

Manuel Live Systems

Projet Live Systems <debian-live@lists.debian.org>

Copyright © 2006-2014 Live Systems Project

Ce programme est un logiciel libre ; vous pouvez le redistribuer ou le modifier suivant les termes de la Licence Générale Publique GNU telle que publiée par la Free Software Foundation : soit la version 3 de cette licence, soit (à votre gré) toute version ultérieure.

Ce programme est distribué dans l'espoir qu'il vous sera utile, mais SANS AUCUNE GARANTIE : sans même la garantie implicite de COMMERCIALISABILITÉ ni d'ADÉQUATION À UN OBJECTIF PARTICULIER. Consultez la Licence Générale Publique GNU pour plus de détails.

Vous devriez avoir reçu une copie de la Licence Générale Publique GNU avec ce programme ; si ce n'est pas le cas, consultez < <http://www.gnu.org/licenses/> >.

Le texte complet de la Licence Générale Publique GNU peut être trouvé dans le fichier /usr/share/common-licenses/GPL-3

Contents

À propos	2
À propos de ce manuel	3
1. À propos de ce manuel	3
1.1 Pour les impatientes	3
1.2 Terminologie	3
1.3 Auteurs	4
1.4 Contribuer à ce document	5
1.4.1 Appliquer des modifications	5
1.4.2 Traduction	6
À propos du Live Systems Project	8
2. À propos du Live Systems Project	8
2.1 Motivation	8
2.1.1 Ce qui ne va pas avec les systèmes live actuels	8
2.1.2 Pourquoi créer notre propre système live ?	8
2.2 Philosophie	8
2.2.1 Seulement des paquets inchangés de Debian « main »	8
2.2.2 Pas de configuration des paquets du système live	9
2.3 Contact	9
Utilisateur	10

Installation	11
3. Installation	11
3.1 Exigences	11
3.2 Installation de live-build	11
3.2.1 À partir du dépôt Debian	11
3.2.2 À partir du code source	11
3.2.3 À partir des instantanés	12
3.3 Installation de live-boot et live-config	12
3.3.1 À partir du dépôt Debian	12
3.3.2 À partir du code source	12
3.3.3 À partir des instantanés	13
Les bases	14
4. Les bases	14
4.1 Qu'est-ce qu'un système live ?	14
4.2 Téléchargement des images précompilées	15
4.3 Utiliser le constructeur web d'images live	15
4.3.1 Utilisation du constructeur web et avertissements	15
4.4 Premières étapes : la construction d'une image ISO hybride	16
4.5 Utilisation d'une image ISO hybride live	16
4.5.1 Graver une image ISO sur un support physique	16
4.5.2 Copie d'une image ISO hybride sur une clé USB	17
4.5.3 Utilisation de l'espace disponible sur une clé USB	17
4.5.4 Démarrer le support live	17
4.6 Utiliser une machine virtuelle pour les tests	18
4.6.1 Test d'une image ISO avec QEMU	18
4.6.2 Test d'une image ISO avec VirtualBox	19

4.7 Construire et utiliser une image HDD	19
4.8 Construction d'une image netboot	20
4.8.1 Serveur DHCP	21
4.8.2 Serveur TFTP	21
4.8.3 Serveur NFS	21
4.8.4 Guide pratique pour expérimenter avec une image Netboot	22
4.8.5 Qemu	22
4.9 Webbooting	22
4.9.1 Obtenir les fichiers webboot	22
4.9.2 Démarrer images webboot	23
Aperçu des outils	24
5. Aperçu des outils	24
5.1 Le paquet live-build	24
5.1.1 La commande lb config	24
5.1.2 La commande lb build	25
5.1.3 La commande lb clean	25
5.2 Le paquet live-boot	25
5.3 Le paquet live-config	25
Gestion d'une configuration	27
6. Gestion d'une configuration	27
6.1 Gérer les modifications de la configuration	27
6.1.1 Pourquoi utiliser des scripts auto ? Que font-ils ?	27
6.1.2 Utiliser les scripts auto d'exemple	27
6.2 Cloner une configuration publiée via Git	28

Personnalisation des contenus	30
7. Vue d'ensemble de la personnalisation	30
7.1 Configuration pendant la construction vs. l'amorçage	30
7.2 Étapes de la construction	30
7.3 Supplément lb config avec des fichiers	31
7.4 Tâches de personnalisation	31
Personnalisation de l'installation de paquets	32
8. Personnalisation de l'installation de paquets	32
8.1 Sources des paquets	32
8.1.1 Distribution, zones d'archive et mode	32
8.1.2 Miroirs de distribution	33
8.1.3 Miroirs de distribution utilisés lors de la construction	33
8.1.4 Miroirs de distribution utilisés pendant l'exécution	33
8.1.5 Dépôts additionnels	33
8.2 Choisir les paquets à installer	34
8.2.1 Listes de paquets	34
8.2.2 Utilisation des métapaquets	34
8.2.3 Listes de paquets locaux	35
8.2.4 Listes de paquets locaux pour l'étape binary	35
8.2.5 Listes de paquets générées	36
8.2.6 Utiliser des conditions dans les listes de paquets	36
8.2.7 Suppression de paquets lors de l'installation	36
8.2.8 Tâches de bureau et de langue	37
8.2.9 Version et type de noyau	37
8.2.10 Noyaux personnalisés	38

8.3 Installation de paquets modifiés ou tiers	38	10.3 Persistance	48
8.3.1 Utiliser packages.chroot pour installer des paquets personnalisés	39	10.3.1 Le fichier persistence.conf	49
8.3.2 Utiliser un dépôt APT pour installer des paquets personnalisés.	39	10.3.2 Utilisation de plusieurs dispositifs de persistance	50
8.3.3 Les paquets personnalisés et APT	39	10.4 Utilisation de la persistance avec chiffrement	50
8.4 Configuration d'APT pendant la construction	39		
8.4.1 Choisir apt ou aptitude	39	Personnalisation de l'image binaire	53
8.4.2 Utilisation d'un proxy avec APT	40		
8.4.3 Régler APT pour économiser de l'espace	40	11. Personnalisation de l'image binaire	53
8.4.4 Passer des options à apt ou aptitude	41	11.1 Chargeurs d'amorçage	53
8.4.5 APT pinning	41	11.2 Métadonnées ISO	53
Personnalisation des contenus	43		
9. Personnalisation des contenus	43	Personnalisation de l'installateur Debian	54
9.1 Includes	43		
9.1.1 Live/chroot local includes	43	12. Personnalisation du contenu pour l'installateur Debian	54
9.1.2 Binary local includes	44	12.1 Types d'installateur Debian	54
9.2 Hooks	44	12.2 Personnalisation de l'installateur Debian par préconfiguration	55
9.2.1 Live/chroot local hooks	44	12.3 Personnalisation de contenu pour l'Installateur Debian	55
9.2.2 Hooks pendant le démarrage	44		
9.2.3 Binary local hooks	44	Projet	56
9.3 Préconfigurer questions de debconf	45		
		Contribuer au projet	57
Personnalisation des comportements pendant l'exécution	46		
10. Personnalisation des comportements pendant l'exécution	46	13. Contribuer au projet	57
10.1 Personnalisation de l'utilisateur live	46	13.1 Faire des changements	57
10.2 Personnalisation des paramètres régionaux et de la langue	46		

Signaler des bogues	59	16.2 Évolutions mineures	65
14. Signaler des bogues	59	16.2.1 Dernière évolution mineure d'une version Debian	65
14.1 Problèmes connus	59	16.2.2 Modèle pour l'annonce d'une évolution mineure	65
14.2 Reconstruire à partir de zéro	59	Dépôts Git	67
14.3 Utiliser des paquets mis à jour	59	17. Dépôts Git	67
14.4 Recueillir l'information	59	17.1 Gestion de multiples dépôts	67
14.5 Isoler le cas qui échoue, si possible	60	Exemples	69
14.6 Utiliser le paquet adéquat pour rapporter un bogue	60	Exemples	70
14.6.1 Pendant la construction durant l'amorçage	61	18. Exemples	70
14.6.2 Pendant la construction durant l'installation de paquets	61	18.1 Utiliser les exemples	70
14.6.3 Pendant le démarrage	61	18.2 Tutoriel 1 : Une image par défaut	70
14.6.4 Pendant l'exécution	61	18.3 Tutoriel 2 : Un utilitaire d'un navigateur Web	71
14.7 Effectuer une recherche	61	18.4 Tutoriel 3 : Une image personnalisée	71
14.8 Où rapporter les bogues	62	18.4.1 Première révision	71
Style de code	63	18.4.2 Deuxième révision	72
15. Style du code	63	18.5 Un client kioske VNC	73
15.1 Compatibilité	63	18.6 Une image de base pour une clé USB de 128 Mo	74
15.2 Indentation	63	18.7 Un bureau GNOME localisé avec un installateur	75
15.3 Adaptateur	63	Appendix	77
15.4 Variables	63		
15.5 Autres	64		
Procédures	65		
16. Procédures	65		
16.1 Évolutions majeures	65		

Guide de style	78
19. Guide de style	78
19.1 Lignes directrices pour les auteurs	78
19.1.1 Caractéristiques linguistiques	78
19.1.2 Procédures	80
19.2 Lignes directrices pour les traducteurs	82
19.2.1 Conseils de traduction	82
SiSU Metadata, document information	83

1	Manuel Live Systems
---	----------------------------

2 **À propos**

À propos de ce manuel

1. À propos de ce manuel

L'objectif principal de ce manuel est de servir de point d'accès unique à tous les documents liés au Live Systems Project et particulièrement aux logiciels produits par le projet pour Debian 8.0 « **jessie** ». Une version mise à jour peut toujours être trouvée sur <http://live-systems.org/> .

Ce manuel a principalement pour but de vous aider à construire un système live et non pas de s'articuler autour des sujets relatifs à l'utilisateur final. Toutefois, l'utilisateur final peut trouver des informations utiles dans ces sections : < **Les Bases** > couvrent le téléchargement des images précompilées et la préparation des images pour être démarrées sur les supports ou sur le réseau, soit en utilisant le constructeur web, soit en exécutant *live-build* directement sur le système. < **Personnalisation des comportements pendant l'exécution** > décrit certaines options qui peuvent être indiquées à l'invite de démarrage, telles que la sélection d'un clavier, des paramètres régionaux et la persistance.

Certaines commandes mentionnées dans le texte doivent être exécutées avec les privilèges de super-utilisateur, qui peuvent être obtenus à l'aide de la commande `su` ou en utilisant `sudo`. Afin de distinguer les commandes qui peuvent être exécutées par un utilisateur sans privilège de celles nécessitant les privilèges de super-utilisateur, les commandes sont précédées respectivement par `$` ou `#`. Notez que ce symbole ne fait pas partie de la commande.

1.1 Pour les impatientes

Même si nous croyons que tout dans ce manuel est important pour

au moins certains de nos utilisateurs, nous nous rendons compte qu'il y a beaucoup de matière à couvrir et que vous pouvez vouloir expérimenter avant d'entrer dans les détails. Par conséquent, nous vous suggérons de lire dans l'ordre suivant.

Tout d'abord, lisez ce chapitre < **À propos de ce manuel** > dès le début et finissant avec la section < **Terminologie** >. Ensuite, passez aux trois tutoriels à l'avant de la section < **Exemples** > destinée à vous apprendre la construction de l'image et les bases de la personnalisation. Lisez en premier < **En utilisant les exemples** >, puis < **Tutoriel 1 : Une image par défaut** >, < **Tutoriel 2 : Un logiciel de navigateur Web** > et finalement < **Tutoriel 3 : Une image personnalisée** >. À la fin de ces tutoriels, vous aurez un avant-goût de ce qui peut être fait avec les systèmes live.

Nous vous encourageons à revenir à l'étude plus approfondie du manuel, en poursuivant par exemple votre lecture par < **Les bases** >, < **Construire une image netboot** > et finissant par la lecture de la < **Vue d'ensemble de la personnalisation** > et les sections suivantes. Après cela, nous espérons que vous serez complètement excités par ce qui peut être fait avec les systèmes live et motivés pour lire le reste du manuel, du début à la fin.

1.2 Terminologie

- **Système Live** : Un système d'exploitation pouvant être démarré sans installation préalable sur un disque dur. Les systèmes live ne modifient pas le système d'exploitation local ou les fichiers installés sur le disque dur sans qu'on leur en donne explicitement l'instruction. D'habitude, les systèmes live sont démarrés à partir des supports tels que des CDs, DVDs, ou des clés USB. Certains systèmes peuvent également être démarrés sur le réseau (via des images netboot, voir < **Construction d'une image netboot** >), et sur l'Internet (via le paramètre d'amorçage `fetch=URL`, voir

< **Webbooting** >).

- **Support live** : À la différence du système live, le support live se réfère au CD, DVD ou clé USB où l'image binaire produite par *live-build* et utilisée pour démarrer le système live est écrite. D'une manière générale, le terme désigne également tout emplacement où réside l'exécutable qui permet de démarrer le système live, tel que l'emplacement des fichiers de démarrage sur le réseau.
- **Live Systems Project** : Le projet qui maintient, entre autres, les paquets *live-boot*, *live-build*, *live-config*, *live-tools* et *live-manual*.
- **Système hôte** : L'environnement utilisé pour créer le système live.
- **Système cible** : L'environnement utilisé pour faire fonctionner le système live.
- **live-boot** : Une collection de scripts utilisés pour lancer des systèmes live.
- **live-build** : Une collection de scripts utilisés pour construire des systèmes live personnalisés.
- **live-config** : Une collection de scripts utilisés pour configurer un système live pendant le processus d'amorçage.
- **live-tools** : Une collection de scripts supplémentaires utilisés pour effectuer des tâches utiles dans un système live en fonctionnement.
- **live-manual** : Ce document est maintenu dans un paquet nommé *live-manual*.
- **Debian Installer (d-i)** : Le système d'installation officiel pour la distribution Debian.
- **Paramètres d'amorçage** : Les paramètres qui peuvent être

entrés à l'invite de démarrage afin de modifier le noyau ou *live-config*.

- **chroot** : Le logiciel *chroot*, `chroot(8)`, nous permet d'exécuter plusieurs instances concurrentes de l'environnement GNU/Linux sur un système sans redémarrage.
- **Image binaire** : Un fichier contenant le système live, tel que *live-image-i386.hybrid.iso* ou *live-image-i386.img*.
- **Distribution cible** : La distribution sur laquelle votre système live sera basée. Celle-ci peut être différente de la distribution de votre système hôte.
- **stable/testing/unstable** : La distribution **stable**, actuellement nommée **wheezy**, contient la dernière version officielle de Debian. La distribution **testing**, temporairement nommée **jessie**, est la prochaine version **stable** où seulement les paquets suffisamment matures peuvent entrer. Un avantage de cette distribution est qu'elle contient des logiciels de versions plus récentes que la version **stable**. La distribution **unstable**, nommée **sid** de façon permanente, est en constante évolution. En général cette distribution est utilisée par les développeurs et ceux qui aiment le risque. Tout au long du manuel, nous avons tendance à utiliser les noms de code pour les évolutions majeures, tels que **jessie** ou **sid**, car c'est ce qui est pris en charge par les outils eux-mêmes.

1.3 Auteurs

La liste des auteurs (dans l'ordre alphabétique) :

- Ben Armstrong
- Brendan Sleight
- Carlos Zuferri

- Chris Lamb
- Daniel Baumann
- Franklin Piat
- Jonas Stein
- Kai Hendry
- Marco Amadori
- Mathieu Geli
- Matthias Kirschner
- Richard Nelson
- Trent W. Buck

1.4 Contribuer à ce document

Ce manuel est conçu comme un projet communautaire et toutes les propositions d'améliorations et de contributions sont bienvenues. Veuillez consulter la section < [Contribuer au projet](#) > pour des informations détaillées sur la façon d'obtenir une clé et de faire des livraisons (commits).

1.4.1 Appliquer des modifications

Afin d'apporter des modifications au manuel anglais, vous devez modifier les fichiers adéquats dans `manual/en/` mais avant de soumettre votre contribution, veuillez prévisualiser votre travail. Afin de prévisualiser *live-manual*, assurez-vous que les paquets nécessaires sont installés en exécutant :

```
# apt-get install make po4a ruby ruby-nokogiri sisu-complete
```

Vous pouvez compiler *live-manual* dans le répertoire de niveau supérieur de votre Git checkout en exécutant :

```
$ make build
```

Comme il faut un certain temps pour construire le manuel dans toutes les langues prises en charge, les auteurs peuvent trouver pratique d'utiliser l'un des raccourcis de construction rapide lors de la révision de la nouvelle documentation ajoutée au manuel anglais. `PROOF=1` construit *live-manual* au format html, mais sans les fichiers html segmentés, et `PROOF=2` construit *live-manual* au format pdf, mais seulement les portraits A4 et US. C'est pourquoi l'utilisation de l'une ou l'autre des possibilités peut sauver une quantité considérable de temps, par exemple :

```
$ make build PROOF=1
```

Lors de la révision d'une des traductions, il est possible de construire une seule langue en exécutant, par exemple :

```
$ make build LANGUAGES=de
```

Il est également possible de construire par type de document, par exemple,

```
$ make build FORMATS=pdf
```

Ou combiner les deux, par exemple :

```
$ make build LANGUAGES=de FORMATS=html
```

Après avoir relu votre travail et vous être assuré que tout va bien, n'utilisez pas `make commit` à moins que vous mettiez à jour les traductions dans le commit. Dans ce cas, ne mélangez pas les modifications apportées au manuel en anglais et les traductions dans la même livraison, mais utilisez des commits séparés. Consultez la section < **Traduction** > pour plus de détails.

1.4.2 Traduction

Afin de traduire *live-manual*, procédez comme suit, selon que vous commencez une traduction à partir de zéro ou vous travaillez sur une traduction déjà existante :

- Commencer une nouvelle traduction à partir de zéro
 - Traduisez les fichiers **about_manual.ssi.pot**, **about_project.ssi.pot** et **index.html.in.pot** dans `manual/pot/` dans votre langue avec votre éditeur préféré (comme *poedit*) . Envoyez les fichiers `.po` traduits à la liste de diffusion pour vérifier leur intégrité. La vérification d'intégrité de *live-manual* garantit non seulement que les fichiers `.po` sont 100% traduits, mais elle détecte également des erreurs possibles.
 - Pour activer une nouvelle langue dans l'autobuild, il suffit d'ajouter les premiers fichiers traduits à `manual/po/-${LANGUAGE}/` et lancer `make commit`. Modifier ensuite `manual/_sisu/home/index.html` en ajoutant le nom de la langue et son nom en anglais entre parenthèses.
- Continuer avec une traduction déjà commencée

- Si votre langue cible a déjà été ajoutée, vous pouvez continuer avec la traduction des fichiers `.po` dans `manual/-po/${LANGUAGE}/` de façon aléatoire avec votre éditeur préféré (comme *poedit*) .

- N'oubliez pas que vous devez faire un `make commit` pour assurer que la traduction des manuels est mise à jour à partir des fichiers `.po`, alors vous pouvez réviser vos modifications avec `make build` avant `git add .`, `git commit -m "Translating..."` et `git push`. Gardez à l'esprit que `make build` peut prendre un temps considérable, vous pouvez relire les langues individuellement comme expliqué dans < **Appliquer des modifications** >

Après l'exécution de `make commit`, vous verrez beaucoup de texte sur l'écran. Il s'agit essentiellement de messages informatifs sur l'état du processus et de quelques indications sur ce qui peut être fait pour améliorer *live-manual*. Si vous ne voyez aucune erreur fatale, vous pouvez généralement continuer et soumettre votre contribution.

live-manual contient deux utilitaires qui peuvent grandement aider les traducteurs à trouver les textes non traduits et modifiés. Le premier est "make translate". Il lance un script qui vous indique en détail le nombre de messages non traduits qu'il y a dans chaque fichier `.po`. Le second, "make fixfuzzy", n'agit que sur les messages modifiés, mais il vous aide à les trouver et à les résoudre un par un.

Gardez à l'esprit que même si ces utilitaires peuvent être vraiment utiles pour faire un travail de traduction sur la ligne de commande, l'utilisation d'un outil spécialisé comme *poedit* est la méthode recommandée pour effectuer la tâche. C'est aussi une bonne idée de lire la documentation sur localisation de debian (l10n) et, plus particulièrement pour *live-manual*, les < **Lignes directrices pour les**

traducteurs >.

71

Remarque : Vous pouvez utiliser `make clean` pour nettoyer votre arbre git avant de faire un push. Cette étape n'est pas obligatoire grâce au fichier `.gitignore` mais c'est une bonne pratique pour éviter d'envoyer certains fichiers involontairement.

72	À propos du Live Systems Project	85	2.1.2 Pourquoi créer notre propre système live ?
73	2. À propos du Live Systems Project	86	Debian est le système d'exploitation universel : Debian a un système live pour servir de vitrine et pour représenter le vrai, seul et unique système Debian avec les principaux avantages suivants :
74	2.1 Motivation		
75	2.1.1 Ce qui ne va pas avec les systèmes live actuels		
76	Lorsque le Live Systems Project a été lancé, il y avait déjà plusieurs systèmes live basés sur Debian et ils faisaient un excellent travail. Du point de vue de Debian, la plupart d'entre eux ont un ou plusieurs des inconvénients suivants :		
77	• Ce ne sont pas des projets Debian et ils manquent donc de soutien au sein de Debian.		87
78	• Ils mélangent des distributions différentes comme testing et unstable .		88
79	• Ils ne prennent en charge que i386.		
80	• Ils modifient le comportement et/ou l'apparence des paquets en les dépouillant pour économiser de l'espace.		
81	• Ils comprennent des paquets ne provenant pas de l'archive Debian.		
82	• Ils offrent des noyaux personnalisés avec des correctifs supplémentaires qui ne font pas partie de Debian.		
83	• Ils sont gros et lents en raison de leur dimension et donc pas recommandés comme systèmes de sauvetage.		
84	• Ils ne sont pas disponibles en différents formats (CDs, DVDs, clés USB et images netboot).		
			89
			90
			91
			92
			2.2 Philosophie
			93
			2.2.1 Seulement des paquets inchangés de Debian
			94
			« main »
			Nous n'utiliserons que les paquets du dépôt Debian dans la section « main ». La section non-free ne fait pas partie de Debian et ne peut donc pas être utilisée pour les images officielles du système live.
			95
			Nous ne changerons pas les paquets. Chaque fois que nous aurons besoin de changer quelque chose, nous le ferons en coordination avec le responsable du paquet dans Debian.
			96
			À titre d'exception, nos propres paquets tels que <i>live-boot</i> , <i>live-build</i> ou <i>live-config</i> peuvent être utilisés temporairement à partir de notre propre dépôt pour des raisons de développement (par
			97

exemple pour créer des instantanés de développement). Ils seront téléchargés sur Debian régulièrement.

2.2.2 Pas de configuration des paquets du système live

Dans cette phase, nous n'offrirons pas de configurations alternatives. Tous les paquets sont utilisés dans leur configuration par défaut comme ils sont après une installation standard de Debian.

Chaque fois que nous aurons besoin d'une configuration par défaut différente, nous la ferons en coordination avec le responsable du paquet dans Debian.

Un système de configuration des paquets est fourni avec debconf permettant la personnalisation des paquets installés sur vos images live, mais pour les < **images live précompilées** > seulement une configuration par défaut sera utilisée sauf si c'est absolument nécessaire pour fonctionner dans l'environnement live. Autant que possible, nous préférons adapter les paquets dans l'archive Debian de sorte qu'ils fonctionnent mieux dans un système live plutôt que faire des changements à l'ensemble d'outils live ou les < **configurations des images live** >. Pour plus d'informations, veuillez consulter < **Vue d'ensemble de la personnalisation** >.

