivi ..... 689  
 source, vérification ..... 190  
 souris ..... 280  
 sous-répertoire, canaux ..... 79  
 spice ..... 561  
 SQL ..... 366  
 SquashFS, construire une image  
   avec guix pack ..... 97  
 SSH ..... 310, 311, 598

SSH, clefs autorisées ..... 313  
 SSH, copie d'éléments du dépôt ..... 229  
 stackage ..... 201  
 standard de la hiérarchie des systèmes  
   de fichiers (FHS) ..... 88  
 statistiques sur les substituts ..... 231  
 statut du déchargement ..... 16  
 stratégie de branche ..... 685  
 stratégie de planification des reconstructions .. 685  
 stratégie en cas d'erreur ..... 591  
 stratégie on-error ..... 591  
 strate de code ..... 166  
 style de code ..... 683  
 substitution ..... 673  
 substituts ..... 18, 44, 54  
 substituts, autorisations ..... 7, 55, 276  
 substituts, comment les désactiver ..... 56  
 substituts, disponibilité ..... 231  
 substituts, quand les utiliser ..... 677  
 sudo vs. guix pull ..... 35  
 suite de tests ..... 9  
 suite de tests, passer outre ..... 188  
 suivi de la provenance, des artefacts logiciels.... 44  
 suivi de provenance, pour  
   l'environnement personnel ..... 637  
 suivi de provenance, pour le  
   système d'exploitation ..... 585, 590, 606  
 suivi de tickets ..... 689  
 support des imprimantes avec CUPS ..... 334  
 support du son ..... 364  
 suppression de paquets ..... 44  
 supprimer des générations du  
   dossier personnel ..... 638  
 supprimer des générations du système ..... 586  
 supprimer des paquets ..... 44  
 syncthing ..... 310  
 sysctl ..... 560  
 syslog ..... 276  
 système d'init ..... 607  
 système de construction ..... 124  
 Système de construction Android NDK ..... 127  
 système de construction Clojure simple ..... 130  
 Système de construction GNU ..... 106  
 Système de fenêtrage X ..... 325  
 système de sécurité global ..... 487  
 Système Guix ..... 1, 2

## T

tâches planifiées ..... 287, 628  
 téléchargement du Guix binaire ..... 6  
 télécharger les sources des paquets ..... 195  
 téléphonie, services ..... 408  
 taille ..... 215  
 temps réel ..... 282  
 test du déchargement ..... 15  
 TeX Live ..... 199  
 thermald ..... 502

tlp ..... 495  
 TLS ..... 572  
 Tor ..... 307  
 traduction ..... 696  
 transactions ..... 43, 44  
 transactions, défaire ..... 49, 65  
 transférer des éléments du dépôt entre  
   plusieurs machines ..... 229  
 transformations de paquets ..... 118  
 transformations de paquets, mises à jour ..... 48  
 type de service ..... 604  
 types de services ..... 600  
 types de sessions ..... 325

## U

UEFI, chargeur d'amorçage ..... 578  
 UEFI, installation ..... 32  
 ulimit ..... 282  
 update-guix-package, mise à jour  
   du paquet guix ..... 695  
 USB\_ModeSwitch ..... 298  
 utilisateurs ..... 260  
 utilisation interactive ..... 177

## V

vérification d'intégrité ..... 62  
 valeurs monadiques ..... 162  
 variables d'environnement ..... 622  
 variantes de paquets ..... 183  
 variantes, des paquets ..... 116  
 Varnish ..... 449  
 verbosité, des outils en ligne de commande .... 182  
 version du paquet ..... 674  
 Vim, édition de code Scheme ..... 683  
 virgule (unquote) ..... 106  
 VM ..... 587  
 VNC (réseau privé virtuel) ..... 477  
 VPN (réseau privé virtuel) ..... 479  
 VPS (serveur privé virtuel) ..... 36  
 vulnérabilités ..... 214, 657

