

japanese english

Live システムマニュアル

Live システムプロジェクト <debian-live@lists.debian.org>

Copyright © 2006-2014 Live Systems Project

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

The complete text of the GNU General Public License can be found in `/usr/share/common-licenses/GPL-3` file.

Contents		Installation	10
About	2	3. Installation	10
このマニュアルについて	3	3.1 Requirements	10
1. このマニュアルについて	3	3.2 Installing live-build	10
1.1 せっかちな人向け	3	3.2.1 From the Debian repository	10
1.2 用語	3	3.2.2 From source	10
1.3 著者	4	3.2.3 From 'snapshots'	11
1.4 この文書への貢献	5	3.3 Installing live-boot and live-config	11
1.4.1 変更の適用	5	3.3.1 From the Debian repository	11
1.4.2 翻訳	6	3.3.2 From source	11
Live システムプロジェクトへの貢献	7	3.3.3 From 'snapshots'	12
2. Live システムプロジェクトについて	7	基本	13
2.1 動機	7	4. 基本	13
2.1.1 現在の Live システムの問題点	7	4.1 Live システムとは何?	13
2.1.2 自身の Live システムを作成する理由	7	4.2 ビルド済みイメージのダウンロード	14
2.2 哲学	7	4.3 ウェブ Live イメージビルダーの利用	14
2.2.1 Debian 「main」 の変更しないパッケージしか 使いません	7	4.3.1 ウェブビルダーの使い方と注意	14
2.2.2 Live システム固有のパッケージ設定はありま せん	7	4.4 最初の段階: ISO hybrid イメージのビルド	14
2.3 連絡先	8	4.5 ISO hybrid Live イメージの利用	15
ユーザ	9	4.5.1 ISO イメージの実際のメディアへの書き込み	15
		4.5.2 ISO hybrid イメージの USB メモリへのコピー	15
		4.5.3 USB メモリの空きスペースの利用	16
		4.5.4 Live メディアのブート	16
		4.6 仮想マシンを利用したテスト	16
		4.6.1 QEMU での ISO イメージのテスト	17
		4.6.2 VirtualBox での ISO イメージのテスト	17
		4.7 HDD イメージのビルド及び利用	18
		4.8 netboot イメージのビルド	18
		4.8.1 DHCP サーバ	19
		4.8.2 TFTP サーバ	20

4.8.3 NFS サーバ	20	7.3 ファイルによる lb config の補完	27
4.8.4 ネットワーク経由のブートをテストする方法	20	7.4 独自化タスク	28
4.8.5 Qemu	20		
4.9 ウェブブート	21	インストールするパッケージの独自化	29
4.9.1 ウェブブートファイルの取得	21		
4.9.2 ウェブブートイメージの起動	21		
Overview of tools	23	8. インストールするパッケージの独自化	29
5. Overview of tools	23	8.1 パッケージソース	29
5.1 The live-build package	23	8.1.1 ディストリビューション、アーカイブ領域とモード	29
5.1.1 The lb config command	23	8.1.2 ディストリビューションミラー	30
5.1.2 The lb build command	24	8.1.3 ビルド時に利用するディストリビューションミラー	30
5.1.3 The lb clean command	24	8.1.4 実行時に利用するディストリビューションミラー	30
5.2 The live-boot package	24	8.1.5 追加リポジトリ	30
5.3 The live-config package	24	8.2 Choosing packages to install	31
設定の管理	25	8.2.1 Package lists	31
6. 設定の管理	25	8.2.2 Using metapackages	31
6.1 設定変更への対応	25	8.2.3 Local package lists	32
6.1.1 自動化スクリプトを使う理由は？ それは何をするもの？	25	8.2.4 Local binary package lists	32
6.1.2 自動化スクリプトの使用例	25	8.2.5 Generated package lists	32
6.2 Git 経由で公開されている設定の複製	26	8.2.6 Using conditionals inside package lists	33
収録内容の独自化	27	8.2.7 Removing packages at install time	33
7. 独自化の概要	27	8.2.8 Desktop and language tasks	33
7.1 ビルド時とブート時の設定	27	8.2.9 Kernel flavour and version	34
7.2 ビルド段階	27	8.2.10 Custom kernels	34
		8.3 Installing modified or third-party packages	35
		8.3.1 Using packages.chroot to install custom packages	35
		8.3.2 Using an APT repository to install custom packages	35
		8.3.3 Custom packages and APT	35