2.3 Contact

- **Liste de diffusion** : Le contact principal du projet est la liste de diffusion < <https://lists.debian.org/debian-live/> >. Vous pouvez envoyer un courriel à la liste directement en adressant votre courrier à < debian-live@lists.debian.org >. Les archives de la liste sont disponibles sur < <https://lists.debian.org/debian-live/> >.
- **IRC** : Un certain nombre d'utilisateurs et de développeurs sont

présents dans le canal #debian-live sur irc.debian.org (OFTC). Quand vous posez une question sur IRC, s'il vous plaît soyez patient en attendant une réponse. Si aucune réponse n'est donnée, veuillez envoyer un courriel à la liste de diffusion.

- **BTS** : Le < **Rapporter des bogues** >.

105

Utilisateur

107 Installation

108 3. Installation

109 3.1 Exigences

110 Les exigences pour la création des images des systèmes live sont très faibles :

- 111 • Accès super-utilisateur (root)
- 112 • Une version mise à jour de *live-build*
- 113 • Un shell POSIX, comme *bash* ou *dash*
- 114 • *debootstrap* ou *cdebootstrap*
- 115 • Linux 2.6.x ou supérieur.

116 Notez que l'utilisation de Debian ou d'une distribution dérivée de Debian n'est pas nécessaire - *live-build* fonctionne sur presque toutes les distributions remplissant les exigences ci-dessus.

117 3.2 Installation de live-build

118 Vous pouvez installer *live-build* d'un certain nombre de façons différentes :

- 119 • À partir du dépôt Debian
- 120 • À partir du code source
- 121 • À partir des instantanés

122 Si vous utilisez Debian, la méthode recommandée consiste à installer *live-build* à partir du dépôt Debian.

123 3.2.1 À partir du dépôt Debian

124 Il suffit d'installer *live-build* comme n'importe quel autre paquet :

```
# apt-get install live-build
```

125

3.2.2 À partir du code source

126

127 *live-build* est développé en utilisant le système de contrôle de version Git. Dans les systèmes basés sur Debian, il est fourni par le paquet *git*. Pour examiner le dernier code, exécutez :

```
$ git clone git://live-systems.org/git/live-build.git
```

128

129 Vous pouvez compiler et installer votre propre paquet Debian en exécutant :

```
$ cd live-build
$ dpkg-buildpackage -b -uc -us
$ cd ..
```

130

131 Maintenant, installez les fichiers récemment construits qui vous intéressent, par exemple

```
# dpkg -i live-build_3.0-1_all.deb
```

132

Vous pouvez également installer *live-build* directement sur votre système en exécutant :

```
# make install
```

et le désinstaller avec :

```
# make uninstall
```

3.2.3 À partir des instantanés

Si vous ne souhaitez pas créer ou installer *live-build* à partir des sources, vous pouvez utiliser des instantanés. Ils sont construits automatiquement à partir de la dernière version du dépôt Git et sont disponibles sur < <http://live-systems.org/debian/> >.

3.3 Installation de live-boot et live-config

Remarque : Vous n'avez pas besoin d'installer *live-boot* ou *live-config* sur votre système afin de créer des systèmes live. Cependant, cela ne fera aucun mal et est utile à des fins de référence. Si vous voulez seulement la documentation, vous pouvez maintenant installer les paquets *live-boot-doc* et *live-config-doc* séparément.

3.3.1 À partir du dépôt Debian

live-boot et *live-config* sont tous les deux disponibles dans le dépôt Debian comme expliqué dans < [Installation de live-build](#) >.

3.3.2 À partir du code source

Pour utiliser les dernières sources du git, vous pouvez suivre la procédure ci-dessous. Veuillez vous assurer que vous êtes familiarisé avec les termes mentionnés dans < [Termes](#) >.

- Examiner les sources de *live-boot* et *live-config*

```
$ git clone git ://live-systems.org/git/live-boot.git
$ git clone git ://live-systems.org/git/live-config.git
```

Consultez les pages de manuel de *live-boot* et *live-config* pour plus de détails sur la personnalisation si la raison pour laquelle vous créez vos paquets à partir des sources.

- Créer les fichiers .deb de *live-boot* et *live-config*

Vous devez créer sur votre distribution cible ou dans un chroot contenant votre plateforme cible : cela signifie que si votre cible est **jessie** alors vous devez créer sur **jessie** .

Utilisez un système de construction automatique personnel tel que *pbuilder* ou *sbuid* si vous avez besoin de créer *live-boot* pour une distribution cible qui diffère de votre système de construction. Par exemple, pour les images live de **jessie** , créez *live-boot* dans un chroot **jessie** . Si votre distribution cible correspond à votre distribution vous pouvez créer directement sur le système de construction en utilisant *dpkg-buildpackage* (fourni par le paquet *dpkg-dev*) :

```
$ cd live-boot
$ dpkg-buildpackage -b -uc -us
$ cd ../live-config
$ dpkg-buildpackage -b -uc -us
```

- Utiliser les fichiers .deb générés nécessaires.

Comme *live-boot* et *live-config* sont installés par le système de construction *live-build*, l'installation de ces paquets dans le système hôte ne suffit pas : vous devez traiter les fichiers .deb générés comme d'autres paquets personnalisés. Comme votre objectif pour la construction à partir du code source est de tester nouvelles choses à court terme avant leur publication officielle, suivez < [Installation de paquets modifiés ou de tiers](#) > pour inclure temporairement les fichiers pertinents dans votre configuration. En particulier, remarquez que les deux paquets sont divisés en une partie générique, une partie de documentation et un ou plusieurs back-ends. Incluez la partie générique, un seul back-end et éventuellement la documentation. En supposant que vous construisiez une image live dans le répertoire courant et ayez généré tous les fichiers .deb pour une version unique des deux paquets dans le répertoire ci-dessus, ces commandes bash copieraient tous les paquets appropriés, y compris les back-ends par défaut :

```
$ cp ../live-boot{_,-initramfs-tools,-doc}*.deb config/packages.chroot/  
$ cp ../live-config{_,-sysvinit,-doc}*.deb config/packages.chroot/
```

3.3.3 À partir des instantanés

Vous pouvez laisser *live-build* utiliser automatiquement les derniers instantanés de *live-boot* et *live-config* en configurant le dépôt tiers live-systems.org dans votre répertoire de configuration de *live-build*.

157 Les bases

158 4. Les bases

159 Ce chapitre contient un bref aperçu du processus de construction et des instructions pour utiliser les trois types d'images les plus couramment utilisées. Le type d'image le plus polyvalent, *iso-hybrid*, peut être utilisé sur une machine virtuelle, un support optique ou un périphérique USB de stockage portable. Dans certains cas particuliers, comme expliqué plus loin, le type *hdd* peut être plus approprié. Le chapitre contient des instructions détaillées pour la construction et l'utilisation d'une image *netboot*, qui est un peu plus compliquée en raison de la configuration requise sur le serveur. C'est un sujet un peu avancé pour tous ceux qui ne connaissent pas déjà le démarrage sur le réseau, mais est inclus ici car une fois la configuration terminée, c'est un moyen très pratique pour tester et déployer des images pour le démarrage sur le réseau local sans le tracas des supports d'images.

160 La section se termine par une brève introduction à *webbooting* qui est, peut-être, la meilleure façon d'utiliser des images différentes à des fins différentes, changeant de l'un à l'autre en fonction des besoins en utilisant l'Internet comme un moyen.

161 Tout au long du chapitre, nous ferons souvent référence à la valeur par défaut des noms des fichiers produits par *live-build*. Si vous < **téléchargez une image précompilée** >, les noms des fichiers peuvent varier.

162 4.1 Qu'est-ce qu'un système live ?

163 Un système live signifie généralement un système d'exploitation démarré sur un ordinateur à partir d'un support amovible, tel qu'un

CD-ROM, une clé USB ou sur un réseau, prêt à l'emploi sans aucune installation sur le disque habituel, avec auto-configuration fait lors de l'exécution (voir < **Termes** >).

Avec les systèmes live, c'est un système d'exploitation, construit pour une des architectures prises en charge (actuellement amd64 et i386). Il est fait à partir des éléments suivants :

- **Image du noyau Linux** , d'habitude appelé *vmlinuz** 165
- **Image du RAM-disque initiale (*initrd*)** : Un disque virtuel RAM configuré pour le démarrage de Linux, contenant possiblement des modules nécessaires pour monter l'image du système et certains scripts pour le faire. 166
- **Image du système** : L'image du système de fichiers du système d'exploitation. Habituellement, un système de fichiers SquashFS comprimé est utilisé pour réduire au minimum la taille de l'image live. Notez qu'il est en lecture seulement. Ainsi, lors du démarrage le système live va utiliser un disque RAM et un mécanisme "union" pour permettre l'écriture de fichiers dans le système en marche. Cependant, toutes les modifications seront perdues lors de l'arrêt à moins que l'option « persistance » soit utilisée (voir < **Persistance** >). 167
- **Chargeur d'amorçage** : Un petit morceau de code conçu pour démarrer à partir du support choisi, il peut présenter un menu rapide ou permettre la sélection des options/configurations. Il charge le noyau Linux et son *initrd* pour fonctionner avec un système de fichiers associé. Différentes solutions peuvent être utilisées, selon le support de destination et le format du système de fichiers contenant les composants mentionnés précédemment : *isolinux* pour démarrer à partir d'un CD ou DVD au format ISO9660, *syslinux* pour démarrer un disque dur ou une clé USB à partir d'une partition VFAT, *extlinux* pour partitions *ext2/3/4* et *btrfs*, *pxelinux* pour netboot PXE, GRUB 168

pour partitions ext2/3/4, etc.

169 Vous pouvez utiliser *live-build* pour construire l'image du système à partir de vos spécifications, configurer un noyau Linux, son initrd, et un chargeur d'amorçage pour les exécuter, tout dans un format en fonction du support (image ISO9660, image disque, etc.).

170 4.2 Téléchargement des images précompilées

171 Bien que l'objectif de ce manuel soit le développement et la création de vos propres images live, vous pouvez simplement vouloir tester une de nos images précompilées comme une introduction à leur utilisation ou à la construction de vos propres images. Ces images sont construites à l'aide de notre < dépôt git live-images > et les versions officielles stables sont publiées sur < <https://www.debian.org/CD/live/> >. En outre, les versions plus anciennes et les futures, et des images non officielles contenant micrologiciels et pilotes non libres sont disponibles sur < <http://live-systems.org/cdimage/release/> >.

172 4.3 Utiliser le constructeur web d'images live

173 En tant que service à la communauté, nous gérons un service de construction d'images web sur < <http://live-build.debian.net/> >. Ce site est maintenu sur la base du meilleur effort. Autrement dit, même si nous nous efforçons de le maintenir à jour et opérationnel à tout moment, et de publier des avis d'importantes interruptions du service, nous ne pouvons pas garantir une disponibilité de 100% ou des constructions d'images rapides, et le service peut parfois avoir des problèmes dont la résolution prend un certain temps. Si vous avez des problèmes ou des questions au sujet du service, veuillez < nous contacter > en donnant le lien vers votre construction.

4.3.1 Utilisation du constructeur web et avertissements

174

L'interface web ne permet actuellement pas d'empêcher l'utilisation de combinaisons d'options invalides, en particulier quand le changement d'une option (c'est-à-dire en utilisant *live-build* directement) modifie les valeurs des autres options énumérées dans le formulaire web, le constructeur web ne modifie pas ces valeurs par défaut. Plus particulièrement, si vous changez la valeur `--architectures` qui est par défaut `i386` pour amd64, vous devez modifier l'option correspondante `--linux-flavours` de la valeur par défaut `486` pour amd64. Voir la page de manuel `lb_config` pour la version de *live-build* installée sur le constructeur web pour plus de détails. Le numéro de version de *live-build* est indiqué au bas de la page web.

175

L'estimation du temps donné par le constructeur web est une estimation brute et peut ne pas refléter la durée effective de votre construction. Cette estimation n'est pas actualisée une fois qu'elle est affichée. Soyez patient. Ne rechargez pas la page de la construction, car cela peut soumettre une nouvelle construction avec les mêmes paramètres. Vous devez < nous contacter > si vous ne recevez pas la notification de votre construction mais seulement une fois que vous êtes sûr que vous avez attendu assez longtemps et vérifié que la notification par e-mail n'a pas été détectée par votre filtre anti-spam.

176

Le constructeur web est limité dans les types d'images qu'il peut construire. Cela permet de garder les choses simples et efficaces à utiliser et à maintenir. Si vous souhaitez effectuer des personnalisations qui ne sont pas prévues par l'interface web, le reste de ce manuel explique comment construire vos propres images en utilisant *live-build*.

177

4.4 Premières étapes : la construction d'une image ISO hybride

Quel que soit le type d'image, vous devrez effectuer les mêmes étapes de base pour créer une image à chaque fois. Comme premier exemple, créez un répertoire de travail, passez dans ce répertoire et exécutez la séquence suivante de commandes *live-build* pour créer une image ISO hybride de base contenant tout le système Debian par défaut sans X.org. Elle est appropriée pour être gravée sur CD ou DVD, et peut également être copiée sur une clé USB.

Le nom du répertoire de travail dépend totalement de vous, mais si vous jetez un œil aux exemples utilisés dans *live-manual*, c'est une bonne idée d'utiliser un nom qui vous aide à identifier l'image avec laquelle vous travaillez dans chaque répertoire, surtout si vous travaillez ou expérimentez avec différents types d'images. Dans ce cas, vous allez construire un système par défaut, nous allons donc l'appeler, par exemple, *live-default*.

```
$ mkdir live-default && cd live-default
```

Ensuite, exécutez la commande `lb config`. Cela va créer une hiérarchie "config/" dans le répertoire courant pour être utilisée par d'autres commandes :

```
$ lb config
```

Aucun paramètre n'est passé à ces commandes, donc les défauts seront utilisés pour l'ensemble de ses diverses options. Consultez [« La commande lb config »](#) pour plus de détails.

Maintenant que la hiérarchie "config/" existe, créez l'image avec la commande `lb build` :

```
# lb build
```

Ce processus peut prendre un certain temps, en fonction de la vitesse de votre ordinateur et de votre connexion réseau. Une fois le processus terminé, il devrait y avoir un fichier image `live-image-i386.hybrid.iso` prêt à l'emploi, dans le répertoire courant.

Remarque : Si vous construisez sur un système amd64 le nom de l'image résultante sera `live-image-amd64.hybrid.iso`. Gardez à l'esprit cette convention de nommage tout au long du manuel.

4.5 Utilisation d'une image ISO hybride live

Après la construction ou le téléchargement d'une image ISO hybride, qui peut être obtenue sur <https://www.debian.org/CD/live/>, l'étape suivante est d'habitude de préparer votre support pour le démarrage, soit sur CD-R(W) ou DVD-R(W), des supports optiques ou une clé USB.

4.5.1 Graver une image ISO sur un support physique

Graver une image ISO est facile. Il suffit d'installer `{xorriso}` et de l'utiliser à partir de la ligne de commande pour graver l'image. Par exemple :

```
# apt-get install xorriso
```

```
$ xorriso -as cdrecord -v dev=/dev/sr0 blank=as_needed live-image-i386.↵
hybrid.iso
```

4.5.2 Copie d'une image ISO hybride sur une clé USB

Les images ISO préparées avec `xorriso` peuvent être simplement copiées sur une clé USB avec `cp` ou un logiciel équivalent. Branchez une clé USB avec une capacité suffisamment grande pour votre fichier image et déterminez quel périphérique elle est, que nous appelons ci-dessous `${USBSTICK}`. C'est le fichier de périphérique de votre clé, tel que `/dev/sdb`, pas une partition, telle que `/dev/sdb1` ! Vous pouvez trouver le nom du périphérique en regardant la sortie de `dmesg` après avoir branché le périphérique, ou mieux encore, `ls -l /dev/disk/by-id`.

Une fois que vous êtes sûr d'avoir le nom correct de l'appareil, utilisez la commande `cp` pour copier l'image sur la clé. **Ceci écrasera tout fichier déjà existant sur votre clé !**

```
$ cp live-image-i386.hybrid.iso ${USBSTICK}
$ sync
```

Remarque : La commande `sync` est utilisée pour s'assurer que toutes les données qui sont stockées dans la mémoire par le noyau lors de la copie de l'image, sont écrites sur la clé USB.

4.5.3 Utilisation de l'espace disponible sur une clé USB

Après avoir copié `live-image-i386.hybrid.iso` sur une clé USB, la première partition sera utilisée par le système live. Pour utiliser l'espace libre restant, utilisez un outil de partitionnement tel que

`gparted` ou `parted` afin de créer une nouvelle partition sur la clé.

```
# gparted ${USBSTICK}
```

Quand la partition est créée, vous devez y créer un système de fichiers où `${PARTITION}` est le nom de la partition, comme `/dev/sdb2`. Un choix possible serait `ext4`.

```
# mkfs.ext4 ${PARTITION}
```

Remarque : Si vous voulez utiliser l'espace supplémentaire avec Windows, ce système d'exploitation ne peut accéder normalement à aucune partition à part la première. Certaines solutions à ce problème ont été discutées sur notre < [liste de diffusion](#) >, mais il semble qu'il n'y a pas de réponse facile.

Rappelez-vous : Chaque fois que vous installez une nouvelle `live-image-i386.hybrid.iso` sur la clé, toutes les données sur la clé seront perdues parce que la table de partition est écrasée par le contenu de l'image, vous devez sauvegarder votre partition supplémentaire d'abord pour la restaurer à nouveau après la mise à jour de l'image live.

4.5.4 Démarrer le support live

La première fois que vous démarrez votre support live, qu'il s'agisse de CD, DVD, clé USB, ou du démarrage par PXE, une certaine configuration dans le BIOS de votre ordinateur peut être d'abord nécessaire. Puisque les BIOS varient grandement en fonctionnalités et raccourcis clavier, on ne peut pas aborder le

sujet en profondeur ici. Certains BIOS fournissent une touche pour ouvrir un menu d'amorçage au démarrage, qui est le moyen le plus facile si elle est disponible sur votre système. Sinon, vous avez besoin d'entrer dans le menu de configuration du BIOS et modifier l'ordre de démarrage pour placer le dispositif de démarrage pour le système live devant votre périphérique de démarrage normal.

Une fois que vous avez démarré le support, un menu de démarrage vous est présenté. Si vous appuyez simplement sur entrée ici, le système va démarrer en utilisant l'entrée par défaut, `Live`. Pour plus d'informations sur les options de démarrage, consultez l'entrée « `Help` » dans le menu et aussi les pages de manuel de *live-boot* et *live-config* dans le système live.

En supposant que vous ayez sélectionné `Live` et démarré une image de bureau live par défaut, après que les messages de démarrage aient défilé, vous devriez être automatiquement connecté au compte `user` et voir un bureau, prêt à l'emploi. Si vous avez démarré une image de la console uniquement, tels que les types standard ou rescue des `< images précompilées >`, vous devriez être automatiquement connecté à la console pour le compte `user` et voir une invite du shell, prête à l'emploi.

4.6 Utiliser une machine virtuelle pour les tests

Exécuter les images live dans une machine virtuelle (VM) peut faire gagner beaucoup de temps. Cela ne vient pas sans avertissements :

- L'exécution d'une VM demande assez de RAM pour le système d'exploitation client et l'hôte et un CPU avec support matériel pour la virtualisation est recommandé.
- Il y a quelques limitations inhérentes à l'exécution sur une VM,

par exemple des performances vidéo médiocres, ou un choix limité de matériel émulé.

- Lors du développement d'un matériel spécifique, il n'existe aucun substitut pour l'exécution que le matériel lui-même.
- Certains ne deviennent visibles que pendant l'exécution dans une VM. En cas de doute, testez votre image directement sur le matériel.

À condition de pouvoir travailler avec ces contraintes, examinez les logiciels VM disponibles et choisissez celui qui convient à vos besoins.

4.6.1 Test d'une image ISO avec QEMU

La VM la plus polyvalente de Debian est QEMU. Si votre processeur dispose d'une gestion matérielle de la virtualisation, vous pouvez utiliser le paquet *qemu-kvm* ; La description du paquet *qemu-kvm* énumère brièvement les exigences.

Tout d'abord, installez *qemu-kvm* si votre processeur le gère. Sinon, installez *qemu*. Dans ce cas, le nom du programme est *qemu* au lieu de *kvm* dans les exemples suivants. Le paquet *qemu-utils* est également valable pour créer des images de disques virtuels avec *qemu-img*.

```
# apt-get install qemu-kvm qemu-utils
```

Démarrer une image ISO est simple :

```
$ kvm -cdrom live-image-i386.hybrid.iso
```

Voir les pages de manuel pour plus de détails.

4.6.2 Test d'une image ISO avec VirtualBox

Afin de tester l'ISO avec *virtualbox* :

```
# apt-get install virtualbox virtualbox-qt virtualbox-dkms
$ virtualbox
```

Créez une nouvelle machine virtuelle, modifiez les paramètres de stockage pour utiliser `live-image-i386.hybrid.iso` comme le périphérique CD/DVD et démarrez la machine.

Remarque : Pour les systèmes live contenant X.org que vous voulez essayer avec *virtualbox*, vous pouvez inclure le paquet des pilotes VirtualBox X.org, *virtualbox-guest-dkms* et *virtualbox-guest-x11*, dans votre configuration de *live-build*. Sinon, la résolution est limitée à 800x600.

```
$ echo "virtualbox-guest-dkms virtualbox-guest-x11" >> config/package-lists<→
/my.list.chroot
```

Pour faire fonctionner le paquet *dmks*, il faut également installer le paquet *linux-headers* pour le noyau utilisé dans l'image. Au lieu de lister manuellement le paquet *linux-headers* correct dans la liste de paquets créée ci-dessus, *live-build* peut faire cela automatiquement.

```
$ lb config --linux-packages "linux-image linux-headers"
```

4.7 Construire et utiliser une image HDD

La construction d'une image HDD est similaire à une ISO hybride à tous les regards, sauf que vous spécifiez `-b hdd` et le nom du fichier résultant est `live-image-i386.img` qui ne peut être gravé sur des supports optiques. Il convient pour le démarrage à partir de clés USB, disques durs USB, et divers autres dispositifs de stockage portables. Normalement, une image ISO hybride peut être utilisée à cette fin, mais si vous avez un BIOS qui ne gère pas correctement les images hybrides, vous devez utiliser une image HDD.

Remarque : si vous avez créé une image ISO hybride avec l'exemple précédent, vous devrez nettoyer votre répertoire de travail avec la commande `lb clean` (voir < [La commande lb clean](#) >) :

```
# lb clean --binary
```

Exécutez la commande `lb config` comme avant, sauf que cette fois en spécifiant le type d'image HDD :

```
$ lb config -b hdd
```

Construisez maintenant l'image avec la commande `lb build`

```
# lb build
```

Quand la création de l'image est finie, un fichier `live-image-i386.img` doit être présent dans le répertoire courant.

L'image binaire générée contient une partition VFAT et le chargeur d'amorçage syslinux, prêts à être écrits directement sur une clé USB. Encore une fois, comme l'utilisation d'une image HDD est juste comme l'utilisation d'une image ISO hybride sur USB, suivez les instructions < **Utiliser une image live ISO hybride** >, en utilisant le nom de fichier `live-image-i386.img` au lieu de `live-image-i386.hybrid.iso`.