## W

weather, disponibilité des substituts ..... 231  
 Web ..... 461  
 web ..... 441  
 WebSSH ..... 316  
 wesnothd ..... 547  
 WiFi ..... 31  
 WiFi, support matériel ..... 26  
 WPA Supplicant ..... 297  
 wssd, Démon de découverte des services web .. 489  
 www ..... 441



**X**

|                                                                    |     |
|--------------------------------------------------------------------|-----|
| X11 .....                                                          | 325 |
| XDMCP (protocole de contrôle de<br>gestion de l'affichage X) ..... | 477 |
| XKB, disposition du clavier .....                                  | 263 |
| <code>xlsfonts</code> .....                                        | 24  |
| XMPP .....                                                         | 400 |
| Xorg, configuration .....                                          | 333 |

|                          |    |
|--------------------------|----|
| <code>xterm</code> ..... | 24 |
|--------------------------|----|

**Z**

|                               |          |
|-------------------------------|----------|
| zabbix zabbix-agent .....     | 431      |
| zabbix zabbix-front-end ..... | 432      |
| zabbix zabbix-server .....    | 429      |
| zram .....                    | 558      |
| zsh .....                     | 625, 636 |

## Index de programmation

### #

#~exp..... 169

### %

%store-directory..... 147

,

'..... 106

(

(gexp..... 169

,

,..... 106

>

>>=..... 163

‘

`..... 106

## A

accountsservice-service..... 352

add-text-to-store..... 158

agate-service-type..... 460

agetty-service..... 270

alsa-service-type..... 364

ar-file?..... 148

autossh-service-type..... 315

## B

base-initrd..... 577

binary-file..... 166

bluetooth-service..... 355

bluetooth-service-type..... 355

build..... 179

build-derivations..... 159

build-expression->derivation..... 161

## C

cat-avatar-generator-service..... 458

close-connection..... 158

colord-service-type..... 354

computed-file..... 172

concatenate-manifests..... 123

configuration->documentation..... 616

connman-service-type..... 297

copy-recursively..... 148

cups-service-type..... 334

current-state..... 165

## D

dbus-service..... 350

debootstrap-os..... 522

debootstrap-variant..... 522

define-configuration..... 612

define-maybe..... 614

define-record-type\*..... 683

delete-file-recursively..... 148

derivation..... 160

dhcpd-service-type..... 299

dicod-service..... 562

directory-exists?..... 147

directory-union..... 174

dnsmasq-service-type..... 473

dovecot-service..... 372

dropbear-service..... 314

## E

earlyoom-service-type..... 556

elf-file?..... 148

elm->package-name..... 681

elm-package-origin..... 681

elogind-service..... 350

empty-serializer..... 615

enlightenment-desktop-service-type..... 349

enter-store-monad..... 179

evaluate-search-paths..... 156

executable-file?..... 147

exim-service-type..... 394

expression->initrd..... 578

extra-special-file..... 269

**F**

fail2ban-jail-service ..... 567  
 file->udev-rule ..... 279  
 file-append ..... 174  
 file-name-predicate ..... 149  
 file-system-label ..... 251, 253  
 file-union ..... 174  
 find-files ..... 149  
 fold-services ..... 605

**G**

generate-documentation ..... 615  
 geoclue-application ..... 355  
 geoclue-service ..... 355  
 getmail-service-type ..... 395  
 gexp->approximate-sexp ..... 176  
 gexp->derivation ..... 171  
 gexp->file ..... 173  
 gexp->script ..... 173  
 gexp? ..... 171  
 git-daemon-service ..... 529  
 git-fetch ..... 115  
 git-http-nginx-location-configuration ..... 531  
 git-version ..... 676  
 gmnisrv-service-type ..... 459  
 grub-theme ..... 583  
 guix-os ..... 523  
 guix-package->elm-name ..... 681  
 guix-publish-service-type ..... 281  
 guix-variant ..... 523  
 gzip-file? ..... 148