8.4 Configuring APT at build time	36	バイナリイメージの独自化	48
8.4.1 Choosing apt or aptitude	36	11. バイナリイメージの独自化	48
8.4.2 Using a proxy with APT	36	11.1 ブートローダ	48
8.4.3 Tweaking APT to save space	36	11.2 ISO メタ情報	48
8.4.4 Passing options to apt or aptitude	37	Debian インストーラの独自化	49
8.4.5 APT pinning	37	12. Debian インストーラの独自化	49
収録内容の独自化	39	12.1 Debian インストーラの種類	49
9. 収録内容の独自化	39	12.2 preseed による Debian インストーラの独自化	50
9.1 Includes	39	12.3 Debian インストーラの収録内容の独自化	50
9.1.1 Live/chroot ローカルインクルード	39	プロジェクト	51
9.1.2 バイナリローカルインクルード	40	プロジェクトへの貢献	52
9.2 フック	40	13. プロジェクトへの貢献	52
9.2.1 Live/chroot ローカルフック	40	13.1 変更を加える	52
9.2.2 ブート時フック	40	バグの報告	54
9.2.3 バイナリローカルフック	40	14. バグの報告	54
9.3 Debconf 質問の preseed	40	14.1 既知の問題	54
Customizing run time behaviours	42	14.2 最初から再ビルド	54
10. Customizing run time behaviours	42	14.3 最新のパッケージを使う	54
10.1 Customizing the live user	42	14.4 情報収集	54
10.2 Customizing locale and language	42	14.5 可能であれば失敗している状況を分離する	55
10.3 Persistence	43	14.6 正しいパッケージに対してバグを報告する	55
10.3.1 The persistence.conf file	45	14.6.1 ビルド時のパッケージ収集中	56
10.3.2 Using more than one persistence store	45	14.6.2 ビルド時のパッケージインストール中	56
10.4 Using persistence with encryption	46		

14.6.3 ブート時	56	例	65
14.6.4 実行時	56		
14.7 調査してください	56	18. 例	65
14.8 バグの報告先	57	18.1 例の使用	65
		18.2 チュートリアル 1: デフォルトイメージ	65
		18.3 チュートリアル 2: ウェブブラウザユーティリティ	66
		18.4 チュートリアル 3: 私的イメージ	66
		18.4.1 最初の改訂	66
		18.4.2 2 回目の改訂	67
		18.5 VNC 公衆クライアント	68
		18.6 128MB USB メモリ向けの基本イメージ	69
		18.7 地域化した GNOME デスクトップとインストーラ	70
コーディングスタイル	58	付録	72
15. コーディングスタイル	58		
15.1 互換性	58	スタイルガイド	73
15.2 インデント	58	19. スタイルガイド	73
15.3 改行	58	19.1 著者向けガイドライン	73
15.4 変数	58	19.1.1 言語特性	73
15.5 その他	59	19.1.2 手順	74
		19.2 翻訳者向けガイドライン	76
		19.2.1 翻訳の手がかり	76
手順	60	SiSU Metadata, document information	78
16. 手順	60		
16.1 主要リリース	60		
16.2 ポイントリリース	60		
16.2.1 ある Debian リリースの最後のポイントリ リース	60		
16.2.2 ポイントリリース告知用テンプレート	60		
Git リポジトリ	62		
17. Git リポジトリ	62		
17.1 リポジトリを複数処理	62		
例	64		

- 1 **Live** システムマニュアル

2 **About**

3 このマニュアルについて

4 1. このマニュアルについて

5 このマニュアルは Live システムプロジェクトと、特に Debian 8.0 「jessie」リリースに向けてプロジェクトにより作られるソフトウェアに関連するあらゆる文書にアクセスするための一元的な起点となります。最新版は常に <http://live-systems.org/> にあります。

6 Live マニュアルは第一に Live システムのビルドの支援を扱い、エンドユーザ向けの話は扱いませんが、エンドユーザにとって有用な情報がいくらかあるかもしれません。<基本>ではビルド済みイメージのダウンロードや、ウェブビルダーを使うかシステム上の live ビルドを直接実行することでメディアやネットワークからイメージをブートさせる準備について触れています。<実行時の挙動の変更>では、キーボードレイアウトやロケールの選択、再起動をまたいで状態を引き継がせる仕組みの利用等、ブートプロンプトで指定できるオプションをいくらか説明しています。