De même, pour tester une image HDD avec Qemu, installez `qemu` comme décrit ci-dessus dans < **Test d'une image ISO avec QEMU** >. Ensuite, exécutez `kvm` ou `qemu`, selon la version dont votre système hôte a besoin en précisant `live-image-i386.img` comme le premier disque dur.

```
$ kvm -hda live-image-i386.img
```

4.8 Construction d'une image netboot

La séquence de commandes suivante va créer une image NetBoot de base contenant le système Debian par défaut sans X.org. Elle peut être démarrée sur le réseau.

Remarque : Si vous avez réalisé quelques-uns des exemples précédents, vous aurez besoin de nettoyer votre répertoire de travail avec la commande `lb clean` :

```
# lb clean
```

Dans ce cas spécifique, un `lb clean --binary` ne serait pas suffisant pour nettoyer les étapes nécessaires. La raison est que dans les configurations de netboot, une configuration

`initramfs` différente doit être utilisée, laquelle *live-build* exécute automatiquement lors de la construction des images netboot. Puisque la création d'`initramfs` appartient à l'étape `chroot`, si on change à netboot dans un répertoire existant, il faut reconstruire le `chroot`. Par conséquent, il faut faire un `lb clean`, (qui permettra d'éliminer l'étape `chroot`, aussi).

Exécutez la commande suivante pour configurer votre image pour démarrer sur le réseau :

```
$ lb config -b netboot --net-root-path "/srv/debian-live" --net-root-server↔
"192.168.0.2"
```

Contrairement aux images ISO et HDD, le démarrage sur le réseau ne sert pas l'image du système de fichiers pour le client. Pour cette raison, les fichiers doivent être servis via NFS. Différents systèmes de fichiers réseau peuvent être choisis avec `lb config`. Les options `--net-root-path` et `--net-root-server` indiquent l'emplacement et le serveur, respectivement, du serveur NFS sur lequel l'image du système de fichiers sera située au moment du démarrage. Assurez-vous que ceux-ci sont fixées à des valeurs appropriées pour votre réseau et serveur.

Construisez maintenant l'image avec la commande `lb build`

```
# lb build
```

Dans un démarrage réseau, le client exécute un petit morceau de logiciel qui réside habituellement sur l'EPROM de la carte Ethernet. Ce programme envoie une requête DHCP pour obtenir une adresse IP et les informations sur ce qu'il faut faire ensuite.

Typiquement, la prochaine étape est d'obtenir un chargeur d'amorçage de niveau supérieur via le protocole TFTP. Cela pourrait être pxelinux, GRUB, ou démarrer directement à un système d'exploitation comme Linux.

Par exemple, si vous décompressez l'archive générée `live-image-i386.netboot.tar` dans le répertoire `/srv/debian-live`, vous trouverez l'image du système de fichiers dans `live/filesystem.squashfs` et le noyau, `initrd` et le chargeur d'amorçage `pxelinux` dans `tftpboot/`.

Nous devons maintenant configurer trois services sur le serveur pour activer le démarrage sur le réseau : le serveur DHCP, le serveur TFTP et le serveur NFS.

4.8.1 Serveur DHCP

Nous devons configurer le serveur DHCP de notre réseau pour être sûr de donner une adresse IP au client du système du démarrage sur le réseau, et pour annoncer l'emplacement du chargeur d'amorçage PXE.

Voici un exemple source d'inspiration, écrit pour le serveur ISC DHCP `isc-dhcp-server` dans le fichier de configuration `/etc/dhcp/dhcpd.conf` :

```
# /etc/dhcp/dhcpd.conf - configuration file for isc-dhcp-server

ddns-update-style none;

option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 600;
max-lease-time 7200;
```

```
log-facility local7;

subnet 192.168.0.0 netmask 255.255.255.0 {
    range 192.168.0.1 192.168.0.254;
    filename "pxelinux.0";
    next-server 192.168.0.2;
    option subnet-mask 255.255.255.0;
    option broadcast-address 192.168.0.255;
    option routers 192.168.0.1;
}
```

4.8.2 Serveur TFTP

Cela sert le noyau et le ramdisk initial pour le système pendant l'exécution.

Vous devriez installer le paquet `tftpd-hpa`. Il peut servir tous les fichiers contenus dans un répertoire racine, d'habitude `/srv/tftp`. Pour le laisser servir des fichiers dans `/srv/debian-live/tftpboot`, exécutez comme utilisateur `root` la commande suivante :

```
# dpkg-reconfigure -plow tftpd-hpa
```

et remplissez le nouveau répertoire du serveur tftp

4.8.3 Serveur NFS

Quand l'ordinateur hôte a téléchargé et démarré un noyau Linux et chargé son `initrd`, il va essayer de monter l'image du système de fichiers live via un serveur NFS.

Vous devez installer le paquet `nfs-kernel-server`.

Ensuite, rendez l'image du système de fichiers disponible via NFS

en ajoutant une ligne comme la suivante `/etc/exports` :

```
/srv/debian-live *(ro,async,no_root_squash,no_subtree_check)
```

et indiquez cette exportation au serveur NFS avec la commande suivante :

```
# exportfs -rv
```

La configuration de ces trois services peut être un peu difficile. Vous pourriez avoir besoin de patience pour obtenir que tous fonctionnent ensemble. Pour plus d'informations, consultez le wiki syslinux sur < <http://www.syslinux.org/wiki/index.php/PXELINUX> > ou la section Debian Installer Manual's TFTP Net Booting sur < <http://d-i.alioth.debian.org/manual/fr.i386/ch04s05.html> >. Ils pourraient aider parce que leurs processus sont très semblables.

4.8.4 Guide pratique pour expérimenter avec une image Netboot

La création d'images netboot est facile avec *live-build*, mais les tests des images sur des machines physiques peuvent prendre vraiment beaucoup de temps.

Afin de rendre notre vie plus facile, nous pouvons utiliser la virtualisation.

4.8.5 Qemu

- Installer *qemu*, *bridge-utils*, *sudo*.

Éditer `/etc/qemu-ifup` :

```
#!/bin/sh
sudo -p "Password for $0 : " /sbin/ifconfig $1 172.20.0.1
echo "Executing /etc/qemu-ifup"
echo "Bringing up $1 for bridged mode..."
sudo /sbin/ifconfig $1 0.0.0.0 promisc up
echo "Adding $1 to br0..."
sudo /usr/sbin/brctl addif br0 $1
sleep 2
```

Obtenir, ou construire un `grub-floppy-netboot`.

Lancer `qemu` avec `"-net nic,vlan=0 -net tap,vlan=0,ifname=tun0"`

4.9 Webbooting

Webbooting est une manière très pratique de télécharger et d'amorcer les systèmes live en utilisant l'Internet comme un moyen. Il ya très peu d'exigences pour webbooting. D'une part, vous avez besoin d'un support avec un chargeur d'amorçage, un disque ram initial et un noyau. D'autre part, un serveur web pour stocker les fichiers squashfs qui contiennent le système de fichiers.

4.9.1 Obtenir les fichiers webboot

Comme d'habitude, vous pouvez construire les images vous-même ou utiliser les fichiers précompilés, qui sont disponibles sur le site du projet sur < <http://live-systems.org/> >. L'utilisation des images précompilées serait pratique pour faire l'essai initial jusqu'à ce que l'on peut affiner leurs propres besoins. Si vous avez construit une image live, vous trouverez les fichiers nécessaires pour webbooting dans le répertoire de construction

sous `binary/live/`. Les fichiers sont appelés `vmlinuz`, `initrd.img` et `filesystem.squashfs`.

Il est également possible d'extraire les fichiers d'une image iso déjà existant. Pour ce faire, on doit monter l'image comme suit :

```
# mount -o loop image.iso /mnt
```

Les fichiers se trouvent sous le répertoire `live/`. Dans ce cas précis, il serait `/mnt/live/`. Cette méthode présente l'inconvénient que vous avez besoin d'être root pour pouvoir monter l'image. Cependant, il présente l'avantage qu'elle est facilement scriptable et ainsi, facilement automatisée.

Mais sans aucun doute, la meilleure façon d'extraire les fichiers d'une image iso et les télécharger sur le serveur web au même temps, est d'utiliser le midnight commander ou *mc*. Si vous avez le paquet *genisoimage* installé, le gestionnaire de fichiers à deux panneaux vous permet de voir le contenu d'un fichier iso dans un panneau et de télécharger les fichiers via FTP dans l'autre panneau. Même si cette méthode nécessite un travail manuel, elle ne nécessite pas les privilèges root.

4.9.2 Démarrer images webboot

Tandis que certains utilisateurs vont utiliser la virtualisation pour tester le webbooting, nous utilisons du matériel réel ici pour correspondre au possible cas d'utilisation suivant qui seulement devrait être considéré comme un exemple.

Afin de démarrer une image webboot il suffit d'avoir les éléments mentionnés ci-dessus, c'est-à-dire, `vmlinuz` et `initrd.img` sur une clé usb dans un répertoire nommé `live/` et installer `syslinux`

comme chargeur de démarrage. Ensuite, démarrer à partir de la clé usb et taper `fetch=URL/CHEMIN/DU/FICHER` aux options de démarrage. *live-boot* va télécharger le fichier `squashfs` et le stocker dans la ram. De cette façon, il est possible d'utiliser le système de fichiers compressé téléchargé comme un système live normal. Par exemple :

```
append boot=live components fetch=http://192.168.2.50/images/webboot/↵
filesystem.squashfs
```

Cas d'utilisation : Vous avez un serveur web dans lequel vous avez stocké deux fichiers `squashfs`, un qui contient un bureau complet, comme par exemple `gnome`, et un d'un système de sauvetage. Si vous avez besoin d'un environnement graphique pour une machine, vous pouvez brancher votre clé usb et télécharger l'image qui contient `gnome`. Si vous avez besoin des outils de sauvetage inclus dans le deuxième type d'image, peut-être pour une autre machine, vous pouvez télécharger celle du système de sauvetage.

Aperçu des outils

5. Aperçu des outils

Ce chapitre fournit un aperçu des trois principaux outils utilisés dans la construction des systèmes live : *live-build*, *live-boot* et *live-config*.

5.1 Le paquet live-build

live-build est une collection de scripts pour construire des systèmes live. Ces scripts sont aussi appelés “commandes”.

L'idée derrière *live-build* est de constituer un cadre qui utilise un répertoire de configuration pour automatiser et personnaliser complètement tous les aspects de la construction d'une image Live.

Plusieurs concepts sont similaires à ceux utilisés pour construire des paquets Debian avec *debhelper* :

- Les scripts ont un emplacement central pour la configuration de leur fonctionnement. Avec *debhelper*, c'est le sous-répertoire `debian/` d'un arbre de paquets. Par exemple, `dh_install` cherchera, entre autres, un fichier appelé `debian/install` pour déterminer quels fichiers doivent exister dans un paquet binaire particulier. De la même manière, *live-build* enregistre sa configuration entièrement dans un sous-répertoire `config/`.
- Les scripts sont indépendants, c'est-à-dire qu'il est toujours sûr d'exécuter chaque commande.

Contrairement à *debhelper*, *live-build* contient des outils pour générer une arborescence de configuration. Cela pourrait être considéré comme similaire à des outils tels que *dh-make*. Pour plus d'informations sur ces outils, continuer la lecture, parce

que le reste de cette section est sur les quatre commandes les plus importantes. Notez que la commande `lb` est une fonction générique pour les commandes *live-build*.

- **lb config** : Responsable de l'initialisation d'un répertoire de configuration pour un système Live. Voir < **La commande lb config** > pour plus d'informations.
- **lb build** : Responsable du démarrage d'un système de construction Live. Voir < **La commande lb build** > pour plus d'informations.
- **lb clean** : Responsable de la suppression des parties d'un système de construction Live. Voir < **La commande lb clean** > pour plus d'informations.

5.1.1 La commande lb config

Comme indiqué dans < **live-build** >, les scripts qui composent *live-build* lisent leur configuration avec la commande source à partir d'un seul répertoire nommé `config/`. Comme la construction de ce répertoire à la main serait fastidieuse et source d'erreurs, la commande `lb config` peut être utilisée pour créer une arborescence de configuration.

Exécuter la commande `lb config` sans aucun argument crée le sous-répertoire `config/` qui est peuplée avec certains paramètres dans fichiers de configuration, et deux sous-répertoires `auto/` et `local/` avec une arborescence de fichiers.

```
$ lb config
[2014-04-25 17 :14 :34] lb config
P : Updating config tree for a debian/wheezy/i386 system
```

L'utilisation de `lb config` sans aucun argument serait appropriée

pour les utilisateurs qui ont besoin d'une image de base, ou qui ont l'intention de fournir plus tard une configuration plus complète via `auto/config` (voir < [Gestion d'une configuration](#) > pour plus de détails).

Normalement, vous voulez indiquer certaines options. Par exemple, pour spécifier le gestionnaire de paquets à utiliser lors de la construction de l'image :

```
$ lb config --apt aptitude
```

Il est possible d'indiquer plusieurs options, telles que :

```
$ lb config --binary-images netboot --bootappend-live "boot=live components↵
hostname=live-host username=live-user" ...
```

Une liste complète des options est disponible dans la page de manuel de `lb_config`.

5.1.2 La commande `lb build`

La commande `lb build` lit dans votre configuration à partir du répertoire `config/`. Elle exécute alors les commandes de niveau inférieur nécessaires à la construction de votre système Live.

5.1.3 La commande `lb clean`

Le rôle de la commande `lb clean` est d'enlever les différentes parties d'une construction afin que autres compilations ultérieures puissent commencer à partir d'un état propre. Par défaut, les étapes `chroot`, `binary` et `source` sont nettoyées, mais

le cache est laissé intact. En outre, les étapes peuvent être nettoyées individuellement. Par exemple, si vous avez effectué des changements qui affectent uniquement la phase binaire, utilisez `lb clean --binary` avant de construire un nouveau binaire. Si vos modifications invalident le bootstrap et/ou les caches de paquets, par exemple, modifications aux options `--mode`, `--architecture` ou `--bootstrap`, vous devez utiliser `lb clean --purge`. Voir la page de manuel de `lb_clean` pour une liste complète des options.

5.2 Le paquet `live-boot`

live-boot est une collection de scripts fournissant des hooks pour *initramfs-tools*. Il est utilisé pour générer un *initramfs* capable de démarrer des systèmes live, comme ceux créés par *live-build*. Cela inclut ISOs, netboot tarballs, et les images pour clés USB.

Au démarrage, il va chercher un support en lecture seule qui contient un répertoire `/live/` où un système de fichiers racine (souvent une image du système de fichiers compressée comme `squashfs`) est stocké. S'il le trouve, il va créer un environnement accessible en écriture, en utilisant `aufs`, afin que les systèmes similaires à Debian puissent démarrer à partir de cet environnement.

Plus d'information sur initial ramfs dans Debian peut être trouvée dans le Debian Linux Kernel Handbook sur < <http://kernel-handbook.alioth.debian.org/> > dans le chapitre sur *initramfs*.

5.3 Le paquet `live-config`

live-config se compose des scripts qui s'exécutent au démarrage après *live-boot* pour configurer le système live automatiquement. Il gère les tâches telles que l'établissement du nom d'hôte,

des paramètres régionaux et du fuseau horaire, la création de l'utilisateur live, l'inhibition des cron jobs et l'autologin de l'utilisateur live.

Gestion d'une configuration

6. Gestion d'une configuration

Ce chapitre explique comment gérer une configuration d'un système live à partir d'une création initiale, à travers des révisions successives et des versions successives du logiciel *live-build* et de l'image live elle-même.

6.1 Gérer les modifications de la configuration

Les configurations live sont rarement parfaites du premier coup. Il peut être bon de passer des options `lb config` à partir de la ligne de commande pour effectuer une construction unique, mais il est plus courant de réviser ces options et de construire à nouveau jusqu'à ce que vous soyez satisfait. Afin de prendre en charge ces changements, vous aurez besoin des scripts automatiques qui assurent le maintien de votre configuration dans un état cohérent.

6.1.1 Pourquoi utiliser des scripts auto ? Que font-ils ?

La commande `lb config` enregistre les options que vous lui passez avec les fichiers dans `config/*` avec beaucoup d'autres options aux valeurs par défaut. Si vous exécutez `lb config` à nouveau, il ne réinitialisera pas l'option qui a été mise par défaut en fonction de vos options initiales. Ainsi, par exemple, si vous exécutez `lb config` à nouveau avec une nouvelle valeur pour `--binary-images`, toutes les options qui ont été mises à leur valeur par défaut pour l'ancienne type d'image ne peuvent plus fonctionner avec la nouvelle option. Ces fichiers ne sont pas destinés à être lus ou modifiés. Ils enregistrent des valeurs pour plus d'une centaine d'options, donc personne (y-compris

vous) ne pourra voir dans ces options lesquelles vous avez réellement indiquées. Finalement, si vous lancez `lb config`, puis mettez *live-build* à niveau et que celui-ci renomme une option, `config/*` contiendra toujours des variables nommées en fonction de l'ancienne option et qui ne seront plus valides.

Pour toutes ces raisons, les scripts `auto/*` vous rendront la vie plus facile. Ils sont de simples emballages pour les commandes `lb config`, `lb build` et `lb clean` qui sont conçus pour vous aider à gérer votre configuration. Le script `auto/config` enregistre votre commande `lb config` avec toutes les options désirées, le script `auto/clean` supprime les fichiers contenant les valeurs des variables de configuration et le script `auto/build` crée un `build.log` de chaque construction. Et chaque fois que vous lancez la commande `lb` correspondante, ces fichiers sont exécutés automatiquement. En utilisant ces scripts, votre configuration est plus facile à lire et a une cohérence interne d'une révision à l'autre. En outre, il sera plus facile pour vous d'identifier et corriger les options qui doivent changer lorsque vous mettez à niveau *live-build* après avoir lu la documentation mise à jour.

6.1.2 Utiliser les scripts auto d'exemple

Pour votre commodité, *live-build* est fourni avec des scripts shell d'exemple, pour les copier et les modifier. Lancez une nouvelle configuration par défaut, puis copiez les exemples :

```
$ mkdir mylive && cd mylive && lb config
$ mkdir auto
$ cp /usr/share/doc/live-build/examples/auto/* auto/
```

Modifiez `auto/config` en ajoutant des options comme bon vous semble. Par exemple :

```
#!/bin/sh
lb config noauto \
  --architectures i386 \
  --linux-flavours 686-pae \
  --binary-images hdd \
  --mirror-bootstrap http://ftp.ch.debian.org/debian/ \
  --mirror-binary http://ftp.ch.debian.org/debian/ \
  "${@}"
```

Maintenant, chaque fois que vous utilisez `lb config`, `auto/config` réinitialisera la configuration basée sur ces options. Lorsque vous souhaitez effectuer des modifications, modifiez les options dans ce fichier au lieu de les passer à `lb config`. Lorsque vous utilisez `lb clean`, `auto/clean` va nettoyer les fichiers ainsi que tous les autres produits de construction. Et enfin, lorsque vous utilisez `lb build`, un journal de la construction est écrit par `auto/build` dans `build.log`.

Remarque : Un paramètre spécial `noauto` est utilisé ici pour éliminer un autre appel à `auto/config`, évitant ainsi une récursion infinie. Assurez-vous que vous ne l'avez pas accidentellement supprimé en modifiant le fichier. Aussi, prenez soin de vous assurer quand vous divisez la commande `lb config` sur plusieurs lignes pour une meilleure lisibilité, comme le montre l'exemple ci-dessus, que vous n'oubliez pas la barre oblique inverse (de sorte que chaque ligne continue à la suivante.

6.2 Cloner une configuration publiée via Git

Utilisez l'option `lb config --config` pour cloner un dépôt Git qui contient une configuration d'un système live. Si vous souhaitez baser votre configuration sur une autre maintenue par le Live Systems Project, allez voir le dépôt sur < <http://live-systems.org/gitweb> >

avec le nom `live-images` sous le titre Packages. Ce dépôt contient les configurations pour les < **images précompilées** >

Par exemple, pour construire une image de récupération, utilisez le dépôt `live-images` comme suit :

```
$ mkdir live-images && cd live-images
$ lb config --config git ://live-systems.org/git/live-images.git
$ cd images/rescue
```

Modifiez `auto/config` et toutes les autres choses dont vous avez besoin dans l'arbre `config` en fonction de vos besoins. Par exemple, les images précompilées non officielles qui contiennent paquets non-free sont faites en ajoutant simplement `--archive-areas "main contrib non-free"`.

Vous pouvez éventuellement définir un raccourci dans votre configuration Git en ajoutant la ligne suivante à votre `$(HOME)/.gitconfig` :

```
[url "git ://live-systems.org/git/"]
  insteadOf = lso :
```

Cela vous permet d'utiliser `lso` : quand vous voulez indiquer l'adresse d'un dépôt `live-systems.org`. Si vous supprimez le suffixe optionnel `.git`, commencer une nouvelle image en utilisant cette configuration est aussi simple que :

```
$ lb config --config lso live-images
```

Le clonage de la totalité du dépôt `live-images` copie les

configurations utilisées pour plusieurs images. Si vous voulez construire une image différente lorsque vous avez terminé avec la première, changez de répertoire et, éventuellement, faites les modifications dont vous avez besoin.

354

Dans tous les cas, n'oubliez pas qu'il faut à chaque fois construire l'image en tant que superutilisateur : `lb build`

Personnalisation des contenus

7. Vue d'ensemble de la personnalisation

Ce chapitre donne un aperçu des diverses façons dont vous pouvez personnaliser un système live.

7.1 Configuration pendant la construction vs. l'amorçage

Les options de configuration d'un système live sont divisées en options au moment de la construction, ces options sont appliquées pendant la création et des options au moment du démarrage, qui sont appliquées pendant le démarrage. Les options au moment du démarrage sont divisées en celles qui surviennent au début, appliquées par le paquet *live-boot*, et celles qui arrivent plus tard, appliquées par *live-config*. Toute option d'amorçage peut être modifiée par l'utilisateur en l'indiquant à l'invite de démarrage. L'image peut également être construite avec les paramètres de démarrage par défaut et alors les utilisateurs peuvent normalement démarrer directement le système live sans indiquer aucune option lorsque toutes les valeurs par défaut sont appropriées. En particulier, l'argument `lb --bootappend-live` se compose de toutes les options de ligne de commande du noyau par défaut pour le système live, comme la persistance, les claviers, ou le fuseau horaire. Voir < [Personnalisation des paramètres régionaux et la langue](#) >, par exemple.

Les options de configuration pendant la construction sont décrites dans la page de manuel pour `lb config`. Les options de configuration pendant l'amorçage sont décrites dans les pages de manuel pour *live-boot* et *live-config*. Bien que les paquets *live-boot* et *live-config* soient installés dans le système live que vous construisez, il est recommandé que vous les installiez

également sur votre système de construction pour vous y référer facilement lorsque vous travaillez sur votre configuration. Cela peut être fait sans danger car aucun des scripts contenus ne sont exécutés à moins que le système soit configuré comme un système live.