**H**

hg-fetch ..... 116  
 hg-version ..... 676  
 home-environment ..... 620  
 host-name-service ..... 269  
 httpd-config-file ..... 442  
 httpd-configuration ..... 442  
 httpd-module ..... 442  
 httpd-service-type ..... 441  
 httpd-virtualhost ..... 444

**I**

imap4d-service-type ..... 399  
 inetd-service-type ..... 304  
 inferior-elm-package-name ..... 681  
 inferior-for-channels ..... 69  
 inferior-package-description ..... 69  
 inferior-package-home-page ..... 69  
 inferior-package-inputs ..... 69  
 inferior-package-location ..... 69  
 inferior-package-name ..... 69  
 inferior-package-native-inputs ..... 69  
 inferior-package-native-search-paths ..... 69

inferior-package-propagated-inputs ..... 69  
 inferior-package-search-paths ..... 69  
 inferior-package-synopsis ..... 69  
 inferior-package-transitive-  
   native-search-paths ..... 69  
 inferior-package-transitive-  
   propagated-inputs ..... 69  
 inferior-package-version ..... 69  
 inferior-package? ..... 69  
 inferior-packages ..... 69  
 inputattach-service-type ..... 561  
 install-file ..... 148  
 interned-file ..... 166  
 invoke ..... 150  
 invoke-error-arguments ..... 150  
 invoke-error-exit-status ..... 150  
 invoke-error-program ..... 150  
 invoke-error-stop-signal ..... 150  
 invoke-error-term-signal ..... 150  
 invoke-error? ..... 150  
 invoke/quiet ..... 151

**K**

keepalived-service-type ..... 322  
 kernel-module-loader-service-type ..... 557  
 keyboard-layout ..... 263  
 kmscon-service-type ..... 273  
 knot-resolver-service-type ..... 472  
 knot-service-type ..... 465

**L**

ladspa-service-type ..... 366  
 let-system ..... 175  
 libvirt-service-type ..... 505  
 lirc-service ..... 561  
 local-file ..... 172  
 login-service ..... 269  
 lookup-inferior-packages ..... 69  
 lookup-package-direct-input ..... 111  
 lookup-package-input ..... 111  
 lookup-package-native-input ..... 111  
 lookup-package-propagated-input ..... 111  
 lookup-qemu-platforms ..... 514  
 lower ..... 179  
 lower-object ..... 176  
 lsh-service ..... 310  
 lxqt-desktop-service-type ..... 349

**M**

|                                |          |
|--------------------------------|----------|
| mail-aliases-service-type..... | 399      |
| make-file-writable.....        | 148      |
| match-record.....              | 683      |
| mate-desktop-service-type..... | 349      |
| maybe-value-set?.....          | 615      |
| mbegin.....                    | 164      |
| mingetty-service.....          | 269      |
| mixed-text-file.....           | 174      |
| mkdir-p.....                   | 148      |
| mlet.....                      | 164      |
| mlet*.....                     | 164      |
| modify-inputs.....             | 117      |
| modify-phases.....             | 152      |
| modify-services.....           | 241, 603 |
| unless.....                    | 164      |
| mwhen.....                     | 164      |

**N**

|                          |     |
|--------------------------|-----|
| nginx-configuration..... | 445 |
| nginx-php-location.....  | 457 |
| nginx-service-type.....  | 444 |
| nscd-service.....        | 274 |

**O**

|                                  |     |
|----------------------------------|-----|
| open-connection.....             | 158 |
| open-inferior.....               | 69  |
| openntpd-service-type.....       | 303 |
| opensmtpd-service-type.....      | 394 |
| openssh-service-type.....        | 311 |
| openvpn-client-service.....      | 480 |
| openvpn-server-service.....      | 480 |
| openvswitch-service-type.....    | 319 |
| operating-system.....            | 238 |
| operating-system-derivation..... | 246 |
| options->transformation.....     | 118 |