7 提示されているコマンドの一部にはスーパーユーザ権限で実行しなければならないものもあります。これは su で root ユーザになるか、sudo を使って実行します。権限のないユーザで実行できるコマンドと実行にスーパーユーザ権限を必要とするコマンドは、それぞれのコマンドの前に \$ があるか # があるかで区別します。この記号はコマンドの一部ではありません。

8 1.1 せっかちな人向け

9 このマニュアルにある全てが、少なくとも一部のユーザにとって重要だと確信していますが、触れている内容が多岐にわたることや、詳細を掘り下げるよりも先に、まずはソフトウェアをうまく使う経験をしたいてであろうということをわかっています。したがって、以下の順に読み進めることを提案します。

10 最初にこの章 <このマニュアルについて> を始めから <用語> 節まで読んでください。次に <例> 節の最初にある 3 つのチュートリアルまで飛ばします。ここではイメージのビルドと独自化の基本について教えるようになっていきます。<例の使用> を最初に読み、引き続き <チュートリアル 1: デフォルトイメージ> と <チュートリアル 2: ウェブブラウザユーティリティ> を、最後に <チュートリアル 3: 私的イメージ> を読んでください。チュートリアル群を終えるまでに、Live システムでできることが何なのかわかってくるでしょう。

11 それから戻り、マニュアルをもっと掘り下げて学習していくことを勧めています。恐らく、その次は <基本> を読み、<netboot イメージのビルド> に軽く目を通して、<独自化概要> とそれに続く章を読んで終えるのがいいでしょう。この時点までに、Live システムでできることを知ることがすっかり面白くなってマニュアルの残りを隅から隅まで読む気になっていることを期待します。

12 1.2 用語

13 • **Live システム** : ハードドライブにインストールしなくてもブートできるオペレーティングシステムです。Live システムはそのコンピュータのハードドライブに既にインストールされているローカルのオペレーティングシステムやファイルを、そうするように指示しない限り変更しません。Live システムは通常、CD や DVD、USB メモリ等のメディアからブートされます。ネットワーク越しにブートできるもの (netboot イメージ経由、<netboot イメージのビルド> 参照) やインターネット越しにブートできるもの (起動パラメータ fetch=URL 経由、<Webbooting> 参照) もあります。

14 • **Live メディア** : Live システムとは異なり、live メディアは live-build により作成されたバイナリを書き込んでその Live システムをブートするのに利用する CD や DVD、USB メモリを指します。もっと広い意味では、この語はネットワークブートファ

イルの位置等、Live システムをブートする目的でこのバイナリ 27
が置かれている任意の場所を指すこともあります。

- 15 • **Live システムプロジェクト** : *live-boot*、*live-build*、*live-config*、*live-tools*、*live-manual* パッケージを特に保守しているプロジェクトです。
- 16 • **ホストシステム** : Live システムの作成に利用される環境です。
- 17 • **ターゲットシステム** : Live システムの実行に利用される環境です。
- 18 • ***live-boot*** : Live システムのブートに利用するスクリプト集です。
- 19 • ***live-build*** : 独自化した Live システムのビルドに利用するスクリプト集です。
- 20 • ***live-config*** : Live システムのブート処理中の設定に利用するスクリプト集です。
- 21 • ***live-tools*** : 実行中の Live システム内で有用なタスクを実行するのに利用する追加のスクリプト集です。
- 22 • ***live-manual*** : この文書は *live-manual* というパッケージで保守されています。
- 23 • **Debian Installer (d-i)** : 公式の Debian ディストリビューション用インストールシステムです。
- 24 • **ブートパラメータ** : *bootloader* プロンプトで入力し、カーネルや *live-config* の動作を変更できるパラメータです。
- 25 • **chroot** : *chroot* プログラム。chroot(8) により、単一のシステム上で異なる GNU/Linux 環境を再起動せずに並行して実行できるようになります。
- 26 • **バイナリイメージ** : *live-image-i386.hybrid.iso* や *live-image-i386.img* 等、Live システムを収録するファイルです。

- **ターゲットディストリビューション** : Live システムがベースとするディストリビューションです。これはホストシステムのディストリビューションとは別のものです。