7.2 Étapes de la construction

Le processus de construction est divisé en étapes, avec des personnalisations différentes appliquées dans l'ordre dans chaque étape. La première étape à exécuter est l'étape **bootstrap**. C'est la phase initiale de peuplement du répertoire chroot avec des paquets pour faire un système Debian de base. Elle est suivie par l'étape **chroot**, qui complète la construction du répertoire chroot, le peuplant de tous les paquets listés dans la configuration, ainsi que tout autre matériel. La plupart de la personnalisation du contenu se produit à ce stade. La dernière étape de la préparation de l'image live est l'étape **binary**, qui construit une image amorçable, en utilisant le contenu du répertoire chroot pour construire le système de fichiers racine pour le système Live. Il comprend l'installateur et tout autre matériel supplémentaire sur le support cible en dehors du système de fichiers du système live. Quand l'image live est construite, s'il est activé, le tarball des sources est construit dans l'étape **source**.

À chacune de ces étapes, les commandes sont appliquées dans un ordre particulier. Les commandes sont ordonnées de manière à assurer que les personnalisations puissent être superposées de manière raisonnable. Par exemple, dans l'étape **chroot**, les préconfigurations sont appliquées avant que tous les paquets ne soient installés, les paquets sont installés avant que tous les fichiers locaux inclus ne soient copiés et les hooks sont exécutés plus tard, quand tous les matériaux sont en place.

364

7.3 Supplément lb config avec des fichiers

365

Bien que lb config crée une arborescence de configuration dans le répertoire config/, pour accomplir vos objectifs, vous pourriez avoir besoin de fournir des fichiers supplémentaires dans les sous-répertoires de config/. Selon l'endroit où les fichiers sont stockés dans la configuration, ils peuvent être copiés dans le système de fichiers du système live ou dans le système de fichiers de l'image binaire, ou peuvent fournir pendant la construction des configurations du système qui seraient lourdes à passer comme options de la ligne de commande. Vous pouvez inclure des choses telles que des listes personnalisées de paquets, art personnalisé, ou des scripts hook à exécuter, soit pendant la construction soit au démarrage, ce qui augmente la flexibilité déjà considérable de debian-live avec le code de votre choix.

366

7.4 Tâches de personnalisation

367

Les chapitres suivants sont organisés par les types des tâches de personnalisation que les utilisateurs effectuent généralement : < **Personnalisation de l'installation de paquets** >, < **Personnalisation des contenus** > et < **Personnalisation des paramètres régionaux et la langue** > couvrent quelques choses que vous pourriez vouloir faire.

Personnalisation de l'installation de paquets

8. Personnalisation de l'installation de paquets

La personnalisation la plus fondamentale d'un système live est sans doute la sélection des paquets à inclure dans l'image. Ce chapitre vous guide tout au long des différentes options de construction pour personnaliser l'installation des paquets avec *live-build*. Le plus large choix influençant les paquets disponibles pour l'installation dans l'image sont la distribution et les zones d'archive. Afin de vous assurer des vitesses de téléchargement décentes, vous devez choisir un miroir de distribution proche. Vous pouvez également ajouter vos propres dépôts pour les rétroportages, paquets expérimentaux ou personnalisés, ou inclure des paquets directement comme fichiers. Vous pouvez définir des listes de paquets, incluant des métapaquets qui installent en même temps de nombreux paquets liés, tels que les paquets pour ordinateurs de bureau ou une langue particulière. Enfin, un certain nombre d'options donne un certain contrôle sur *apt*, ou si vous préférez, *aptitude*, pendant la construction quand les paquets sont installés. Vous pouvez trouver cela très pratique si vous utilisez un proxy, si vous voulez désactiver l'installation des paquets recommandés pour économiser l'espace, ou avez besoin de contrôler quelles versions des paquets sont installées via APT pinning, pour ne nommer que quelques possibilités.

8.1 Sources des paquets

8.1.1 Distribution, zones d'archive et mode

La distribution que vous choisissez a le plus large impact sur les paquets qui sont disponibles pour l'inclusion dans votre image live. Indiquez le nom de code, qui est par défaut **jessie** pour la version

de *live-build* dans **jessie**. Toute distribution actuelle dans l'archive peut être indiquée par son nom de code ici. (Voir < **Termes** > pour plus de détails.) L'option `--distribution` influence non seulement la source des paquets dans l'archive, mais indique également à *live-build* comment construire chaque distribution prise en charge. Par exemple, pour construire sur **unstable**, **sid**, précisez :

```
$ lb config --distribution sid
```

Dans l'archive de distribution, les zones d'archive (« archive areas ») sont les principales divisions de l'archive. Dans Debian, ce sont *main*, *contrib* et *non-free*. Seule *main* contient des logiciels qui font partie de la distribution Debian, c'est donc la valeur par défaut. Une ou plusieurs valeurs peuvent être indiquées, par exemple :

```
$ lb config --archive-areas "main contrib non-free"
```

La prise en charge d'*experimental* est disponible pour certains dérivés de Debian grâce à l'option `--mode`. L'option par défaut est *debian* mais seulement si vous construisez sur un système Debian ou un système inconnu. Si *lb config* est appelé sur un des dérivés pris en charge, il créera par défaut une image de ce dérivé. Si par exemple *lb config* est lancé en mode *ubuntu*, les noms de distribution et des zones d'archives pour ce dérivé spécifique seront gérés à la place de ceux de Debian. Le mode modifie également le comportement de *live-build* en fonction des dérivés.

Remarque : Les projets pour lesquels ces modes ont été ajoutés sont chargés d'aider les utilisateurs de ces options. Le Live

Systems Project, en retour, fournit un soutien de développement sur une base du meilleur effort seulement, en fonction des commentaires sur les projets dérivés que nous n'avons pas développés ou pris en charge nous-mêmes.

8.1.2 Miroirs de distribution

L'archive Debian est répliquée sur un grand réseau de miroirs autour du monde pour que les habitants de chaque région puissent choisir un miroir proche ayant la meilleure vitesse de téléchargement. Chacune des options `--mirror-*` régit quel miroir de distribution est utilisé dans les différentes étapes de la construction. Rappelez-vous dans les [« Étapes de la construction »](#) que l'étape **bootstrap** a lieu quand le chroot est initialement peuplé par *debootstrap* avec un système minimal, et l'étape **chroot** a lieu quand le chroot utilisé pour construire le système de fichiers du système live est construit. Ainsi, les commutateurs des miroirs correspondants sont utilisés pour ces étapes et plus tard, dans l'étape **binary**, les valeurs `--mirror-binary` et `--mirror-binary-security` sont utilisées, remplaçant tout miroir utilisé dans une étape antérieure.

8.1.3 Miroirs de distribution utilisés lors de la construction

Pour définir les miroirs de distribution utilisés pendant la construction pour pointer vers un miroir local, il suffit de fixer `--mirror-bootstrap`, `--mirror-chroot-security` et `--mirror-chroot-backports` comme suit.

```
$ lb config --mirror-bootstrap http://localhost/debian/ \
--mirror-chroot-security http://localhost/debian-security/ \
```

```
--mirror-chroot-backports http://localhost/debian-backports/
```

Le miroir chroot, indiqué avec `--mirror-chroot`, est par défaut la valeur de `--mirror-bootstrap`.

8.1.4 Miroirs de distribution utilisés pendant l'exécution

Les options `--mirror-binary*` régissent les miroirs de distribution placés dans l'image binaire. Elles peuvent être utilisées pour installer des paquets supplémentaires lors de l'exécution du système live. Les valeurs par défaut emploient `http.debian.net`, un service qui choisit un miroir géographiquement proche basé, entre autres choses, sur la famille IP de l'utilisateur et la disponibilité des miroirs. C'est un choix approprié lorsque vous ne pouvez pas prédire quel miroir sera le meilleur pour tous vos utilisateurs. Autrement, vous pouvez indiquer vos propres valeurs, comme indiqué dans l'exemple ci-dessous. Une image construite avec cette configuration ne serait appropriée que pour les utilisateurs sur un réseau où le "mirror" est accessible.

```
$ lb config --mirror-binary http://mirror/debian/ \
--mirror-binary-security http://mirror/debian-security/ \
--mirror-binary-backports http://mirror/debian-backports/
```

8.1.5 Dépôts additionnels

Vous pouvez ajouter d'autres dépôts, élargissant votre choix de paquets au-delà de ceux disponibles dans votre distribution cible. Cela peut être, par exemple, pour avoir des paquets rétroportés, personnalisés ou expérimentaux. Pour configurer des

dépôts supplémentaires, créez les fichiers `config/archives/-your-repository.list.chroot`, et/ou `config/archives/your-repository.list.binary`. Comme avec les options `--mirror-*`, ces fichiers donnent les dépôts utilisés dans l'étape **chroot** lors de la construction de l'image, et dans l'étape **binary**, c'est-à-dire pendant l'exécution du système live.

Par exemple, `config/archives/live.list.chroot` vous permet d'installer les paquets du dépôt des instantanés debian live pendant la construction du système live.

```
deb http://live-systems.org/ sid-snapshots main contrib non-free
```

Si vous ajoutez la même ligne à `config/archives/live.list.binary`, le dépôt sera ajouté au répertoire `/etc/apt/sources.list.d/` de votre système live.

Si ces fichiers existent, ils seront sélectionnés automatiquement.

Vous devriez également mettre la clé GPG utilisée pour signer le dépôt dans les fichiers `config/archives/your-repository.key.{binary,chroot}`

Si vous avez besoin d'un APT pinning personnalisé, les préférences APT peuvent être placées dans les fichiers `config/archives/your-repository.pref.{binary,chroot}` et elles seront automatiquement ajoutées à votre système live dans le répertoire `/etc/apt/preferences.d/`.

8.2 Choisir les paquets à installer

Il y a un certain nombre de façons pour choisir quels paquets *live-build* s'installeront dans votre image, couvrant toute une variété de besoins. Vous pouvez tout simplement nommer les

paquets individuels à installer dans une liste de paquets. Vous pouvez également choisir des métapaquets dans ces listes, ou les sélectionner en utilisant les champs de contrôle de fichiers des paquets. Enfin, vous pouvez placer des paquets dans votre arbre `config/` qui est bien adapté aux essais de nouveaux paquets ou expérimentaux avant qu'ils ne soient disponibles sur un dépôt.

8.2.1 Listes de paquets

Les listes de paquets sont un excellent moyen d'exprimer quels paquets doivent être installés. La syntaxe de la liste gère des sections conditionnelles, ce qui les rend faciles à construire et à adapter pour leur utilisation dans des configurations multiples. Les noms des paquets peuvent également être injectés dans la liste en utilisant des assistants de shell pendant la construction.

Remarque : Le comportement de *live-build* pour indiquer un paquet qui n'existe pas est déterminé par votre choix de l'utilitaire APT. Consultez [« Choisir apt ou aptitude »](#) pour plus de détails.

8.2.2 Utilisation des métapaquets

La façon la plus simple de remplir votre liste de paquets consiste à utiliser un métapaquet de tâche maintenu par votre distribution. Par exemple :

```
$ lb config
$ echo task-gnome-desktop > config/package-lists/desktop.list.chroot
```

Cela remplace l'ancienne méthode des listes prédéfinies gérée

dans `live-build 2.x`. Contrairement aux listes prédéfinies, les métapaquets ne sont pas spécifiques au projet Live Systems. Au lieu de cela, ils sont maintenus par des groupes de travail spécialisés dans la distribution et reflètent donc le consensus de chaque groupe sur les paquets pour mieux servir les besoins des utilisateurs. Ils couvrent également une gamme beaucoup plus large des cas d'utilisation que les listes prédéfinies qu'ils remplacent.

Tous les métapaquets de tâches sont préfixés avec `task-`, donc un moyen rapide pour déterminer lesquels sont disponibles (même s'il peut y avoir une poignée de faux positifs dont le nom correspond mais qui ne sont pas des métapaquets) est de faire correspondre le nom du paquet avec :

```
$ apt-cache search --names-only ^task-
```

En plus, vous trouverez d'autres métapaquets à des fins diverses. Certains sont des sous-ensembles de paquets de tâches plus larges, comme `gnome-core`, tandis que d'autres sont des pièces individuelles spécialisées d'un Debian Pure Blend, comme les métapaquets `education-*`. Pour lister tous les métapaquets dans l'archive, installez le paquet `debtags` et listez tous les paquets ayant l'étiquette `role ::metapackage` comme suit :

```
$ debtags search role ::metapackage
```

8.2.3 Listes de paquets locaux

Que vous listiez des métapaquets, des paquets individuels ou

une combinaison des deux, toutes les listes de paquets locaux sont stockées dans `config/package-lists/`. Comme plus d'une liste peut être utilisée, cela se prête bien à une conception modulaire. Par exemple, vous pouvez décider de consacrer une liste à un choix particulier de bureau, l'autre à une collection de paquets connexes qui pourraient aussi bien être utilisés au-dessus d'un bureau différent. Cela vous permet d'expérimenter avec différentes combinaisons d'ensembles de paquets avec un minimum de tracas en utilisant des listes communes entre les différents projets d'images live.

Les listes de paquets qui existent dans ce répertoire ont besoin d'avoir un suffixe `.list` pour être traitées, puis un suffixe d'étape supplémentaire `.chroot` ou `.binary` pour indiquer à quelle étape la liste est destinée.

Remarque : Si vous n'indiquez pas le suffixe de l'étape, la liste sera utilisée pour les deux étapes. Normalement, vous voulez indiquer `.list.chroot` de sorte que les paquets soient seulement installés dans le système de fichiers live et ne pas avoir une copie supplémentaire des `.deb` placée sur le support.

8.2.4 Listes de paquets locaux pour l'étape binary

Pour faire une liste pour l'étape binary, placez un fichier avec le suffixe `.list.binary` dans `config/package-lists/`. Ces paquets ne sont pas installés dans le système de fichiers live, mais sont inclus sur le support live sous `pool/`. Vous utiliserez généralement cette liste avec une des variantes d'installation non-live. Comme mentionné ci-dessus, si vous voulez que cette liste soit la même que votre liste pour l'étape chroot, utilisez simplement le suffixe `.list`.

8.2.5 Listes de paquets générées

415

422

416 Il arrive parfois que la meilleure façon de composer une liste soit de la générer avec un script. Toute ligne commençant par un point d'exclamation indique une commande à exécuter dans le chroot lorsque l'image est construite. Par exemple, on pourrait inclure la ligne `! grep-aptavail -n -sPackage -FPriority standard | sort` dans une liste de paquets qui permet de produire une liste triée des paquets disponibles avec `Priority : standard`.

417 En fait, la sélection des paquets avec la commande `grep-aptavail` (du paquet `dctrl-tools`) est si utile que `live-build` fournit un script `Packages` à titre de commodité. Ce script accepte deux arguments : `field` et `pattern`. Ainsi, vous pouvez créer une liste avec le contenu suivant :

418

```
$ lb config
$ echo '! Packages Priority standard' > config/package-lists/standard.list.<↵
chroot
```

```
#if ARCHITECTURES amd64
ia32-libs
#endif
```

Vous pouvez tester pour un certain nombre de valeurs, par exemple pour installer *memtest86+* si `--architectures i386` ou `--architectures amd64` est indiqué :

423

424

```
#if ARCHITECTURES i386 amd64
memtest86+
#endif
```

Vous pouvez également tester avec des variables pouvant contenir plus d'une valeur, par exemple pour installer *vrms* si `contrib` ou `non-free` est indiqué via `--archive-areas` :

425

426

```
#if ARCHIVE_AREAS contrib non-free
vrms
#endif
```

L'imbrication des conditions n'est pas prise en charge.

427

8.2.6 Utiliser des conditions dans les listes de paquets

419

420 Toutes les variables de configuration de *live-build* stockées dans `config/*` (sans le préfixe `LB_`) peuvent être utilisées dans des instructions conditionnelles dans les listes de paquets. Généralement, cela correspond à toute option `lb config` en majuscule et avec tirets changés en caractères de soulignement. Mais en pratique, ce ne sont que celles qui influencent la sélection des paquets qui font sens, comme `DISTRIBUTION`, `ARCHITECTURES` ou `ARCHIVE_AREAS`.

421 Par exemple, pour installer *ia32-libs* si `--architectures amd64` est indiqué :

8.2.7 Suppression de paquets lors de l'installation

428

Il est possible de lister des paquets dans des fichiers utilisant les extensions `.list.chroot_live` et `.list.chroot_install` à l'intérieur du répertoire `config/package-lists`. Si une liste « install » et une liste « live » existent conjointement, les paquets dans la liste `.list.chroot_live` seront supprimés par un hook après l'installation (si l'utilisateur utilise l'installateur). Les paquets

429

dans la liste `.list.chroot_install` sont présents à la fois dans le système live et dans le système installé. Il s'agit d'un paramétrage spécial pour l'installateur et peut se révéler utile si vous avez choisi `--debian-installer live` dans votre configuration, et souhaitez supprimer des paquets spécifiques aux systèmes live lors de l'installation.

8.2.8 Tâches de bureau et de langue

Les tâches de bureau et de langue sont des cas particuliers qui ont besoin d'une certaine planification et de configuration supplémentaire. Dans l'installateur Debian, si le support a été préparé pour un environnement de bureau particulier, la tâche correspondante sera automatiquement installée. Ainsi, il y a tâches internes `gnome-desktop`, `kde-desktop`, `lxde-desktop` et `xfce-desktop`, dont aucune n'est proposée dans le menu `tasksel`. De même, il n'y a pas d'élément de menu pour les tâches de langue, mais le choix de la langue de l'utilisateur lors de l'installation influence le choix des tâches de langue correspondantes.

Lors du développement d'une image de bureau live, l'image s'amorce généralement directement sur un bureau de travail. Les choix de l'environnement de bureau et la langue par défaut ont été faits pendant la construction et non pas pendant l'exécution comme dans le cas de l'installateur de Debian. Cela ne veut pas dire qu'une image live ne pourrait pas être construite pour prendre en charge plusieurs environnements de bureau ou plusieurs langues et offrir à l'utilisateur un choix, mais ce n'est pas le comportement par défaut de *live-build*.

Comme aucune disposition n'est faite automatiquement pour les tâches de la langue, qui comprennent des éléments tels que des polices spécifiques à la langue et des paquets de méthodes de saisie, vous devez les indiquer dans votre configuration si vous les

voulez. Par exemple, une image de bureau GNOME contenant la prise en charge de l'allemand pourrait inclure les métapaquets de tâches suivants :

```
$ lb config
$ echo "task-gnome-desktop task-laptop" >> config/package-lists/my.list.<↵
chroot
$ echo "task-german task-german-desktop task-german-gnome-desktop" >> <↵
config/package-lists/my.list.chroot
```

8.2.9 Version et type de noyau

Un ou plusieurs types de noyau seront inclus dans votre image par défaut, en fonction de l'architecture. Vous pouvez choisir différents types avec l'option `--linux-flavours`. Chaque type est suffixé à partir de `linux-image` pour former le nom de chaque métapaquet qui dépend à son tour d'un paquet noyau exact à inclure dans votre image.

Ainsi, par défaut, une image pour l'architecture `amd64` comprendra le métapaquet `linux-image-amd64`, et une image pour l'architecture `i386` comprendra les métapaquets `linux-image-486` et `linux-image-686-pae`. Au moment de la rédaction, ces paquets dépendent de `linux-image-3.2.0-4-amd64`, `linux-image-3.2.0-4-486` et `linux-image-3.2.0-4-686-pae`, respectivement.

Lorsque plus d'une version du paquet du noyau est disponible dans vos archives configurées, vous pouvez indiquer un nom du paquet du noyau différent avec l'option `--linux-packages`. Par exemple, supposons que vous construisiez une image pour l'architecture `amd64` et ajoutiez l'archive expérimentale pour faire des test afin que vous puissiez installer le noyau `linux-image-3.7-trunk-amd64`. Vous devez configurer cette image comme suit :

439

```
$ lb config --linux-packages linux-image-3.7-trunk
$ echo "deb http://ftp.debian.org/debian/ experimental main" > config/↔
archives/experimental.list.chroot
```

440 8.2.10 Noyaux personnalisés

441 Vous pouvez créer et inclure vos propres noyaux personnalisés, à condition qu'ils soient intégrés dans le système de gestion des paquets Debian. Le système de *live-build* ne gère pas les noyaux qui ne sont pas construits sous forme de paquets `.deb`.

442 La façon correcte et recommandée de déployer vos propres paquets du noyau est de suivre les instructions dans le `kernel-handbook`. N'oubliez pas de modifier l'ABI et les suffixes de manière appropriée, puis d'inclure une construction complète des paquets `linux` et `linux-latest` dans votre dépôt.

443 Si vous optez pour la construction des paquets du noyau sans les métapaquets correspondants, vous devez indiquer une chaîne `--linux-packages` appropriée tel que discuté dans < **Version et type de noyau** >. Comme nous l'expliquons dans < **Installation de paquets modifiés ou tiers** >, il est préférable que vous incluiez vos paquets de noyau personnalisés à votre propre dépôt, bien que les alternatives discutées dans cette section fonctionnent bien également.

444 Donner des conseils sur la façon de personnaliser votre noyau sort du cadre de ce document. Toutefois, vous devez au moins vous assurer que votre configuration répond à ces exigences minimales :

- 445 • Utilisez un ramdisk initial.
- 446 • Incluez un module d'union de systèmes de fichiers (par exemple

aufs).

- Incluez tous les autres modules du système de fichiers requis pour votre configuration (habituellement `squashfs`). 447

8.3 Installation de paquets modifiés ou tiers 448

Bien que ce soit contre la philosophie d'un système live, il peut parfois être nécessaire de construire un système live avec des versions modifiées des paquets du dépôt Debian. Cela peut être pour modifier ou prendre en charge des fonctionnalités supplémentaires, des langues et la marque, ou même pour supprimer des éléments indésirable dans les paquets existants. De même, les paquets « tiers » peuvent être utilisés pour ajouter des fonctionnalités sur mesure et/ou propriétaires. 449

Cette section ne couvre pas les conseils concernant la construction ou la maintenance des paquets modifiés. La méthode de Joachim Breitner 'How to fork privately' < <http://www.joachim-breitner.de/blog/archives/282-How-to-fork-privately.html> > peut, cependant, vous intéresser. La création de paquets sur mesure est traitée dans le guide du nouveau mainteneur Debian à < <https://www.debian.org/doc/maint-guide/> > et ailleurs 450

Il y a deux façons d'installer des paquets personnalisés modifiés : 451

- `packages.chroot` 452
- Utiliser un dépôt APT personnalisé 453

Utiliser `packages.chroot` est plus simple à réaliser et utile pour les personnalisations ponctuelles mais a un certain nombre d'inconvénients, alors qu'utiliser un dépôt personnalisé APT est plus fastidieux à mettre en place. 454

8.3.1 Utiliser `packages.chroot` pour installer des paquets personnalisés

4655

Pour installer un paquet personnalisé, il suffit de le copier dans le répertoire `config/packages.chroot/`. Les paquets qui sont dans ce répertoire seront automatiquement installés dans le système live pendant la construction du système - vous n'avez pas besoin de les indiquer ailleurs.

Les paquets **doivent** être nommés de la manière prescrite. Une façon simple de le faire consiste à utiliser `dpkg-name`.

L'utilisation de `packages.chroot` pour l'installation de paquets personnalisés a des inconvénients :

- Il n'est pas possible d'utiliser APT de façon sécurisée.
- Vous devez installer tous les paquets appropriés dans le répertoire `config/packages.chroot/`.
- Il ne se prête pas au stockage de configurations des systèmes live dans le contrôle de révision.

8.3.2 Utiliser un dépôt APT pour installer des paquets personnalisés.

Contrairement à l'utilisation de `packages.chroot`, lorsque vous utilisez un dépôt personnalisé APT, vous devez vous assurer que vous indiquez les paquets ailleurs. Voir < [Choisir les paquets à installer](#) > pour plus de détails.