**P**

|                                    |         |
|------------------------------------|---------|
| package->cross-derivation.....     | 166     |
| package->derivation.....           | 166     |
| package->development-manifest..... | 123     |
| package->manifest-entry.....       | 123     |
| package-cross-derivation.....      | 108     |
| package-derivation.....            | 108     |
| package-development-inputs.....    | 112     |
| package-file.....                  | 166     |
| package-input-rewriting.....       | 119     |
| package-input-rewriting/spec.....  | 119     |
| package-mapping.....               | 120     |
| package-name->name+version.....    | 147     |
| package-with-c-toolchain.....      | 112     |
| packages->manifest.....            | 49, 123 |
| pam-limits-service.....            | 282     |
| plain-file.....                    | 172     |
| polkit-service.....                | 352     |

|                                   |     |
|-----------------------------------|-----|
| postgresql-role-service-type..... | 369 |
| program-file.....                 | 173 |
| prosody-service-type.....         | 400 |
| pulseaudio-service-type.....      | 365 |

**Q**

|                         |     |
|-------------------------|-----|
| qemu-platform-name..... | 514 |
| qemu-platform?.....     | 514 |
| quasiquote.....         | 106 |
| quote.....              | 106 |

**R**

|                            |     |
|----------------------------|-----|
| radicale-service-type..... | 400 |
| raw-initrd.....            | 577 |
| report-invoke-error.....   | 151 |
| reset-gzip-timestamp.....  | 148 |
| return.....                | 163 |
| rngd-service.....          | 282 |
| rsync-service-type.....    | 308 |
| run-in-store.....          | 179 |
| run-with-state.....        | 165 |
| run-with-store.....        | 165 |

**S**

|                                    |          |
|------------------------------------|----------|
| scheme-file.....                   | 173      |
| screen-locker-service.....         | 334      |
| search-input-directory.....        | 150      |
| search-input-file.....             | 150      |
| serialize-configuration.....       | 615      |
| service.....                       | 602      |
| service-extension.....             | 605      |
| service-extension?.....            | 605      |
| service-kind.....                  | 603      |
| service-value.....                 | 603      |
| service?.....                      | 603      |
| set-current-state.....             | 165      |
| set-xorg-configuration.....        | 264, 334 |
| shepherd-configuration-action..... | 611      |
| simple-service.....                | 605      |
| source-module-closure.....         | 168      |
| specification->package.....        | 240      |
| specifications->manifest.....      | 124      |
| spice-vdagent-service.....         | 561      |
| state-pop.....                     | 165      |
| state-push.....                    | 165      |
| store-file-name?.....              | 147      |
| strip-store-file-name.....         | 147      |
| substitute*.....                   | 149      |
| symbolic-link?.....                | 147      |
| syncthing-service-type.....        | 310      |
| syslog-service.....                | 276      |

**T**

text-file..... 165  
text-file\*..... 173  
this-operating-system..... 250  
this-package..... 111  
tlp-service-type..... 495  
tor-hidden-service..... 308  
transmission-daemon-service-type..... 415  
transmission-password-hash..... 416  
transmission-random-salt..... 416

**U**

udev-rule..... 279  
udev-rule-service..... 279  
udev-service..... 279  
udisks-service..... 354  
unquote..... 106  
url-fetch..... 115  
uuid..... 251, 254

**V**

valid-path?..... 158  
verbosity..... 179  
virtlog-service-type..... 512

**W**

webssh-service-type..... 316  
which..... 150  
with-directory-excursion..... 148  
with-extensions..... 169, 170  
with-imported-modules..... 168, 170  
with-monad..... 163  
with-parameters..... 175  
wrap-program..... 153  
wrap-script..... 153

**X**

xorg-start-command..... 334

**Z**

zram-device-service-type..... 558