- **安定版 (stable)/テスト版 (testing)/不安定版 (unstable)** : 安定版 (**stable**) ディストリビューション、現在のコード名 wheezy には、公式にリリースされた最新の Debian ディストリビューションが含まれます。テスト版 (**testing**) ディストリビューション、一時的コード名 **jessie** は次期 *{安定版 (stable)}* リリースを集める場です。このディストリビューションを使う主な利点はソフトウェアのバージョンが *{安定版 (stable)}* リリースと比べて新しいということです。**unstable** ディストリビューション、恒久的コード名 **sid** は Debian の開発が活発に行われる場です。通常、このディストリビューションは開発者や、苦勞をいとわず最新版を使いたい人が利用します。マニュアル全体を通して、リリースを指すのに **jessie** や **sid** 等のコード名を使っています。それこそが、ツール自体がサポートしているものだからです。

1.3 著者

著者一覧 (アルファベット順):

- Ben Armstrong
- Brendan Sleight
- Carlos Zuferrri
- Chris Lamb
- Daniel Baumann
- Franklin Piat
- Jonas Stein
- Kai Hendry

- Marco Amadori
- Mathieu Geli
- Matthias Kirschner
- Richard Nelson
- Trent W. Buck

5139

サポートされている全言語のマニュアルをビルドするにはある程度時間がかかるため、著者が英語版のマニュアルに追加した新しい文書について見直す場合は見直し用に処理を省略させると好都合かもしれません。PROOF=1 を使うと html 形式の *live-manual* をビルドしますが分割版の html ファイルを作成しません。PROOF=2 を使うと pdf 形式の *live-manual* をビルドしますが A4 とレター縦だけです。これが PROOF= を指定すると時間の節約が見込める理由です。例えば:

1.4 この文書への貢献

このマニュアルの作成はコミュニティ中心のプロジェクトで、改善提案や貢献は全て、非常に歓迎されます。コミットキーの取得方法や良いコミットを行うための詳細な情報については、[プロジェクトへの貢献](#) 節を見てください。

1.4.1 変更の適用

マニュアルの英語版に変更を加える場合、manual/en/ にある正しいファイルを編集しないといけません、その貢献を提出する前に出来上がりを確認してください。Live マニュアルの出来上がりを確認する際は、

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

を実行してビルドに必要なパッケージがインストールされていることを確認してください。Git により取得した最上位のディレクトリから

```
$ make build
```

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

翻訳の一つを見直す場合に一つの言語だけをビルドすることもできます。例えば:

```
$ make build LANGUAGES=ja
```

を実行します。文書の種類を指定してビルドすることもできます。例えば

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

あるいは両方を組み合わせて

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

修正が済んで全て良くなったらコミットですが、そのコミットで翻訳を更新するのでない限り `make commit` を使わないようにして

ください。また、その場合にマニュアルの英語版への変更と翻訳を一度にコミットするのではなく、それぞれ分けてコミットするようにしてください。さらなる詳細については [「翻訳」](#) 節を見てください。

1.4.2 翻訳

live-manual を翻訳するには、新しく最初から翻訳を開始するのか、それとも既に存在する翻訳について作業を続けるのか、によって以下の手順を追ってください：

- 新しく最初から翻訳を開始する
 - `manual/pot/` にある `about_manual.ssi.pot`、`about_project.ssi.pot`、`index.html.in.pot` ファイルを (*poedit* 等) 好みのエディタで自分の言語に翻訳し、整合性確認のため翻訳した `.po` ファイルをメーリングリストに送ってください。*live-manual* の整合性確認では `.po` ファイルが 100% 翻訳されていることだけでなく誤りの可能性を検出します。
 - 確認が済んだ後は、自動ビルドでの新しい言語の有効化は最初の翻訳済みファイルを `manual/po/{言語}/` に追加して `make commit` を実行すれば十分です。それから `manual/_sisu/home/index.html` を編集して言語の名前と () 内にその英語名を追加してください。
- 既に存在する翻訳について作業を続ける
 - 対象の言語が既に追加されている場合は、(*poedit* 等) 好みのエディタで `manual/po/` にある残りの `po` ファイルを手当たり次第に翻訳を続けてください。
 - 翻訳済みマニュアルが `.po` ファイルから更新されていることを確実にするためには `make commit` を行う必要があることと、`git add .`、`git commit -m "Translating..."`、`git push` を実行する前に `make build` を実行すると変更を確認できるということを忘れないでください。 `make build`

には相当の時間がかかる可能性があるため、[「変更の適用」](#) で説明しているように、見直しの際は 1 つの言語だけをビルドして確認できることを覚えておくといいいでしょう。