S'il peut sembler inutile de créer un dépôt APT pour installer des paquets personnalisés, l'infrastructure peut être facilement réutilisée à une date ultérieure pour offrir les mises à jour des paquets modifiés.

8.3.3 Les paquets personnalisés et APT

live-build utilise apt pour installer tous les paquets dans le système live, il héritera donc des comportements de ce logiciel. Un exemple pertinent est que (en supposant une configuration par défaut), s'il y a un paquet disponible dans deux dépôts différents avec des numéros de version différents, APT choisira d'installer le paquet ayant le numéro de version le plus grand.

Pour cette raison, vous pouvez incrémenter le numéro de version dans les fichiers `debian/changelog` de vos paquets personnalisés pour vous assurer que votre version modifiée sera installée au lieu d'une autre provenant des dépôts officiels Debian. Cela peut aussi être fait en modifiant les préférences d'APT pinning dans le système live – voir < [APT pinning](#) > pour plus d'informations.

8.4 Configuration d'APT pendant la construction

Vous pouvez configurer APT par un certain nombre d'options appliquées uniquement pendant la construction. (La configuration d'APT utilisée dans le système live en fonctionnement peut être configurée de façon normale pour un système live, c'est-à-dire en incluant les configurations appropriées dans `config/includes.chroot/`.) Pour une liste complète, regardez les options commençant par apt dans la page de manuel de `lb_config`.

8.4.1 Choisir apt ou aptitude

Vous pouvez choisir d'utiliser soit *apt*, soit *aptitude*. Le choix du logiciel utilisé dépend de l'argument `--apt` de `lb config`. Choisissez la méthode ayant le comportement que vous préférez pour l'installation de paquets, la différence notable étant la manière dont les paquets manquants sont traités.

- apt : Avec cette méthode, si un paquet manquant est indiqué, l'installation va échouer. C'est le réglage par défaut.
- aptitude : Avec cette méthode, si un paquet manquant est indiqué, l'installation va réussir.

8.4.2 Utilisation d'un proxy avec APT

Une configuration communément requise par APT est de gérer la construction d'une image derrière un proxy. Vous pouvez indiquer votre proxy APT avec les options `--apt-ftp-proxy` ou `--apt-http-proxy` si nécessaire, par exemple

```
$ lb config --apt-http-proxy http://proxy/
```

8.4.3 Régler APT pour économiser de l'espace

Vous pouvez avoir besoin d'économiser de l'espace sur le support d'image, auquel cas l'une ou l'autre ou les deux options suivantes peuvent être d'intérêt.

Si vous ne voulez pas inclure les index d'APT dans l'image, vous pouvez les omettre avec :

```
$ lb config --apt-indices false
```

Cela n'influencera pas les entrées dans `/etc/apt/sources.list`, mais seulement le fait que `/var/lib/apt` contienne les fichiers index ou non. La contrepartie est qu'APT a besoin de ces index afin d'opérer dans le système live. Par conséquent, avant de procéder

à `apt-cache search` ou `apt-get install` par exemple, l'utilisateur doit faire `apt-get update` pour créer ces index.

Si vous trouvez que l'installation des paquets recommandés fait trop gonfler votre image, à condition d'être prêt à faire face aux conséquences décrites ci-dessous, vous pouvez désactiver l'option par défaut d'APT avec :

```
$ lb config --apt-recommends false
```

La conséquence la plus importante de la désactivation des recommandations est que `live-boot` et `live-config` recommandent certains paquets qui fournissent des fonctionnalités importantes utilisées par la plupart de configurations live, telles que `user-setup` qui est recommandé par `live-config` qui est utilisé pour créer l'utilisateur live. Sauf exception, vous aurez besoin de rajouter au moins certaines de ces recommandations à vos listes de paquets ou votre image ne fonctionnera pas comme prévu, si elle fonctionne. Regardez les paquets recommandés pour chacun des paquets `live-*` inclus dans votre construction et si vous n'êtes pas sûr de pouvoir les omettre, ajoutez-les à nouveau dans vos listes de paquets.

La conséquence la plus générale est que si vous n'installez pas les paquets recommandés par un paquet, c'est-à-dire les « paquets qu'on trouverait avec celui-ci dans toute installation standard » (Debian Policy Manual, section 7.2), certains paquets dont vous avez vraiment besoin peuvent être omis. Par conséquent, nous vous suggérons d'examiner la différence que la désactivation des recommandations induit sur votre liste de paquets (voir le fichier `binary.packages` généré par `lb build`) et incluez dans votre liste tous les paquets manquants que vous souhaitez toujours installer. Alternativement, si seulement un petit nombre de paquets que

vous ne souhaitez pas est exclus, laissez les recommandations activées et définissez une priorité APT pin négative sur les paquets sélectionnés pour éviter les installer, comme expliqué dans < **APT pinning** >.

8.4.4 Passer des options à apt ou aptitude

S'il n'y a pas d'option `lb config` pour modifier le comportement d'APT de la façon dont vous avez besoin, utilisez `--apt-options` ou `--aptitude-options` pour passer des options par le biais de l'outil APT configuré. Consultez les pages de manuel `apt` et `aptitude` pour les détails. Notez que les deux options ont des valeurs par défaut que vous aurez besoin de conserver en plus des remplacements que vous pouvez fournir. Ainsi, par exemple, supposons que vous ayez inclus quelque chose de `snapshot.debian.org` à des fins de test et que vous vouliez indiquer `Acquire::Check-Valid-Until=false` pour satisfaire APT avec le fichier `Release` obsolète. Vous le feriez comme dans l'exemple suivant, avec l'ajout de la nouvelle option après la valeur par défaut `--yes` :

```
$ lb config --apt-options "--yes -oAcquire::Check-Valid-Until=false"
```

Veuillez lire attentivement les pages de manuel pour bien comprendre ces options et quand les utiliser. Ce n'est qu'un exemple et ne doit pas être interprété comme un conseil pour configurer votre image de cette façon. Par exemple, cette option ne serait pas appropriée pour une version finale d'une image live.

Pour les configurations plus compliquées concernant des options `apt.conf`, vous pourriez vouloir créer un fichier `config/apt/-`

`apt.conf`. Consultez aussi les autres options `apt-*` pour obtenir quelques raccourcis pratiques pour les options fréquemment utilisées.

8.4.5 APT pinning

Pour plus de contexte, veuillez d'abord lire la page de manuel `apt_preferences(5)`. APT pinning peut être configuré soit pendant la construction, soit pendant l'exécution. Dans le premier cas, créez `config/archives/*.pref`, `config/archives/*.pref.chroot`, et `config/apt/preferences`. Dans le second cas, créez `config/includes.chroot/etc/apt/preferences`.

Imaginons que vous voulez construire un système live **jessie** mais qu'il faut installer tous les paquets live qui finissent dans l'image binaire de **sid** pendant la construction. Vous devez ajouter **sid** à votre fichier APT sources et fixer tous les paquets live avec une priorité supérieure mais tous les autres paquets avec une priorité inférieure à la priorité par défaut de sorte que seuls les paquets que vous voulez soient installés à partir de **sid** pendant la construction et que tous les autres viennent de la distribution du système cible, **jessie**. Ce qui suit devrait accomplir cela :

```
$ echo "deb http://mirror/debian/ sid main" > config/archives/sid.list.<↵
chroot
$ cat >> config/archives/sid.pref.chroot << EOF
Package : live-*
Pin : release n=sid
Pin-Priority : 600

Package : *
Pin : release n=sid
Pin-Priority : 1
EOF
```

495

Une priorité pin négative évitera installer un paquet, comme dans le cas où vous ne voudriez pas un paquet qui est recommandé par un autre paquet. Supposons que vous construisiez une image LXDE en utilisant `task-lxde-desktop` dans `config/package-lists/desktop.list.chroot` mais que vous ne vouliez pas que l'utilisateur soit invité à stocker les mots de passe wifi dans le trousseau de clés. Cette liste comprend *lxde-core*, qui recommande *gksu*, qui à son tour recommande *gnome-keyring*. Donc, vous voulez omettre le paquet recommandé *gnome-keyring*. Cela peut être fait en ajoutant ce qui suit à `config/apt/preferences` :

496

```
Package : gnome-keyring
Pin : version *
Pin-Priority : -1
```

Personnalisation des contenus

9. Personnalisation des contenus

Ce chapitre aborde la personnalisation fine des contenus du système live au-delà du simple choix des paquets à inclure. Les inclusions vous permettent d'ajouter ou de remplacer des fichiers arbitraires dans votre image du système live, les hooks vous permettent d'exécuter des commandes arbitraires dans différentes étapes de la construction et au démarrage et la préconfiguration (preseeding) vous permet de configurer les paquets quand ils sont installés en fournissant des réponses aux questions debconf.

9.1 Includes

Bien qu'idéalement un système live comprendrait des fichiers entièrement fournis par des paquets non modifiés, il peut être pratique de fournir ou de modifier certains contenus par le biais de fichiers. Avec les « includes », il est possible d'ajouter (ou remplacer) des fichiers arbitraires dans votre image du système live. *live-build* prévoit deux mécanismes pour leur utilisation :

- Chroot local includes : Ils vous permettent d'ajouter ou remplacer des fichiers sur le système de fichiers chroot/Live. Veuillez consulter < [Live/chroot local includes](#) > pour plus d'informations.
- Binary local includes : Ils vous permettent d'ajouter ou de remplacer des fichiers dans l'image binaire. Veuillez consulter < [Binary local includes](#) > pour plus d'informations.

Veuillez consulter les < [Termes](#) > pour plus d'informations sur la distinction entre les images “Live” et “binary”.

9.1.1 Live/chroot local includes

Les chroot local includes peuvent être utilisés pour ajouter ou remplacer des fichiers dans le système de fichiers chroot/Live afin qu'ils puissent être utilisés dans le système Live. Une utilisation typique est de peupler l'arborescence du répertoire de l'utilisateur (/etc/skel) utilisée par le système live pour créer le répertoire home de l'utilisateur Live. Une autre est de fournir des fichiers de configuration qui peuvent être simplement ajoutés ou remplacés à l'image sans traitement, voir < [Live/chroot local hooks](#) > si le traitement est nécessaire.

Pour inclure des fichiers, il suffit de les ajouter à votre répertoire config/includes.chroot. Ce répertoire correspond au répertoire racine / du système live. Par exemple, pour ajouter un fichier /var/www/index.html dans le système live, utilisez :

```
$ mkdir -p config/includes.chroot/var/www
$ cp /path/to/my/index.html config/includes.chroot/var/www
```

Votre configuration aura alors le schéma suivant :

```
-- config
[... ]
|-- includes.chroot
|   |-- var
|       |-- www
|           |-- index.html
[... ]
```

Les chroot local includes sont installés après l'installation de paquets de sorte que les fichiers installés par les paquets sont remplacés.

9.1.2 Binary local includes

Pour inclure des matériels tels que des documents ou des vidéos sur le système de fichiers des supports, afin qu'ils soient accessibles dès l'insertion du support sans démarrer le système live, vous pouvez utiliser binary local includes. Cela fonctionne de façon similaire aux chroot local includes. Par exemple, supposons que les fichiers `~/video_demo.*` sont des vidéos de démonstration du système live décrits par et liés par une page d'index HTML. Copiez simplement le matériel dans `config/includes.binary/` comme suit :

```
$ cp ~/video_demo.* config/includes.binary/
```

Ces fichiers apparaissent maintenant dans le répertoire racine du support live.

9.2 Hooks

Les hooks permettent l'exécution des commandes dans les étapes de la construction chroot et binary afin de personnaliser l'image.

9.2.1 Live/chroot local hooks

Pour exécuter des commandes à l'étape chroot, créer un script hook avec le suffixe `.hook.chroot` contenant les commandes dans le répertoire `config/hooks/`. Le hook s'exécutera dans le chroot après que le reste de votre configuration chroot ait été appliqué, donc n'oubliez pas de vous assurer que votre configuration inclut tous les paquets et les fichiers dont votre hook a besoin pour

fonctionner. Consultez les exemples de scripts chroot hook pour diverses tâches courantes de personnalisation chroot fournis dans `/usr/share/doc/live-build/examples/hooks` que vous pouvez copier ou faire un lien symbolique pour les utiliser dans votre propre configuration.

9.2.2 Hooks pendant le démarrage

Pour exécuter des commandes pendant le démarrage, vous pouvez fournir *live-config* hooks comme expliqué dans la section "Personnalisation" de sa page de manuel. Examinez les hooks de *live-config* fournis dans `/lib/live/config/`, en notant les numéros de séquence. Fournissez ensuite votre propre hook précédé d'un numéro de séquence appropriée, soit comme un chroot local include dans `config/includes.chroot/lib/live/-config/`, soit comme un paquet personnalisé tel que discuté dans [« Installation des paquets modifiés ou de tiers »](#).

9.2.3 Binary local hooks

Pour exécuter des commandes à l'étape binaire, créez un script hook avec le suffixe `.hook.binary` contenant les commandes dans le répertoire `config/hooks/`. Le hook sera exécuté après toutes les autres commandes binaires, mais avant `binary_checksums`, la dernière commande binaire. Les commandes de votre hook ne s'exécutent pas dans le chroot, afin de prendre soin de ne pas modifier les fichiers en dehors de l'arbre de construction, ou vous pourriez endommager votre système de construction ! Consultez les exemples de scripts de binary hook pour diverses tâches courantes de personnalisation binaires fournis dans `/usr/share/doc/live-build/examples/hooks` que vous pouvez copier ou lier symboliquement pour les utiliser dans votre propre configuration.

524 9.3 Préconfigurer questions de debconf

525 Les fichiers dans le répertoire `config/preseed/` avec le suffixe
.`cfg` suivis de l'étape (`.chroot` or `.binary`) sont considérés comme
des fichiers de préconfiguration debconf et sont installés par *live-*
build en utilisant `debconf-set-selections`.

526 Pour plus d'informations sur debconf, veuillez consulter
`debconf(7)` dans le paquet *debconf*.

Personnalisation des comportements pendant l'exécution

10. Personnalisation des comportements pendant l'exécution

Toute la configuration qui est faite pendant l'exécution est faite par *live-config*. Voici quelques options parmi les plus courantes de *live-config* qui peuvent intéresser les utilisateurs. Une liste complète de toutes les possibilités peut être trouvée dans la page de manuel de *live-config*.

10.1 Personnalisation de l'utilisateur live

Une considération importante est que l'utilisateur live est créé par *live-boot* au démarrage, non pas par *live-config* pendant la construction. Cela influence non seulement l'emplacement où les documents relatifs à l'utilisateur live sont introduits dans la construction, tel que discuté dans < **Live/chroot local includes** >, mais aussi tous les groupes et autorisations associés à l'utilisateur live.

Vous pouvez indiquer d'autres groupes pour l'utilisateur live en utilisant une des possibilités pour configurer *live-config*. Par exemple, pour ajouter l'utilisateur live au groupe fuse, vous pouvez ajouter le fichier suivant dans `config/includes.chroot/etc/live/config/user-setup.conf` :

```
LIVE_USER_DEFAULT_GROUPS="audio cdrom dip floppy video plugdev netdev ↵
powerdev scanner bluetooth fuse"
```

ou utiliser `live-config.user-default-groups=audio,cdrom,dip,floppy,video,plugdev,netdev,powerdev,scanner,bluetooth,fuse` comme paramètre d'amorçage.

Il est également possible de changer le nom de l'utilisateur par défaut « user » et du mot de passe par défaut « live ». Si vous voulez faire cela, vous pouvez le faire facilement comme suit :

Pour modifier le nom de l'utilisateur par défaut, vous pouvez simplement l'indiquer dans votre configuration :

```
$ lb config --bootappend-live "boot=live components username=live-user"
```

Une façon possible de changer le mot de passe par défaut est d'utiliser un hook comme décrit dans < **Hooks pendant le démarrage** >. Pour ce faire vous pouvez utiliser le hook "passwd" de `/usr/share/doc/live-config/examples/hooks`, ajouter un préfixe correct (par exemple 2000-passwd) et l'ajouter à `config/includes.chroot/lib/live/config/`

10.2 Personnalisation des paramètres régionaux et de la langue

Au démarrage du système live, la langue est impliquée dans deux étapes :

- la génération des paramètres régionaux
- le réglage de la disposition du clavier

Les paramètres régionaux par défaut pendant la construction d'un système Live sont `locales=en_US.UTF-8`. Pour définir les paramètres régionaux qui doivent être générés, utilisez le paramètre `locales` dans l'option `--bootappend-live` de `lb config`, par exemple

```
$ lb config --bootappend-live "boot=live components locales=de_CH.UTF-8"
```

Plusieurs paramètres régionaux peuvent être indiqués dans une liste séparée par des virgules.

Ce paramètre, ainsi que les paramètres de configuration du clavier indiqués ci-dessous, peut également être utilisé sur la ligne de commande du noyau. On peut indiquer des paramètres régionaux avec `language_country` (dans ce cas, le codage par défaut est utilisé) ou l'expression complète `language_country.encoding`. Une liste des paramètres régionaux et le codage pour chacun peuvent être trouvés dans `/usr/share/i18n/SUPPORTED`.

La configuration du clavier pour la console et pour X est faite par `live-config` en utilisant le paquet `console-setup`. Pour les configurer, utilisez les paramètres de démarrage `keyboard-layouts`, `keyboard-variants`, `keyboard-options` et `keyboard-model` avec l'option `--bootappend-live`. On peut trouver les options valides dans `/usr/share/X11/xkb/rules/base.lst`. Pour trouver les dispositions et les variantes correspondantes à une langue, essayez de rechercher le nom anglais de la nation où la langue est parlée, par exemple :

```
$ egrep -i '(!|german.*switzerland)' /usr/share/X11/xkb/rules/base.lst
! model
! layout
  ch          German (Switzerland)
! variant
  legacy      ch : German (Switzerland, legacy)
  de_noadkeys ch : German (Switzerland, eliminate dead keys)
  de_sundeadkeys ch : German (Switzerland, Sun dead keys)
  de_mac      ch : German (Switzerland, Macintosh)
! option
```

Chaque variante présente une description de la disposition

appliquée.

Souvent, seule la disposition doit être configurée. Par exemple, pour obtenir les fichiers des paramètres régionaux de l'allemand et la disposition du clavier suisse allemand dans X, utilisez :

```
$ lb config --bootappend-live "boot=live components locales=de_CH.UTF-8 \
keyboard-layouts=ch"
```

Toutefois, pour les cas d'utilisation très spécifiques, on peut inclure d'autres paramètres. Par exemple, pour mettre en place un système français avec une disposition French-Dvorak (Bépo) avec un clavier USB TypeMatrix EZ-Reach 2030, utilisez :

```
$ lb config --bootappend-live \
"boot=live components locales=fr_FR.UTF-8 keyboard-layouts=fr keyboard-
variants=bepo keyboard-model=tm2030usb"
```

Plusieurs valeurs peuvent être indiquées dans des listes séparées par des virgules pour chacune des options `keyboard-*`, à l'exception de `keyboard-model` qui accepte une seule valeur. Veuillez consulter la page de manuel `keyboard(5)` pour plus de détails et des exemples des variables `XKBMODEL`, `XKBLAYOUT`, `XKBVARIANT` et `XKBOPTIONS`. Si plusieurs valeurs `keyboard-variants` sont données, elles seront jumelées une à une avec les valeurs `keyboard-layouts` (voir `setxkbmap(1)` option `-variant`). On peut utiliser des valeurs vides ; par exemple pour régler deux dispositions, une par défaut US QWERTY et l'autre US Dvorak, utilisez :

```
$ lb config --bootappend-live \
  "boot=live components keyboard-layouts=us,us keyboard-variants=,dvorak<->
  "
```

10.3 Persistance

Le paradigme d'un Live CD est d'être un système pré-installé qui amorce sur un support en lecture seule, comme un cdrom, où les données et les modifications ne survivent pas aux redémarrages du matériel hôte qui l'exécute.

Un système live est une généralisation de ce paradigme et gère ainsi d'autres supports en plus de CDs. Malgré tout, dans son comportement par défaut, il doit être considéré en lecture seule et toutes les évolutions pendant l'exécution du système sont perdues à l'arrêt.

La « persistance » est un nom commun pour les différents types de solutions pour sauver, après un redémarrage, certaines ou toutes les données, de cette évolution pendant l'exécution du système. Pour comprendre comment cela fonctionne, il peut être utile de savoir que même si le système est démarré et exécuté à partir d'un support en lecture seule, les modifications des fichiers et répertoires sont écrites sur des supports inscriptibles, typiquement un disque ram (tmpfs) et les données des disques RAM ne survivent pas à un redémarrage.

Les données stockées sur ce disque virtuel doivent être enregistrées sur un support inscriptible persistant comme un support de stockage local, un partage réseau ou même une séance d'un CD/DVD multisession (ré)inscriptible. Tous ces supports sont pris en charge dans les systèmes live de différentes manières, et tous, à part le dernier, nécessitent un paramètre d'amorçage spécial à préciser au moment du démarrage : persistence.

Si le paramètre de démarrage persistence est réglé (et nopersistence n'est pas utilisé), les supports de stockage locaux (par exemple les disques durs, clés USB) seront examinés pour trouver des volumes persistants pendant le démarrage. Il est possible de limiter les types de volumes persistants à utiliser en indiquant certains paramètres de démarrage décrits dans la page de manuel *live-boot*(7). Un volume persistant est un des éléments suivants :

- une partition, identifiée par son nom GPT.
- un système de fichiers, identifié par son étiquette de système de fichiers.
- un fichier image situé sur la racine d'un système de fichiers en lecture (même une partition NTFS d'un système d'exploitation étranger), identifié par son nom de fichier.

L'étiquette du volume pour les overlays doit être persistence mais elle sera ignorée à moins de contenir dans sa racine un fichier nommé `persistence.conf` qui est utilisé pour personnaliser entièrement la persistance du volume, c'est-à-dire indiquer les répertoires que vous voulez sauvegarder dans votre volume de persistance après un redémarrage. Voir < **Le fichier persistence.conf** > pour plus de détails.

Voici quelques exemples montrant comment préparer un volume à utiliser pour la persistance. Cela peut être, par exemple, une partition ext4 sur un disque dur ou sur une clé usb créée avec :

```
# mkfs.ext4 -L persistence /dev/sdb1
```

Voir aussi < **Utilisation de l'espace disponible sur une clé USB** >.

Si vous avez déjà une partition sur votre périphérique, vous pouvez simplement modifier l'étiquette avec l'un des exemples suivants :

```
# tune2fs -L persistence /dev/sdb1 # for ext2,3,4 filesystems
```

Voici un exemple montrant comment créer un fichier image avec un système de fichiers ext4 pour être utilisé pour la persistance :

```
$ dd if=/dev/null of=persistence bs=1 count=0 seek=1G # for a 1GB sized ↵
    image file
$ /sbin/mkfs.ext4 -F persistence
```

Une fois que le fichier image est créé, à titre d'exemple, pour rendre /usr persistant mais seulement enregistrer les modifications que vous apportez à ce répertoire et non pas tout le contenu de /usr, vous pouvez utiliser l'option « union ». Si le fichier image se trouve dans votre répertoire personnel, copiez-le à la racine du système de fichiers de votre disque dur et montez-le dans /mnt comme suit :

```
# cp persistence /
# mount -t ext4 /persistence /mnt
```

Ensuite, créez le fichier persistence.conf ajoutant du contenu et démontez le fichier image.

```
# echo "/usr union" >> /mnt/persistence.conf
# umount /mnt
```

Maintenant, redémarrez dans votre support live avec le paramètre de démarrage "persistence".

10.3.1 Le fichier persistence.conf

Un volume ayant l'étiquette persistence doit être configuré avec un fichier persistence.conf pour créer des répertoires persistants arbitraires. Ce fichier, situé sur le système de fichiers racine du volume, contrôle quels répertoires il rend persistants, et de quelle manière.

La façon de configurer les montages overlays est décrite en détail dans la page de manuel persistence.conf(5), mais un simple exemple devrait suffire pour la plupart des utilisations. Imaginons que nous voulions rendre notre répertoire personnel et APT cache persistants dans un système de fichiers ext4 sur la partition /dev/sdb1 :

```
# mkfs.ext4 -L persistence /dev/sdb1
# mount -t ext4 /dev/sdb1 /mnt
# echo "/home" >> /mnt/persistence.conf
# echo "/var/cache/apt" >> /mnt/persistence.conf
# umount /mnt
```

Puis nous redémarrons. Lors du premier démarrage, les contenus du /home et /var/cache/apt seront copiés dans le volume persistant. À partir de ce moment, tous les changements dans ces répertoires seront stockés dans le volume persistant. Veuillez remarquer que les chemins répertoriés dans le fichier persistence.conf ne peuvent pas contenir d'espaces ou d'éléments spéciaux . et ... En outre, ni /lib, /lib/live (ou un de leurs sous-répertoires), ni / ne peuvent être rendus persistants en utilisant des montages personnalisés. Comme solution à

cette limitation, vous pouvez ajouter / union à votre fichier `persistence.conf` pour obtenir une persistance complète.

10.3.2 Utilisation de plusieurs dispositifs de persistance

Il existe différentes méthodes d'utilisation de multiples dispositifs de persistance pour les différents cas d'utilisation. Par exemple, utiliser plusieurs dispositifs à la fois ou en sélectionner un seul, entre plusieurs, à des fins très spécifiques.

Plusieurs volumes overlays différents (avec leurs propres fichiers `persistence.conf`) peuvent être utilisés au même temps, mais si plusieurs volumes rendent le même répertoire persistant, un seul d'entre eux sera utilisé. Si les deux sont « imbriqués » (un est un sous-répertoire de l'autre) le premier sera monté avant le second de sorte qu'aucun ne sera caché par l'autre. Monter des éléments personnalisés imbriqués est problématique s'ils sont énumérés dans le même fichier `persistence.conf`. Voir la page de manuel `persistence.conf(5)` pour savoir comment gérer ce cas si vous en avez vraiment besoin (remarque : ce n'est généralement pas le cas).

Un cas d'utilisation possible : Si vous souhaitez stocker les données de l'utilisateur, c'est-à-dire `/home` et les données du superutilisateur, c'est-à-dire `/root` dans des partitions différentes, créer deux partitions avec l'étiquette `persistence` et ajouter un fichier `persistence.conf` dans chacun comme ça

```
# echo "/home" > persistence.conf pour la première partition qui permettra de sauver les fichiers de l'utilisateur et # echo "/root" > persistence.conf pour la seconde partition qui permettra de stocker les fichiers du superutilisateur. Enfin, utiliser le paramètre d'amorçage persistence.
```

Si un utilisateur a besoin de stockages persistants multiples du même type pour différents endroits ou essais, tel que `private`

et `work`, le paramètre de démarrage `persistence-label` utilisé en conjonction avec le paramètre de démarrage `persistence` permettra de multiples mais uniques supports persistants. Dans le cas où un utilisateur voudrait utiliser une partition persistante étiquetée `private`, pour des données personnelles comme les marque-pages du navigateur ou d'autres types, il utiliserait les paramètres de démarrage : `persistence persistence-label=private`. Et pour stocker des données liées au travail, comme des documents, des projets de recherche ou d'autres types, il utiliserait les paramètres de démarrage : `persistence persistence-label=work`.

Il est important de se rappeler que chacun de ces volumes, `private` et `work`, a également besoin d'un fichier `persistence.conf` dans sa racine. La page de manuel *live-boot* contient plus d'informations sur la façon d'utiliser ces étiquettes avec des noms anciens.

10.4 Utilisation de la persistance avec chiffrement

Utiliser la persistance signifie que certaines données sensibles peuvent être exposées au risque. Surtout si les données persistantes sont stockées sur un dispositif portable tel qu'une clé USB ou un disque dur externe. C'est quand le chiffrement est plus pratique. Même si la procédure complète peut paraître compliquée en raison du nombre d'étapes à suivre, il est vraiment facile de manipuler des partitions chiffrées avec *live-boot*. Pour utiliser **luks**, qui est le type de chiffrement pris en charge, vous devez installer *cryptsetup* tant sur la machine avec laquelle vous créez la partition chiffrée et aussi dans le système live avec lequel vous allez utiliser la partition persistante chiffrée.

Pour installer *cryptsetup* sur votre machine :

```
# apt-get install cryptsetup
```

Pour installer *cryptsetup* dans votre système live, ajouter à vos listes de paquets :

```
$ lb config
$ echo "cryptsetup" > config/package-lists/encryption.list.chroot
```

Une fois que vous avez votre système live avec *cryptsetup*, vous avez essentiellement besoin de créer une nouvelle partition, la chiffrer et démarrer avec les paramètres *persistence* et *persistence-encryption=luks*. Nous aurions pu déjà anticipée cette étape et ajoutée ces paramètres de démarrage selon la procédure habituelle :

```
$ lb config --bootappend-live "boot=live components persistence persistence↵
-encryption=luks"
```

Allons dans les détails pour tous ceux qui ne connaissent pas bien le chiffrement. Dans l'exemple suivant, nous allons utiliser une partition sur une clé usb qui correspond au dispositif */dev/sdc2*. S'il vous plaît être averti que vous devez déterminer quelle partition est celui que vous allez utiliser dans votre cas particulier.

La première étape est de brancher votre clé usb et de déterminer quel dispositif il est. La méthode recommandée pour lister les dispositifs dans *live-manual* est utiliser *ls -l /dev/disk/by-id*. Après cela, créer une nouvelle partition et la chiffrer avec un mot de passe comme suit :

```
# cryptsetup --verify-passphrase luksFormat /dev/sdc2
```

Ensuite, ouvrir la partition luks dans le mappeur de dispositifs virtuel. Utilisez n'importe quel nom que vous aimez. Nous utilisons **live** ici à titre d'exemple :

```
# cryptsetup luksOpen /dev/sdc2 live
```

La prochaine étape est de remplir le dispositif avec des zéros avant de créer le système de fichiers :

```
# dd if=/dev/zero of=/dev/mapper/live
```

Maintenant, nous sommes prêts à créer le système de fichiers. Notez que nous ajoutons l'étiquette *persistence* de sorte que le dispositif est monté en tant que magasin de persistance au moment du démarrage.

```
# mkfs.ext4 -L persistence /dev/mapper/live
```

Pour continuer avec notre configuration, nous avons besoin de monter le dispositif, par exemple dans */mnt*.

```
# mount /dev/mapper/live /mnt
```

Et créer le fichier *persistence.conf* à la racine de la partition. Ceci

est, comme expliqué précédemment, strictement nécessaire. Voir
< [Le fichier persistence.conf](#) >.

609

```
# echo "/" union" > /mnt/persistence.conf
```

610 Puis, démontez le point de montage :

611

```
# umount /mnt
```

612 Et éventuellement, bien qu'il pourrait être un bon moyen de
sécuriser les données que nous venons d'ajouter à la partition,
nous pouvons fermer le dispositif :

613

```
# cryptsetup luksClose live
```

614 Résumons la procédure. Jusqu'ici nous avons créé un système
capable d'utiliser le chiffrement, qui peut être copié sur une clé usb
comme expliqué dans < [Copie d'une image ISO hybride sur une
clé USB](#) >. Nous avons également créé une partition chiffrée, qui
peut être située dans la même clé USB pour le porter autour et
nous avons configuré la partition chiffrée pour être utilisée comme
magasin de persistance. Alors maintenant, nous avons seulement
besoin de démarrer le système live. Au moment du démarrage,
live-boot nous demandera le mot de passe pour monter la partition
chiffrée à être utilisé pour la persistance.

Personnalisation de l'image binaire

11. Personnalisation de l'image binaire

11.1 Chargeurs d'amorçage

live-build utilise *syslinux* et certains de ses dérivés (selon le type d'image) comme chargeurs d'amorçage par défaut. Vous pouvez facilement les personnaliser pour répondre à vos besoins.

Pour utiliser un thème complet, copiez `/usr/share/live/build/-bootloaders` dans `config/bootloaders` et modifiez les fichiers là. Si vous ne voulez pas modifier toutes les configurations du chargeur d'amorçage prises en charge, il suffit de fournir une copie locale personnalisée d'un des chargeurs, par exemple copiez la configuration d'**isolinux** dans `config/bootloaders/isolinux`, selon votre cas d'utilisation.

Lors de la modification d'un des thèmes par défaut, si vous souhaitez utiliser une image de fond personnalisée qui sera affichée avec le menu de démarrage, ajouter une image `splash.png` de 640x480 pixels. Ensuite, supprimer le fichier `splash.svg`.

Il y a beaucoup de possibilités quand il s'agit de faire des changements. Par exemple, les dérivés de *syslinux* sont configurés par défaut avec un timeout de 0 (zéro), ce qui signifie qu'ils se mettront en pause indéfiniment à leur écran de démarrage jusqu'à ce que vous pressiez une touche.

Pour modifier le délai de démarrage d'une image *iso-hybrid*, vous pouvez modifier un fichier **isolinux.cfg** en précisant le timeout dans les unités de 1/10 secondes. Un **isolinux.cfg** modifié pour démarrer cinq secondes plus tard serait semblable à ceci :

```
include menu.cfg
default vesamenu.c32
prompt 0
timeout 50
```

11.2 Métadonnées ISO

En créant une image binaire ISO9660, vous pouvez utiliser les options suivantes pour ajouter différentes métadonnées textuelles pour votre image. Cela peut vous aider à facilement identifier la version ou la configuration d'une image sans la démarrer.

- **LB_ISO_APPLICATION/--iso-application NAME** : Cela devrait décrire l'application qui sera sur l'image. Le nombre maximum de caractères pour ce champ est 128.
- **LB_ISO_PREPARER/--iso-preparer NAME** : Cela devrait décrire le préparateur de l'image, généralement avec certains détails de contact. Le défaut de cette option est la version de *live-build* que vous utilisez, ce qui peut faciliter le débogage plus tard. Le nombre maximum de caractères pour ce champ est 128.
- **LB_ISO_PUBLISHER/--iso-publisher NAME** : Ce doit décrire l'éditeur de l'image, généralement avec certains détails de contact. Le nombre maximum de caractères pour ce champ est 128.
- **LB_ISO_VOLUME/--iso-volume NAME** : Cela devrait indiquer l'ID de volume de l'image. Il est utilisé comme une étiquette visible par l'utilisateur sur certaines plateformes comme Windows et Apple Mac OS. Le nombre maximum de caractères pour ce champ est 128.

Personnalisation de l'installateur Debian

12. Personnalisation du contenu pour l'installateur Debian

Les images des systèmes live peuvent être intégrées avec l'installateur Debian. Il y a un certain nombre de types d'installation différents, variant en fonction de ce qui est inclus et de la façon dont fonctionne l'installateur.

Veuillez noter l'utilisation prudente des lettres majuscules pour désigner « l'Installateur Debian » dans cette section - lorsqu'il est utilisé comme cela, nous faisons explicitement référence à l'installateur officiel pour le système Debian. On le voit souvent abrégé en « d-i ».

12.1 Types d'installateur Debian

Les trois principaux types de programme d'installation sont :

Installateur Debian « Normal » : C'est une image du système live avec un noyau et initrd séparés qui (lorsqu'ils sont sélectionnés à partir du chargeur d'amorçage approprié) se lancent dans une instance d'installateur Debian standard, exactement comme si vous aviez téléchargé une image de CD de Debian et l'aviez démarrée. Les images contenant un système live et un installateur indépendant sont souvent appelées « images combinées ».

Sur ces images, Debian est installé par l'extraction et l'installation de paquets .deb à l'aide de *debootstrap*, à partir des supports locaux ou du réseau, résultant en un système Debian par défaut installé sur le disque dur.

Tout ce processus peut être préconfiguré et personnalisé d'un certain nombre de façons. Consultez les pages correspondantes

dans le manuel de l'Installateur Debian pour plus d'informations. Une fois que vous avez un fichier de préconfiguration qui fonctionne, *live-build* peut automatiquement l'ajouter à l'image et l'activer pour vous.

Installateur Debian “Live” : C'est une image du système live avec un noyau et initrd séparés qui (lorsqu'ils sont sélectionnés à partir du chargeur d'amorçage approprié) se lancent dans une instance de l'installateur Debian.

L'installation continue de manière identique à l'installation « normale » décrite ci-dessus, mais à l'étape de l'installation des paquets, au lieu d'utiliser *debootstrap* pour aller chercher et installer des paquets, l'image du système de fichiers live est copiée vers la cible. Ce résultat est obtenu avec un udeb spécial appelé *live-installer*.

Après cette étape, l'installateur Debian continue normalement, en installant et configurant des éléments tels que les chargeurs d'amorçage et les utilisateurs locaux, etc.

Remarque : Pour prendre en charge les deux options – installateur normal et live – dans le chargeur d'amorçage du même support live, vous devez désactiver *live-installer* en utilisant la préconfiguration `live-installer/enable=false`.

Installateur Debian “de bureau” : Indépendamment du type d'installateur Debian inclus, *d-i* peut être lancé à partir du bureau en cliquant sur une icône, ce qui est plus facile à utiliser dans certaines situations. Pour pouvoir en faire usage, le paquet *debian-installer-launcher* doit être inclus.

Notez que, par défaut, *live-build* n'inclut pas les images de l'installateur Debian dans les images, il doit être spécifiquement activé avec `lb config`. De même, veuillez noter que pour que l'installateur “de bureau” fonctionne, le noyau du système live doit correspondre au noyau que *d-i* utilise pour l'architecture indiquée.

Par exemple :

```
$ lb config --architectures i386 --linux-flavours 486 \
    --debian-installer live
$ echo debian-installer-launcher >> config/package-lists/my.list.chroot
```

des paquets udeb d-i construits localement. Placez-les dans `config/packages.binary/` pour les inclure dans l'image. Plusieurs fichiers supplémentaires ou de remplacement et plusieurs répertoires peuvent aussi être inclus dans l'initrd de l'installateur, d'une manière similaire à `< Live/chroot local includes >` en plaçant le contenu dans `config/includes.installer/`.

12.2 Personnalisation de l'installateur Debian par préconfiguration

Comme décrit dans le manuel de Debian Installer, appendice B sur <https://www.debian.org/releases/stable/i386/apb.html>, « La préconfiguration est une façon de donner des réponses aux questions posées pendant le processus d'installation, sans avoir à entrer manuellement les réponses alors que l'installation est en marche. Cela permet d'automatiser entièrement la plupart des types d'installation et elle offre certaines fonctionnalités que ne sont pas disponibles pendant les installations normales ». Ce type de personnalisation se fait mieux avec *live-build* en plaçant la configuration dans un fichier `preseed.cfg` inclus dans `config/includes.installer/`. Par exemple, pour préconfigurer les paramètres régionaux pour `en_US` :

```
$ echo "d-i debian-installer/locale string en_US" \
    >> config/includes.installer/preseed.cfg
```

12.3 Personnalisation de contenu pour l'Installateur Debian

À des fins expérimentales ou de débogage, vous pouvez inclure

651

Projet

Contribuer au projet

13. Contribuer au projet

Lorsque vous soumettez une contribution, veuillez indiquer clairement le copyright et inclure toute mention légale relative à la licence. Notez que pour être acceptée, la contribution doit être déposée sous la même licence que le reste du document, c'est-à-dire la GPL version 3 ou ultérieure.

Les contributions au projet, comme les traductions et les correctifs, sont bienvenues. Les livraisons au dépôt sont possibles pour tout le monde, cependant, nous vous demandons d'envoyer les changements importants sur la liste de diffusion au préalable. Veuillez consulter < [Contacter](#) > pour plus d'informations.

Le Live Systems Project utilise Git en tant que système de contrôle de version et de gestion du code source. Comme expliqué dans < [Dépôts Git](#) > il existe deux branches de développement principales : **debian** et **debian-next**. Tout le monde peut faire des livraisons aux branches *debian-next* des dépôts *live-boot*, *live-build*, *live-config*, *live-images*, *live-manual* et *live-tools*.

Cependant, il y a certaines restrictions. Le serveur rejette :

- Push non fast-forward.
- Commits merge.
- Ajout ou suppression d'étiquettes ou des branches.

Même si toutes les livraisons pourraient être révisées, nous vous demandons d'utiliser votre bon sens et de faire bonnes livraisons avec de bons messages de livraison.

- Veuillez écrire les commentaires de livraison à l'aide de phrases complètes, en commençant par une majuscule et en terminant par un point, et avec la forme 'Fixing/Adding/Removing/Correcting/Translating/...'.

- Écrivez de bons messages de livraison. La première ligne doit être un résumé précis du contenu du commit qui sera inclus dans le changelog. Si vous avez besoin de faire quelques explications supplémentaires, écrivez-les au-dessous en laissant une ligne vide après la première, puis une autre ligne vide après chaque paragraphe. Les lignes des paragraphes ne doivent pas dépasser 80 caractères.
- Faites des livraisons de façon atomique, c'est-à-dire, ne mélangez pas des choses sans liens entre elles dans la même livraison. Faites un commit différent pour chaque modification que vous apportez.

13.1 Faire des changements

Afin de faire un push sur les dépôts, vous devez suivre la procédure suivante. Ici, nous utilisons *live-manual* comme un exemple pour le remplacer par le nom du dépôt dans lequel vous souhaitez travailler. Pour plus d'informations sur la façon de modifier *live-manual*, consultez < [Contribuer à ce document](#) >.

- Téléchargez la clé publique :

```
$ mkdir -p ~/.ssh/keys
$ wget http://live-systems.org/other/keys/git@live-systems.org -O ~/.ssh/keys/git@live-systems.org
$ wget http://live-systems.org/other/keys/git@live-systems.org.pub -O ~/.ssh/keys/git@live-systems.org.pub
$ chmod 0600 ~/.ssh/keys/git@live-systems.org*
```

- Ajoutez la section suivante à votre configuration openssh-client :

```
$ cat >> ~/.ssh/config << EOF
```

```
Host live-systems.org
  Hostname live-systems.org
  User git
  IdentitiesOnly yes
  IdentityFile ~/.ssh/keys/git@live-systems.org
EOF
```

```
$ git push
```

- Clonez *live-manual* avec ssh :

```
$ git clone git@live-systems.org :live-manual.git
$ cd live-manual && git checkout debian-next
```

- Assurez-vous d'avoir renseignés les champs d'auteur et d'email dans Git :

```
$ git config user.name "John Doe"
$ git config user.email john@example.org
```

Important : Rappelez-vous que vous devez livrer les changements sur la branche **debian-next** .

- Effectuez vos modifications. Dans cet exemple, vous devriez commencer par écrire une nouvelle section traitant l'application des correctifs et ensuite préparer l'ajout des fichiers et écrire le message de livraison comme ceci :

```
$ git commit -a -m "Adding a section on applying patches."
```

- Envoyez votre commit au serveur :

680 Signaler des bogues

681 14. Signaler des bogues

682 Les systèmes live sont loin d'être parfaits, mais nous voulons les rendre aussi parfaits que possible – avec votre aide. N'hésitez pas à signaler un bogue. Il est préférable de remplir un rapport deux fois plus que jamais. Toutefois, ce chapitre contient des recommandations pour présenter de bons rapports de bogues.

683 Pour les impatientes :

- 684 • Commencez toujours par vérifier les mises à jour du statut de l'image sur notre page d'accueil < <http://live-systems.org/> > pour voir les problèmes connus.
- 685 • Avant de présenter un rapport de bogue, toujours essayer de reproduire le bogue avec **les versions les plus récentes** de la branche de *live-build*, *live-boot*, *live-config* et *live-tools* que vous utilisez (comme la dernière version 4.x de *live-build* si vous utilisez *live-build* 4).
- 686 • Essayez de donner des informations aussi précises que possible sur le bogue. Cela comprend (au moins) la version de *live-build*, *live-boot*, *live-config* et *live-tools*, de la distribution utilisée et du système live que vous construisez.

687 14.1 Problèmes connus

688 Puisque les distributions Debian **testing** et Debian **unstable** sont des cibles mouvantes, quand vous les indiquez comme distributions du système cible, une construction avec succès n'est pas toujours possible.

689 Si cela vous pose trop de difficulté, ne construisez pas un système

basé sur **testing** ou **unstable**, mais utilisez plutôt **stable**. *live-build* utilise toujours la version **stable** par défaut.

Les problèmes connus sont énumérés dans la section « statut » sur notre page < <http://live-systems.org/> >.

Le sujet de ce manuel n'est pas de vous former à identifier et corriger correctement les problèmes dans les paquets des distributions de développement. Cependant, il y a deux choses que vous pouvez toujours essayer : Si une construction échoue lorsque la distribution cible est **testing**, essayez **unstable**. Si **unstable** ne fonctionne pas non plus, revenez à **testing** et fixez la nouvelle version du paquet qui échoue de **unstable** (voir < [APT pinning](#) > pour plus de détails).

692 14.2 Reconstruire à partir de zéro

Afin de vous assurer qu'un bogue en particulier n'est pas causé par un système mal construit, veuillez toujours reconstruire l'ensemble du système live à partir de zéro pour voir si le bogue est reproductible.

694 14.3 Utiliser des paquets mis à jour

L'utilisation de paquets obsolètes peut causer des problèmes importants en essayant de reproduire (et finalement régler) votre problème. Assurez-vous que votre système de construction est mis à jour et tous les paquets inclus dans votre image sont mis à jour aussi.

696 14.4 Recueillir l'information

Veuillez fournir assez d'informations avec votre rapport. Incluez au

moins la version exacte de *live-build* où le bogue est rencontré et les mesures pour le reproduire. Veuillez utiliser votre bon sens et incluez d'autres renseignements pertinents, si vous pensez que cela pourrait aider à résoudre le problème.

Pour tirer le meilleur parti de votre rapport de bogue, nous avons au moins besoin des informations suivantes :

- L'architecture du système hôte
- Distribution du système hôte
- Version de *live-build* sur le système hôte
- Version de Python sur le système hôte
- Version de *debootstrap* et/ou *cdebootstrap* sur le système hôte
- L'architecture du système live
- Répartition du système live
- Version de *live-boot* sur le système live
- Version de *live-config* sur le système live
- Version de *live-tools* sur le système live

Vous pouvez générer un journal du processus de construction en utilisant la commande *tee*. Nous recommandons de faire cela automatiquement avec un script *auto/build* (voir < [Gestion d'une configuration](#) > pour les détails).

```
# lb build 2>&1 | tee build.log
```

Au démarrage, *live-boot* et *live-config* stockent un journal dans */var/log/live/boot.log*. Vérifiez-les pour des messages d'erreur.

Par ailleurs, pour écarter d'autres erreurs, c'est toujours une

bonne idée de faire un tar de votre répertoire *config/* et de le télécharger quelque part (**ne pas** l'envoyer en pièce jointe à la liste de diffusion), de sorte que nous puissions essayer de reproduire les erreurs que vous rencontrez. Si cela est difficile (en raison par exemple de la taille) vous pouvez utiliser la sortie de *lb config --dump* qui produit un résumé de votre arbre de config (c'est-à-dire les listes des fichiers dans les sous-répertoires de *config/* mais ne les inclut pas).

N'oubliez pas d'envoyer tous les journaux produits avec les paramètres régionaux anglais. Par exemple, exécutez vos commandes *live-build* précédées par *LC_ALL=C* ou *LC_ALL=en_US*.

14.5 Isoler le cas qui échoue, si possible

Si possible, isolez le cas qui échoue au plus petit changement possible. Il n'est pas toujours facile de faire cela, donc si vous ne pouvez pas le gérer pour votre rapport, ne vous inquiétez pas. Toutefois, si vous planifiez bien votre cycle de développement, en utilisant de petits ensembles de changements par itération, vous pourriez être capable d'isoler le problème en construisant une configuration simple « base » qui correspond étroitement à la configuration réelle avec seulement le changement cassé ajouté. S'il est difficile de trier vos modifications qui cassent, il est possible que vous incluez trop dans chaque ensemble de modifications et vous devriez développer en petits incréments.

14.6 Utiliser le paquet adéquat pour rapporter un bogue

Si vous ne savez pas quel composant est responsable du bogue ou si le bogue est un bogue général concernant les systèmes live,

vous pouvez remplir un rapport de bogue sur le pseudo-paquet debian-live.

718 Toutefois, nous apprécierions que vous essayiez de le réduire en fonction de l'endroit où le bogue apparaît.

719 14.6.1 Pendant la construction durant l'amorçage

720 *live-build* amorce d'abord un système Debian de base avec *debootstrap* ou *cdebootstrap*. Selon l'outil d'amorçage utilisé et de la distribution Debian, il peut échouer. Si un bogue apparaît ici, vérifiez si l'erreur est liée à un paquet Debian spécifique (plus probable), ou si elle est liée à l'outil d'amorçage lui-même.

721 Dans les deux cas, ce n'est pas un bogue dans le système live, mais plutôt dans Debian lui-même que probablement nous ne pouvons pas le résoudre directement. Veuillez signaler un bogue sur l'outil d'amorçage ou du paquet défaillant.

722 14.6.2 Pendant la construction durant l'installation de paquets

723 *live-build* installe des paquets supplémentaires de l'archive Debian et en fonction de la distribution Debian utilisée et l'état quotidien de l'archive, il peut échouer. Si un bogue apparaît ici, vérifiez si l'erreur est également reproductible sur un système normal.

724 Si c'est le cas, ce n'est pas un bogue dans le système live, mais plutôt dans Debian – veuillez envoyer le rapport sur le paquet défaillant. L'exécution de *debootstrap* séparément du système de construction ou l'exécution de `lb bootstrap --debug` vous donnera plus d'informations.

725 Aussi, si vous utilisez un miroir local et/ou un proxy et vous

rencontrez un problème, veuillez toujours le reproduire en amorçant d'abord sur un miroir officiel.

14.6.3 Pendant le démarrage

Si votre image ne démarre pas, veuillez le signaler à la liste de diffusion avec les informations demandées dans < [Recueillir l'information](#) >. N'oubliez pas de mentionner, comment/quand l'image a échoué, soit en virtualisation ou sur du matériel réel. Si vous utilisez une technologie de virtualisation de quelconque sorte, veuillez toujours tester sur du matériel réel avant de signaler un bogue. Fournir une copie d'écran de l'échec est également très utile.

14.6.4 Pendant l'exécution

Si un paquet a été installé avec succès, mais qu'il échoue lors de l'exécution du système Live, il s'agit probablement d'un bogue dans le système live. Cependant :

14.7 Effectuer une recherche

Avant de présenter le bogue, veuillez rechercher sur le web le message d'erreur ou un symptôme particulier que vous obtenez. Comme il est hautement improbable que vous soyez la seule personne faisant l'expérience d'un problème particulier, il y a toujours une chance qu'il ait été discuté ailleurs, et qu'une solution possible, un correctif, ou une solution de contournement ait été proposés.

Vous devez prêter une attention particulière à la liste de diffusion du système live, ainsi qu'à la page d'accueil, car elles sont susceptibles de contenir des informations à jour. Si ces

informations existent, incluez toujours les références au sein de vos rapports de bogues.

733 En outre, vous devriez vérifier les listes de bogues en cours de *live-build*, *live-boot*, *live-config* et *live-tools* pour voir si quelque chose de semblable n'a pas déjà été signalée.

734 14.8 Où rapporter les bogues

735 Le Live Systems Project conserve la trace de tous les bogues dans le système de suivi des bogues (BTS). Pour plus d'informations sur la façon d'utiliser le système, veuillez consulter < <https://bugs.debian.org/> >. Vous pouvez également soumettre les bogues en utilisant la commande `reportbug` du paquet du même nom.

736 En général, vous devez signaler les erreurs de construction contre le paquet *live-build*, les erreurs lors du démarrage contre *live-boot*, et les erreurs d'exécution contre *live-config*. Si vous n'êtes pas sûr du paquet approprié ou si vous avez besoin d'aide avant de soumettre un rapport de bogue, veuillez signaler le bogue contre le pseudo-paquet *debian-live*. Nous le réattribuerons s'il y a lieu.

737 Veuillez noter que les bogues trouvés dans les distributions dérivées de Debian (comme Ubuntu et autres) **ne** doivent **pas** être rapportés au BTS de Debian, sauf s'ils peuvent être également reproduits sur un système Debian en utilisant les paquets Debian officiels.

738 **Style de code**

739 **15. Style du code**

740 Ce chapitre documente le style du code utilisé dans les systèmes
live.

741 **15.1 Compatibilité**

- 742 • N'utilisez pas une syntaxe ou sémantique qui soit unique au shell
Bash. Par exemple, l'utilisation de tableaux (arrays).
- 743 • N'utilisez que le sous-ensemble POSIX – par exemple, utilisez
\$(foo) au lieu de 'foo'.
- 744 • Vous pouvez vérifier vos scripts avec 'sh -n' et 'checkbashisms'.
- 745 • Assurez-vous que tout le code fonctionne avec 'set-e '.

746 **15.2 Indentation**

- 747 • Utilisez toujours des tabulations au lieu des espaces.

748 **15.3 Adaptateur**

- 749 • Généralement, les lignes sont de 80 caractères au maximum.
- 750 • Utilisez le « style Linux » des sauts de ligne :

751 Mal :

752

```
if foo; then
    bar
fi
```

753

Bien :

754

```
if foo
then
    bar
fi
```

- La même chose vaut pour les fonctions :

755

Mal :

756

757

```
Foo () {
    bar
}
```

Bien :

758

759

```
Foo ()
{
    bar
}
```

15.4 Variables

760

- Les variables sont toujours en lettres majuscules. 761
- Les variables utilisées dans *live-build* commencent toujours par
le préfixe LB_. 762
- Les variables temporaires internes dans *live-build* devraient
commencer avec le préfixe <=underscore>LB_. 763
- Les variables locales commencent avec le préfixe <=underscore><=underscore>L 764

- Les variables en relation avec un paramètre de démarrage dans *live-config* commencent par `LIVE_`.
- Toutes les autres variables dans *live-config* commencent par le préfixe `_`.
- Utilisez des accolades autour des variables ; écrivez par exemple `${F00}` au lieu de `$F00`.
- Protégez toujours les variables avec des guillemets pour respecter les espaces potentiels : écrire `"${F00}"` en lieu de `${F00}`.
- Pour des raisons de cohérence, utilisez toujours les guillemets lors de l'attribution des valeurs aux variables :

Mal :

```
F00=bar
```

Bien :

```
F00="bar"
```

- Si plusieurs variables sont utilisées, utilisez les guillemets pour l'expression complète :

Mal :

```
if [ -f "${F00}/foo/${BAR}/bar ]
then
    foobar
fi
```

Bien :

```
if [ -f "${F00}/foo/${BAR}/bar" ]
then
    foobar
fi
```

15.5 Autres

- Utilisez `|` (sans les guillemets autour) comme séparateur dans les appels à `sed`, par exemple `sed -e `s|'|`` (sans `"`).
- N'utilisez pas la commande `test` pour des comparaisons ou des tests, utilisez `[` `]` (sans `"`); par exemple `if [-x /bin/foo]; ...` et non pas `if test -x /bin/foo; ...`.
- Utilisez `case` dans la mesure du possible au lieu de `test`, parce qu'il est plus facile à lire et plus rapide à exécuter.
- Utilisez des noms en majuscule pour les fonctions pour éviter toute interférence avec l'environnement des utilisateurs.

784 Procédures

785 16. Procédures

786 Ce chapitre documente les procédures au sein du Live Systems Project pour différentes tâches qui ont besoin de coopération avec d'autres équipes dans Debian.

787 16.1 Évolutions majeures

788 La libération d'une nouvelle version stable majeure de Debian inclut un grand nombre de différentes équipes travaillant ensemble. À un certain point, l'équipe Live arrive et construit des images du système live. Les conditions pour ce faire sont les suivantes :

- 789 • Un miroir contenant les versions publiées des archives Debian et debian-security auxquelles le build de debian-live peut avoir accès.
- 790 • Les noms de l'image doivent être connus (par exemple debian-live-VERSION-ARCH-FLAVOUR.iso).
- 791 • Les données qui proviennent de debian-cd doivent être synchronisées (udeb exclude lists).
- 792 • Les images sont construites et mises en miroir sur cdimage.debian.org.

793 16.2 Évolutions mineures

- 794 • Encore une fois, nous avons besoin de miroirs de Debian et Debian-security mis à jour.
- 795 • Les images sont construites et mises en miroir sur cdimage.debian.org.
- 796 • Envoyer un courriel d'annonce.

797 16.2.1 Dernière évolution mineure d'une version Debian

N'oubliez pas de régler à la fois les miroirs pour chroot et binary lors de la construction de la dernière série d'images pour une version de Debian après qu'elle ait été déplacée de ftp.debian.org vers archive.debian.org. De cette façon, les vieilles images live précompilées sont encore utiles, sans modifications des utilisateurs.

799 16.2.2 Modèle pour l'annonce d'une évolution mineure

Un courriel pour l'annonce d'une évolution mineure peut être généré en utilisant le modèle ci-dessous et la commande suivante :

```
$ sed \
-e 's|@MAJOR@|7.0|g' \
-e 's|@MINOR@|7.0.1|g' \
-e 's|@CODENAME@|wheezy|g' \
-e 's|@ANNOUNCE@|2013/msgXXXXX.html|g'
```

Veillez vérifier le courriel avant l'envoi et le passer à d'autres pour la relecture.

```
Updated Live @MAJOR@ : @MINOR@ released

The Live Systems Project is pleased to announce the @MINOR@ update of the
Live images for the stable distribution Debian @MAJOR@ (codename "<←
@CODENAME@").

The images are available for download at :

<http://live-systems.org/cdimage/release/current/>
```

and later at :

<<http://cdimage.debian.org/cdimage/release/current-live/>>

This update includes the changes of the Debian @MINOR@ release :

<<https://lists.debian.org/debian-announce/@ANNOUNCE@>>

Additionally it includes the following Live-specific changes :

- * [INSERT LIVE-SPECIFIC CHANGE HERE]
- * [INSERT LIVE-SPECIFIC CHANGE HERE]
- * [LARGER ISSUES MAY DESERVE THEIR OWN SECTION]

About Live Systems

The Live Systems Project produces the tools used to build official live systems and the official live images themselves for Debian.

About Debian

The Debian Project is an association of Free Software developers who volunteer their time and effort in order to produce the completely free operating system Debian.

Contact Information

For further information, please visit the Live Systems web pages at <<http://live-systems.org/>>, or contact the Live Systems team at <debian-live@lists.debian.org>.

Dépôts Git

17. Dépôts Git

La liste de tous les dépôts disponibles du Live Systems Project est sur < <http://live-systems.org/gitweb/> >. Les URLs git du projet ont la forme : protocole `://live-systems.org/git/dépôt`. Ainsi, afin de cloner *live-manual* en lecture seule, lancer :

```
$ git clone git ://live-systems.org/git/live-manual.git
```

Ou,

```
$ git clone https ://live-systems.org/git/live-manual.git
```

Ou,

```
$ git clone http ://live-systems.org/git/live-manual.git
```

Les adresses pour cloner avec autorisation d'écriture ont la forme : `git@live-systems.org ://dépôt`.

Donc, encore une fois, pour cloner *live-manual* sur ssh, vous devez taper :

```
$ git clone git@live-systems.org live-manual.git
```

L'arbre git est composé de plusieurs branches différentes.

Les branches **debian** et **debian-next** sont particulièrement remarquables car elles contiennent le travail réel qui sera finalement inclus dans chaque nouvelle version.

Après avoir cloné quelques-uns des dépôts existants, vous serez sur la branche **debian**. Ceci est approprié pour jeter un œil à l'état de la dernière version du projet, mais avant de commencer le travail, il est essentiel de passer à la branche **debian-next**. Pour ce faire :

```
$ git checkout debian-next
```

La branche **debian-next**, qui n'est pas toujours fast-forward, est l'endroit où toutes les modifications sont envoyées en premier, avant de les fusionner dans la branche **debian**. Pour faire une analogie, c'est comme un terrain d'essai. Si vous travaillez sur cette branche et avez besoin de faire un pull, vous devrez faire un `git pull --rebase` de sorte que vos modifications locales sont stockées tandis qu'on fait une mise à jour à partir du serveur, puis vos changements seront mis au-dessus de tout cela.

17.1 Gestion de multiples dépôts

Si vous avez l'intention de cloner plusieurs des dépôts Live Systems et passer à la branche **debian-next** tout de suite pour vérifier le dernier code, écrire un correctif ou contribuer avec une traduction, vous devriez savoir que le serveur git fournit un fichier `mrconfig` pour faciliter la manipulation de multiples dépôts. Pour l'utiliser, vous devez installer le paquet *mr*, puis lancer :

```
$ mr bootstrap http ://live-systems.org/other/mr/mrconfig
```

822

Cette commande va automatiquement cloner et passer à la branche **debian-next** les dépôts de développement des paquets Debian produits par le projet. Ceux-ci comprennent, entre autres, le dépôt *live-images*, qui contient les configurations utilisées pour les images précompilées que le projet publie pour un usage général. Pour plus d'informations sur la façon d'utiliser ce dépôt, voir < [Cloner une configuration publiée via Git](#) >

Exemples

Exemples

18. Exemples

Ce chapitre présente des exemples de constructions pour des cas d'utilisation spécifiques avec des systèmes live. Si vous débutez dans la construction de vos propres images des systèmes live, nous vous recommandons de regarder d'abord les trois tutoriels à la suite car chacun d'entre eux enseigne de nouvelles techniques qui vous aideront à utiliser et à comprendre les exemples restants.

18.1 Utiliser les exemples

Pour utiliser ces exemples, vous avez besoin d'un système pour les construire qui doit répondre aux exigences énumérées dans < **Exigences** > et sur lequel *live-build* est installé comme décrit dans < **Installation de live-build** >.

Notez que, pour des raisons de concision, dans ces exemples, nous n'indiquons pas de miroir local à utiliser pour la construction. Vous pouvez accélérer considérablement les téléchargements si vous utilisez un miroir local. Vous pouvez indiquer ces options lorsque vous utilisez `lb config`, tel que décrit dans < **Miroirs de distribution utilisés pendant la construction** >, ou pour plus de commodité, fixez par défaut votre système de construction dans `/etc/live/build.conf`. Il suffit de créer ce fichier et de définir les variables `LB_MIRROR_*` correspondantes à votre miroir préféré. Tous les autres miroirs utilisés dans la construction seront choisis par défaut à partir de ces valeurs. Par exemple :

```
LB_MIRROR_BOOTSTRAP="http://mirror/debian/"
LB_MIRROR_CHROOT_SECURITY="http://mirror/debian-security/"
```

```
LB_MIRROR_CHROOT_BACKPORTS="http://mirror/debian-updates/"
```

18.2 Tutoriel 1 : Une image par défaut

Cas d'utilisation : Créer une image simple d'abord, en apprenant les bases de *live-build*.

Dans ce tutoriel, nous construirons une image ISO hybride par défaut contenant uniquement des paquets de base (pas de Xorg) et quelques paquets de prise en charge live, en guise de premier exercice dans l'utilisation de *live-build*.

Vous ne pouvez pas faire plus simple que cela :

```
$ mkdir tutorial1 ; cd tutorial1 ; lb config
```

Examinez le contenu du répertoire `config/` si vous le souhaitez. Vous verrez stockés ici une arborescence de configuration, prête à être personnalisée ou, dans ce cas, utilisée immédiatement pour construire une image par défaut.

Maintenant, en tant que superutilisateur, construisez l'image en enregistrant un journal avec `tee`.

```
# lb build 2>&1 | tee build.log
```

En supposant que tout se passe bien, après un certain temps, le répertoire courant contiendra `live-image-i386.hybrid.iso`. Cette image ISO hybride peut être démarrée directement dans une machine virtuelle comme décrit dans < **Test d'une image ISO avec QEMU** > et < **Test d'une image ISO avec VirtualBox** >.

ou bien copiée sur un support optique ou un périphérique USB comme décrit dans < [Graver une image ISO sur un support physique](#) > et < [Copie d'un image ISO hybride sur une clé USB](#) >, respectivement.

18.3 Tutoriel 2 : Un utilitaire d'un navigateur Web

Cas d'utilisation : Créer l'image d'un utilitaire de navigation Web, en apprenant à appliquer des personnalisations.

Dans ce tutoriel, nous allons créer une image utilisable comme un utilitaire de navigation web, ce qui servira d'introduction à la personnalisation d'images live.

```
$ mkdir tutorial2
$ cd tutorial2
$ lb config
$ echo "task-lxde-desktop iceweasel" >> config/package-lists/my.list.chroot
$ lb config
```

Notre choix de LXDE pour cet exemple reflète notre volonté de fournir un environnement de bureau minimal, puisque le but de l'image est l'utilisation unique que nous avons à l'esprit, le navigateur web. On pourrait aller encore plus loin et offrir une configuration par défaut pour le navigateur web dans config/includes.chroot/etc/iceweasel/profile/, ou des paquets de prise en charge supplémentaires pour visualiser différents types de contenu web, mais nous laissons cela en exercice pour le lecteur.

Construisez l'image, encore une fois en tant que superutilisateur, et gardez un journal comme dans < [Tutoriel 1](#) > :

```
# lb build 2>&1 | tee build.log
```

Encore une fois, vérifiez que l'image est OK et faites un test, comme dans < [Tutoriel 1](#) > :

18.4 Tutoriel 3 : Une image personnalisée

Cas d'utilisation : Créer un projet pour construire une image personnalisée, contenant vos logiciels préférés à emporter avec vous sur une clé USB où que vous alliez, et évoluant dans des révisions successives selon les changements de vos besoins et de vos préférences.

Puisque nous allons changer notre image personnalisée pendant un certain nombre de révisions, et que nous voulons suivre ces changements, essayer des choses expérimentalement et éventuellement les annuler si les choses ne fonctionnent pas, nous garderons notre configuration dans le populaire système de contrôle de version git. Nous allons également utiliser les meilleures pratiques d'autoconfiguration via auto scripts tel que décrit dans < [Gestion d'une configuration](#) >.

18.4.1 Première révision

```
$ mkdir -p tutorial3/auto
$ cp /usr/share/doc/live-build/examples/auto/* tutorial3/auto/
$ cd tutorial3
```

Éditez auto/config comme suit :

```
#!/bin/sh

lb config noauto \
  --architectures i386 \
  --linux-flavours 686-pae \
  "${@}"
```

862

Exécutez `lb config` pour générer l'arbre de configuration, en utilisant le script `auto/config` qu'on a créé :

```
$ lb config
```

Remplissez maintenant votre liste de paquets locaux :

```
$ echo "task-lxde-desktop iceweasel xchat" >> config/package-lists/my.list.<↵>
chroot
```

Tout d'abord, `--architectures i386` assure que sur notre système de construction amd64, nous construisons une version de 32 bits qui peut être utilisée sur la plupart des machines. Deuxièmement, nous utilisons `--linux-flavours 686-pae` parce que nous ne prévoyons pas d'utiliser cette image sur des systèmes très anciens. Troisièmement, nous avons choisi le méta-paquet de la tâche `lxde` pour nous donner un bureau minimal. Et enfin, nous avons ajouté deux premiers paquets préférés : `iceweasel` et `xchat`.

Maintenant, construisez l'image :

```
# lb build
```

Notez que contrairement aux deux premiers tutoriels, nous n'avons plus besoin de taper `2>&1 |tee build.log` parce que cela est maintenant inclus dans `auto/build`.

Une fois que vous avez testé l'image (comme dans < [Tutoriel 1](#) >) et vous êtes satisfait de son fonctionnement, il est temps d'initialiser notre dépôt git, en n'ajoutant que les scripts `auto` que nous avons juste créés, puis de faire le premier commit :

```
$ git init
$ cp /usr/share/doc/live-build/examples/gitignore .gitignore
$ git add .
$ git commit -m "Initial import."
```

18.4.2 Deuxième révision

Dans cette révision, nous allons nettoyer à partir de la première construction, ajouter le paquet `vlc` à notre configuration, reconstruire, tester et faire le commit.

La commande `lb clean` va nettoyer tous les fichiers générés par la construction précédente à l'exception du cache, ce qui évite d'avoir à retélécharger les paquets. Cela garantit que la commande `lb build` suivante exécutera à nouveau toutes les étapes pour régénérer les fichiers de notre nouvelle configuration.

```
# lb clean
```

Ajoutez maintenant le paquet `vlc` à votre liste de paquets locaux dans `config/package-lists/my.list.chroot` :

```
$ echo vlc >> config/package-lists/my.list.chroot
```

Construisez à nouveau :

```
# lb build
```

Testez et, quand vous êtes satisfait, faites le commit de la prochaine révision :

```
$ git commit -a -m "Ajout du lecteur de média vlc."
```

Bien sûr, des changements plus compliqués de la configuration sont possibles, comme l'ajout de fichiers dans les sous-répertoires de `config/`. Quand vous livrez de nouvelles révisions, prenez soin de ne pas modifier à la main ou d'envoyer dans le dépôt les fichiers de niveau supérieur dans `config` contenant les variables `LB_*`, car ce sont aussi des produits de la création et ils sont toujours nettoyés par `lb clean` et recréés avec `lb config` via leurs scripts `auto` respectifs.

Nous sommes arrivés à la fin de notre série de tutoriels. Alors que de nombreux types de personnalisations sont possibles, même en n'utilisant que les fonctionnalités explorées dans ces exemples simples, une variété presque infinie d'images différentes peut être créée. Les autres exemples de cette section couvrent plusieurs autres cas d'utilisation tirés des expériences recueillies chez les utilisateurs des systèmes live.

18.5 Un client kioske VNC

Cas d'utilisation : Créer une image avec *live-build* pour démarrer

directement sur un serveur VNC.

Créez un répertoire de construction et créez une configuration à l'intérieur, désactivez « recommends » pour faire un système minimal. Puis créez deux listes initiales de paquets : la première générée par un script fourni par *live-build* nommée *Packages* (voir < **Générer listes de paquets** >), et la seconde incluant *xorg*, *gdm3*, *metacity* et *xvnc4viewer*.

```
$ mkdir vnc-kiosk-client
$ cd vnc-kiosk-client
$ lb config -a i386 -k 686-pae --apt-recommends false
$ echo '! Packages Priority standard' > config/package-lists/standard.list.<↵
chroot
$ echo "xorg gdm3 metacity xvnc4viewer" > config/package-lists/my.list.<↵
chroot
```

Comme il est expliqué dans < **Régler APT pour économiser de l'espace** >, il se peut que vous deviez ajouter quelques paquets recommandés pour faire fonctionner votre image correctement.

Une façon facile d'énumérer les paquets recommandés est d'utiliser *apt-cache*. Par exemple :

```
$ apt-cache depends live-config live-boot
```

Dans cet exemple, nous avons découvert que nous devons ré-inclure plusieurs paquets recommandés par *live-config* et *live-boot* : *user-setup* pour l'autologin et *sudo* un logiciel essentiel pour arrêter le système. En outre, il pourrait être utile d'ajouter *live-tools* pour copier l'image dans la RAM et *eject* pour éjecter le support live. Alors :

```
$ echo "live-tools user-setup sudo eject" > config/package-lists/recommends.
.list.chroot
```

Après cela, créez le répertoire `/etc/skel` dans `config/includes.chroot` avec un fichier `.xsession` personnalisé pour l'utilisateur par défaut qui va lancer *metacity* et *xvncviewer*, en reliant le port 5901 sur un serveur à 192.168.1.2 :

```
$ mkdir -p config/includes.chroot/etc/skel
$ cat > config/includes.chroot/etc/skel/.xsession << EOF
#!/bin/sh

/usr/bin/metacity &
/usr/bin/xvncviewer 192.168.1.2 :1

exit
EOF
```

Construire l'image :

```
# lb build
```

Amusez-vous bien !

18.6 Une image de base pour une clé USB de 128 Mo

Cas d'utilisation : Créer une image par défaut avec certains composants supprimés afin de l'adapter sur une clé USB de 128 Mo avec un peu d'espace laissé pour l'utiliser à votre convenance.

Pour l'optimisation d'une image adaptée à la dimension de certains

supports, vous avez besoin de comprendre le compromis que vous faites entre la taille et la fonctionnalité. Dans cet exemple, nous ne réduisons la taille que pour faire place aux éléments supplémentaires dans la limite de 128 Mo, mais en évitant de détruire l'intégrité des paquets contenus, telle que la purge des données de localisation via le paquet *localepurge*, ou d'autres optimisations "intrusives". On notera en particulier que nous utilisons `--debootstrap-options` pour créer un système minimal à partir de zéro.

```
$ lb config -k 486 --apt-indices false --apt-recommends false --debootstrap-
-options "--variant=minbase" --firmware-chroot false --memtest none
```

Pour faire que l'image fonctionne correctement, nous devons ajouter à nouveau au moins deux paquets recommandés qui sont laissés de côté par l'option `--apt-recommends false`. Voir < [Régler APT pour économiser de l'espace](#) >

```
$ echo "user-setup sudo" > config/package-lists/recommends.list.chroot
```

Maintenant, construisez l'image de la manière habituelle :

```
# lb build 2>&1 | tee build.log
```

Sur le système de l'auteur au moment de l'écriture, la configuration ci-dessus produisait une image de 77 Mo. Cela se compare favorablement avec l'image de 177 Mo produite par la configuration par défaut dans < [Tutoriel 1](#) >.

La plus grande source d'économie d'espace ici, en comparaison

avec la construction d'une image par défaut sur une architecture i386, est de sélectionner uniquement la saveur du noyau 486 au lieu de la valeur par défaut `-k "486 686-pae"`. Laisser tomber les index d'APT avec `--apt-indices false` permet aussi d'économiser une bonne quantité d'espace, le compromis étant que vous devez faire `apt-get update` avant d'utiliser `apt` dans le système live. Abandonner les paquets recommandés avec `--apt-recommends false` économise de l'espace supplémentaire, au détriment de certains paquets dont vous vous attendriez autrement qu'ils soient là. `--debootstrap-options "--variant=minbase"` construit un système minimal dès le début. L'utilisation de `--firmware-chroot false` n'inclut pas automatiquement les paquets de micrologiciels. Finalement, `--memtest none` prévient l'installation d'un testeur de mémoire.

Remarque : Un système minimal peut également être obtenu en utilisant des hooks comme par exemple le `stripped.hook.chroot` dans `/usr/share/doc/live-build/examples/hooks`, il peut gagner de petites quantités supplémentaires d'espace et produire une image de 62 Mo. Cependant, il le fait par l'élimination de la documentation et d'autres fichiers des paquets installés sur le système. Cela porte atteinte à l'intégrité de ces paquets et, comme averti par le commentaire d'en-tête, cela peut avoir des conséquences imprévues. C'est pourquoi l'utilisation de *debootstrap* est la méthode recommandée pour atteindre cet objectif.

18.7 Un bureau GNOME localisé avec un installateur

Cas d'utilisation : Créer une image de bureau GNOME, localisée pour la Suisse et incluant un installateur.

Nous voulons faire une image iso-hybrid pour l'architecture i386

en utilisant notre bureau préféré, dans ce cas, GNOME, contenant tous les paquets qui seraient installés par l'installateur Debian standard pour GNOME.

Notre premier problème est la découverte des noms des tâches appropriées. Actuellement, *live-build* ne peut pas aider à faire cela. Alors que nous pourrions être chanceux et trouver ces noms par essais et erreurs, il existe un outil, `grep-dctrl`, qui peut être utilisé pour découvrir les descriptions des tâches dans `tasksel-data`. Pour la préparation, assurez-vous d'avoir ces deux outils :

```
# apt-get install dctrl-tools tasksel-data
```

Maintenant, nous pouvons rechercher les tâches appropriées, d'abord avec :

```
$ grep-dctrl -FTest-lang de /usr/share/tasksel/descs/debian-tasks.desc -<->
sTask
Task : german
```

Par cette commande, nous découvrons que la tâche est appelée, assez clairement, `german`. Maintenant, pour trouver les tâches liées :

```
$ grep-dctrl -FEnhances german /usr/share/tasksel/descs/debian-tasks.desc -<->
sTask
Task : german-desktop
Task : german-kde-desktop
```

Pendant le démarrage, nous allons générer la locale **de_CH.UTF-8** et sélectionner la disposition de clavier **ch**. Maintenant, nous

allons mettre les morceaux ensemble. En nous rappelant grâce à < **Utilisation des métapaquets** > que les métapaquets sont préfixés task-, nous précisons ces paramètres de la langue pendant l'amorçage, puis nous ajoutons les paquets de priorité standard et tous nos métapaquets découverts à notre liste de paquets comme suit :

912

```
$ mkdir live-gnome-ch
$ cd live-gnome-ch
$ lb config \
  -a i386 \
  -k 486 \
  --bootappend-live "boot=live components locales=de_CH.UTF-8 keyboard-<↵
    layouts=ch" \
  --debian-installer live
$ echo '! Packages Priority standard' > config/package-lists/standard.list.<↵
  chroot
$ echo task-gnome-desktop task-german task-german-desktop >> config/package-<↵
  -lists/desktop.list.chroot
$ echo debian-installer-launcher >> config/package-lists/installer.list.<↵
  chroot
```

913

Notez que nous avons inclus le paquet *debian-installer-launcher* pour lancer l'installateur à partir du bureau live, nous avons également précisé le noyau 486, parce qu'il est actuellement nécessaire faire que l'installateur et le noyau du système live coïncident pour que le lanceur fonctionne correctement.

Appendix

Guide de style

19. Guide de style

19.1 Lignes directrices pour les auteurs

Cette section traite des considérations générales à prendre en compte lors de la rédaction de la documentation technique pour *live-manual*. Elles sont divisées en caractéristiques linguistiques et en procédures recommandées.

Remarque : Les auteurs doivent lire d'abord < [Contribuer à ce document](#) >

19.1.1 Caractéristiques linguistiques

- *Utilisez un anglais simple*

Gardez à l'esprit qu'un pourcentage élevé de vos lecteurs ne sont pas de langue maternelle anglaise. Donc, en règle générale, essayez d'utiliser des phrases significatives courtes, suivies d'un point.

Cela ne signifie pas que vous devez utiliser un style naïf et simpliste. Il s'agit d'une suggestion pour essayer d'éviter, autant que possible, phrases subordonnées complexes qui rendent le texte difficile à comprendre pour les locuteurs non natifs de l'anglais.

- *Variété de l'anglais*

Les variétés les plus répandues de l'anglais sont la britannique et l'américain, donc il est très probable que la plupart des auteurs utilisent l'un ou l'autre. Dans un environnement collaboratif, la variété idéale serait « l'anglais international », mais il est très difficile, pour ne pas dire impossible, de se prononcer sur quelle

variété parmi toutes celles qui existent déjà, est la meilleure à utiliser.

Nous croyons que les différentes variétés peuvent se mélanger sans créer incompréhensions, mais en termes généraux, vous devriez essayer d'être cohérent et avant de décider entre britannique, américain ou toute autre variété anglaise à votre discrétion, s'il vous plaît jeter un oeil à la façon dont d'autres gens écrivent et essayer de les imiter.

- *Soyez équilibré*

Ne soyez pas partial. Évitez d'inclure des références à des idéologies totalement étrangères à *live-manual*. L'écriture technique doit être aussi neutre que possible. Il est dans la nature même de l'écriture scientifique.

- *Soyez politiquement correct*

Essayez d'éviter un langage sexiste autant que possible. Si vous avez besoin de faire référence à la troisième personne du singulier, de préférence utilisez « they » plutôt que « he » ou « she » ou inventions maladroites telles que « s/he », « s(he) », etc.

- *Soyez concis*

Allez droit au but et ne pas errer sans but. Donner autant d'informations que nécessaire, mais ne donnez pas plus d'informations que nécessaire, c'est-à-dire, ne pas expliquer les détails inutiles. Vos lecteurs sont intelligents. Présumez une certaine connaissance préalable de leur part.

- *Minimiser le travail de traduction*

Gardez à l'esprit que ce que vous écrivez devra être traduit dans plusieurs autres langues. Cela implique qu'un certain nombre de gens devront faire un travail supplémentaire si vous ajoutez des informations inutiles ou redondantes.

• *Soyez cohérent*

Comme suggéré précédemment, il est presque impossible de normaliser un document collaboratif dans un ensemble parfaitement unifié. Cependant, tous les efforts de votre côté pour écrire d'une manière cohérente avec le reste des auteurs seront appréciés.

• *Soyez cohésive*

Utilisez autant de dispositifs de formation de texte que nécessaire pour rendre votre texte cohésive et sans ambiguïtés. (Les dispositifs de formation de texte sont des marqueurs linguistiques tels que les connecteurs).

• *Soyez descriptif*

Il est préférable de décrire le point en une ou plusieurs paragraphes que simplement en utilisant un certain nombre de phrases dans un style typique "changelog". Décrivez-le ! Vos lecteurs apprécieront ça.

• *Dictionnaire*

Cherchez le sens des mots dans un dictionnaire ou une encyclopédie si vous ne savez pas comment exprimer certaines notions en anglais. Mais gardez à l'esprit qu'un dictionnaire peut être votre meilleur ami ou peut se transformer en votre pire ennemi si vous ne savez pas comment l'utiliser correctement.

L'anglais possède le plus grand vocabulaire qui existe (avec plus d'un million de mots). Beaucoup de ces mots sont des emprunts d'autres langues. Lorsque l'on regarde le sens des mots dans un dictionnaire bilingue, la tendance d'un locuteur non natif de l'anglais est de choisir celui qui sonne plus similaire dans leur langue maternelle. Cela se transforme souvent en un discours excessivement formelle qui ne semble pas tout à fait naturel en anglais.

935

En règle générale, si un concept peut être exprimé en utilisant différents synonymes, c'est un bon conseil choisir le premier mot proposé par le dictionnaire. En cas de doute, le choix des mots d'origine germanique (Habituellement mots monosyllabiques) est souvent la bonne chose à faire. Il faut savoir que ces deux techniques peuvent produire un discours plutôt informel, mais au moins votre choix des mots sera d'utilisation grand et généralement accepté.

944

L'utilisation d'un dictionnaire de collocations est recommandé. Ils sont extrêmement utiles quand il s'agit de savoir quels mots surviennent généralement ensemble.

945

Encore une fois, c'est une bonne pratique apprendre à partir du travail des autres. Grâce à un moteur de recherche pour vérifier comment les autres auteurs utilisent certaines expressions peut aider beaucoup.

946

• *Faux amis, expressions idiomatiques et autres expressions*

947

Attention aux faux amis. Peu importe comment vous êtes compétent dans une langue étrangère, vous ne pouvez pas vous empêcher de tomber de temps en temps dans le piège de ce qu'on appelle les « faux amis », des mots qui se ressemblent dans les deux langues, mais dont les significations ou les utilisations pourrait être complètement différent.

948

Essayez d'éviter les expressions idiomatiques autant que possible. Les expressions idiomatiques sont des expressions qui peuvent transmettre un sens complètement différent de ce que leurs mots individuels semblent signifier. Parfois, les expressions idiomatiques peuvent être difficiles à comprendre, même pour les locuteurs natifs de l'anglais !

949

• *Évitez l'argot, les abréviations, les contractions...*

950

Même si vous êtes encouragés à utiliser un langage simple,

951

l'anglais de tous les jours, la rédaction technique appartient au registre formel de la langue. 960

Essayez d'éviter l'argot, abréviations inhabituels qui sont difficiles à comprendre et surtout les contractions qui tentent d'imiter le langage parlé. Sans oublier typique expressions d'irc et favorables à la famille.

19.1.2 Procédures

• *Tester avant d'écrire*

Il est important que les auteurs testent leurs exemples avant de les ajouter à *live-manual* pour s'assurer que tout fonctionne comme décrit. Tester sur un chroot propre ou une machine virtuelle peut être un bon point de départ. Par ailleurs, il serait idéal si les tests ont ensuite été effectués sur des machines différentes avec un matériel différent pour repérer d'éventuels problèmes qui pourraient survenir.

• *Exemples*

En fournissant un exemple essayer d'être aussi précis que possible. Un exemple est, après tout, juste un exemple.

Il est souvent préférable d'utiliser une ligne qui ne s'applique qu'à un cas particulier que l'utilisation d'abstractions qui peuvent confondre vos lecteurs. Dans ce cas, vous pouvez fournir une brève explication des effets de l'exemple proposé.

Il peut y avoir des exceptions lorsque l'exemple suggère d'utiliser certaines commandes potentiellement dangereuses qui, si elles sont mal utilisées, peuvent causer des pertes de données ou d'autres effets indésirables similaires. Dans ce cas, vous devez fournir une explication approfondie des effets secondaires possibles.

• *Liens externes*

Les liens externes ne doivent être utilisés lorsque l'information sur ces sites est cruciale quand il s'agit de comprendre un point particulier. Même si, essayez d'utiliser des liens externes aussi peu que possible. Les liens internet sont susceptibles de changer de temps en temps résultant en des liens brisés et en laissant vos arguments dans un état incomplet.

D'ailleurs, les gens qui lisent le manuel hors ligne n'auront pas la chance de suivre ces liens.

• *Évitez les marques et les choses qui violent la licence sous laquelle le manuel est publié*

Essayez d'éviter les marques autant que possible. Gardez à l'esprit que d'autres projets peuvent utiliser la documentation que vous écrivez. Donc, vous compliquez les choses pour eux si vous ajoutez certaines matières spécifiques.

live-manual est sous la licence GNU GPL. Cela a un certain nombre de conséquences qui s'appliquent à la distribution de la matière (de toute nature, y compris les graphiques ou logos protégées) qui est publiée avec le manuel.

• *Ecrire un premier projet, réviser, modifier, améliorer, refaire si nécessaire*

- Remue-méninges !. Vous devez organiser vos idées d'abord dans une séquence logique des événements.

- Une fois que vous avez en quelque sorte organisé ces idées dans votre esprit écrire un premier projet.

- Réviser la grammaire, la syntaxe et l'orthographe. Gardez à l'esprit que les noms propres des versions, tels que **jessie** ou **sid**, ne doivent pas être capitalisés lorsqu'ils sont utilisés comme noms de code. Pour vérifier l'orthographe, on peut exécuter la cible "spell". C'est à dire, `make spell`

- Améliorer vos phrases et refaire n'importe quelle partie si nécessaire.

• *Chapitres*

Utilisez le système de numérotation classique des chapitres et sous-titres. Par exemple 1, 1.1, 1.1.1, 1.1.2 ... 1.2, 1.2.1, 1.2.2 ... 2, 2.1 ... et cetera. Voir marqueurs ci-dessous.

Si vous avez d'énumérer une série d'étapes ou phases dans votre description, vous pouvez également utiliser des nombres ordinaux : premier, deuxième, troisième ... ou d'abord, puis, après cela, enfin ... Sinon, vous pouvez utiliser les éléments à puces.

• *Balises*

Et finalement, *live-manual* utilise < [SiSU](#) > pour traiter les fichiers texte et pour produire plusieurs formats. Il est recommandé de consulter le < [manuel SiSU](#) > pour se familiariser avec son balisage, ou bien tapez :

```
$ sisu --help markup
```

Voici quelques exemples de balisage qui peuvent éventuellement vous aider :

- Pour accent/texte en gras :

```
*{foo}* or !{foo}!
```

il produit : **foo** or **foo** . Utiliser pour mettre l'accent sur certains mots-clés.

- Pour italique :

```
/ {foo} /
```

il produit : *foo*. Utiliser par exemple, pour les noms des paquets Debian.

- Pour monospace :

```
# {foo} #
```

il produit : `foo`. Utiliser par exemple, pour les noms des commandes. Et aussi pour souligner certains mots clés ou des choses comme les chemins.

- Pour les blocs de code :

```
code{
    $ foo
    # bar
}code
```

il produit :

```
$ foo
# bar
```

Utilisez `code{` pour ouvrir et `}code` pour fermer les balises. Il est important de se rappeler de laisser un espace au début de chaque ligne de code.

19.2 Lignes directrices pour les traducteurs

Cette section traite des considérations générales à prendre en compte lors de la traduction du contenu du *live-manual*.

Comme recommandation générale, les traducteurs doivent avoir lu et compris les règles de traduction qui s'appliquent à leurs langues spécifiques. Habituellement, les groupes de traduction et listes de diffusion fournissent des informations sur la façon de produire un travail de traduction qui respecte les normes de qualité de Debian.

Remarque : Les traducteurs doivent aussi lire < [Contribuer à ce document](#) >. En particulier, la section < [Traduction](#) >.

19.2.1 Conseils de traduction

- *Commentaires*

Le rôle du traducteur est de transmettre le plus fidèlement possible le sens des mots, des phrases, des paragraphes et des textes comme écrit par les auteurs originaux dans leur langue cible.

Donc, ils devraient s'abstenir d'ajouter des commentaires personnels ou des morceaux d'informations supplémentaires de leur propre chef. S'ils veulent ajouter un commentaire pour d'autres traducteurs travaillant sur les mêmes documents, ils peuvent le laisser dans l'espace réservé pour cela. Autrement dit, l'en-tête des chaînes dans les fichiers **po** précédés d'un signe **#**. Nombreux logiciels de traduction graphiques peuvent gérer automatiquement ces types de commentaires.

- *NDT, Note Du Traducteur*

Il est parfaitement acceptable cependant, inclure un mot ou une expression entre parenthèses dans le texte traduit si, et seulement si, cela fait le sens d'un mot ou d'une expression difficile plus clair

pour le lecteur. A l'intérieur des parenthèses, le traducteur doit mettre en évidence que l'addition a été leur en utilisant l'abréviation "NDT" ou "Note Du Traducteur".

- *Phrases impersonnelles*

Les documents écrits en anglais font une utilisation intensive de la forme impersonnelle "you". Dans d'autres langues qui ne partagent pas cette caractéristique, ce pourrait donner la fausse impression que les textes originaux traitent directement le lecteur quand en réalité ils ne le font pas. Les traducteurs doivent être conscients de ce fait et traduire dans leur langue le plus fidèlement que possible.

- *Faux amis*

Le piège des "faux amis" expliqué avant s'applique surtout aux traducteurs. Vérifiez le sens des faux amis suspects en cas de doute.

- *Balisage*

Les traducteurs qui travaillent d'abord avec les fichiers **pot** et plus tard avec les fichiers **po** trouveront de nombreuses caractéristiques de balisage dans les chaînes. Ils peuvent traduire le texte de toute façon, tant qu'il est traduisible, mais il est extrêmement important qu'ils utilisent exactement le même balisage que la version originale anglaise.

- *Blocs de code*

Même si les blocs de code sont généralement intraduisibles, les inclure dans la traduction est la seule façon d'obtenir une traduction complète 100%. Et même si cela signifie plus de travail au début car ça peut exiger l'intervention des traducteurs si le code change, il est le meilleur moyen, à long terme, d'identifier ce qui a déjà été traduit et ce qui n'a pas, lors de la vérification de l'intégrité des fichiers .po.

- 1010 • *Sauts de ligne*
- 1011 Les textes traduits doivent avoir les mêmes sauts de lignes exactes que les textes originaux. Veuillez appuyer sur “Entrée” ou tapez si elles apparaissent dans les fichiers originaux. Ces nouvelles lignes apparaissent souvent, par exemple, dans les blocs de code.
- 1012 Ne vous méprenez pas, cela ne signifie pas que le texte traduit doit avoir la même longueur que la version anglaise. C’est presque impossible.
- 1013 • *Chaînes intraduisibles*
- 1014 Les traducteurs ne doivent jamais traduire :
- 1015 - Les noms de code des versions (qui doivent être écrits en minuscules)
- 1016 - Les noms des logiciels
- 1017 - Les commandes fournies à titre d’exemples
- 1018 - Métadonnées (souvent entre deux points :**metadata** :)
- 1019 - Liens
- 1020 - Les chemins

SiSU Metadata, document information

Titre : Manuel Live Systems

Auteur : Projet Live Systems <debian-live@lists.debian.org>

Droits relatifs à la ressource : Copyright : Copyright (C) 2006-2014 Live Systems Project
 License : Ce programme est un logiciel libre ; vous pouvez le redistribuer ou le modifier suivant les termes de la Licence Générale Publique GNU telle que publiée par la Free Software Foundation : soit la version 3 de cette licence, soit (à votre gré) toute version ultérieure.

Ce programme est distribué dans l'espoir qu'il vous sera utile, mais SANS AUCUNE GARANTIE : sans même la garantie implicite de COMMERCIALISABILITÉ ni d'ADÉQUATION À UN OBJECTIF PARTICULIER. Consultez la Licence Générale Publique GNU pour plus de détails.

Vous devriez avoir reçu une copie de la Licence Générale Publique GNU avec ce programme ; si ce n'est pas le cas, consultez < <http://www.gnu.org/licenses/> >.

Le texte complet de la Licence Générale Publique GNU peut être trouvé dans le fichier /usr/share/common-licenses/GPL-3

Éditeur : Projet Live Systems <debian-live@lists.debian.org>

Date : 2014-10-25

Version Information

Fichier source : live-manual.ssm.sst

Filetype : SiSU text 2.0, UTF-8 Unicode text, with very long lines

Source Digest : SHA256(live-manual.ssm.sst)=13de032a0fbc460eccc14f6-790dad9a47d39f23e0330ddb3bd623d9f3a94ef

Generated

Dernière production du document (metaverse) : 2014-10-25 13 :14 :09 +0000

Généré par : SiSU 5.7.1 of 2014w41/7 (2014-10-19)

Version de Ruby : ruby 2.1.3p242 (2014-09-19) [x86_64-linux-gnu]