`make commit` を実行するとテキストがいくらか流れていくのを目にするでしょう。これは基本的に処理状態についての情報を示すメッセージで、Live マニュアルの改善のために何ができるのかということを知る手がかりにもなります。致命的エラーが起きていない限り、通常はそのまま進めて貢献を提出できます。

live マニュアルには、翻訳者が未翻訳や変更された文字列を検索するのを大きく支援する 2 つのユーティリティが付属しています。1 つ目は「`make translate`」です。これは各 `.po` ファイル中にどれだけ未翻訳文字列があるのか、詳細を報告するスクリプトを実行します。2 つ目は「`make fixfuzzy`」で、こちらは変更された文字列だけを対象としますが、1 つ 1 つ見つけて修正する作業を支援します。

こういったユーティリティはコマンドラインで翻訳作業を行うのには実際に役立つかもしれませんが、推奨する作業方法は *poedit* のような専用ツールの利用だということに留意してください。Debian 地域化 (I10n) 文書や、特に *live* マニュアル向けの [「翻訳者向けのガイドライン」](#) を読むのも良いことです。

注意: `git` ツリーを `push` する前に `make clean` を実行してきれいにすることができます。この手順は `.gitignore` ファイルのおかげで強制ではありませんが、ファイルを意図せずコミットすることを避けられる良い実践となります。

72	Live システムプロジェクトへの貢献	87	• これは Debian のサブプロジェクトです。	
73	2. Live システムプロジェクトについて		• 単一のディストリビューションの (現在の) 状態を反映します。	88
74	2.1 動機		• 可能な限り多くのアーキテクチャで動作します。	89
	2.1.1 現在の Live システムの問題点		• 変更しない Debian パッケージだけで構成されます。	90
75			• Debian アーカイブにないパッケージは何も含まれません。	91
76			• 変更しない Debian のカーネルを追加パッチなしで利用します。	92
	Live システムプロジェクトが始まったとき、利用可能な Debian ベースの Live システムは既に複数あり、素晴らしい作業を行っていました。Debian の視点から見て、そのほとんどには以下のような不満があります。			
77	• Debian のプロジェクトではないために Debian でのサポートがない。			
78	• 異なるディストリビューション、例えば <i>*{安定版 (testing)}*</i> と <i>*{不安定版 (unstable)}*</i> を混ぜて使っている。			
79	• サポートしているのが i386 だけ。			
80	• 容量節約のためにパッケージの挙動や見た目を変更している。			
81	• Debian アーカイブ外のパッケージを収録している。			
82	• Debian のものではない追加パッチを適用した独自カーネルを使っている。			
83	• 本体のサイズのために巨大で遅く、レスキュー用途に合わない。			
84	• 異なる形式、例えば CD、DVD、USB メモリ、netboot イメージから利用できない。			
85	2.1.2 自身の Live システムを作成する理由			
86	Debian はユニバーサルオペレーティングシステムです: Debian に Live システムがあることで Debian システムを案内、正確に表現することができるとともに、主に以下の利点があります:			
			2.2 哲学	93
			2.2.1 Debian 「main」の変更しないパッケージしか使いません	94
			「main」Debian リポジトリのパッケージだけを利用します。「non-free」は Debian の中には含まれないため、公式の Live システムのイメージでは利用できません。	95
			いかなるパッケージも変更しません。何か変更が必要であれば Debian のそのパッケージのメンテナと調整を行います。	96
			例外として、 <i>live-boot</i> や <i>live-build</i> 、 <i>live-config</i> といった私達の独自のパッケージを開発用の目的 (例えば開発用スナップショットの作成) のため私達自身のリポジトリから一時的に利用するかもしれません。このパッケージ群は定期的に Debian にアップロードされます。	97
			2.2.2 Live システム固有のパッケージ設定はありません	98
			現段階で、インストール例や代替設定は組み込んでいません。パッケージが利用されるのは Debian を普通にインストールした後のものなので全てデフォルト設定です。	99
			別のデフォルト設定が必要であれば Debian のそのパッケージの	100

メンテナと調整を行います。

101 debconf を使うことで提供されるパッケージ設定システムにより、
独自に作成した Live システムのイメージを使って独自に設定した
パッケージをインストールすることができるようになりますが、
〈ビルド済み Live イメージ〉では Live 環境で動作させるために
絶対に必要だという場合を除いて、パッケージをそのデフォルト
設定のままにすることを選択しました。Live 用ツールチェーン
や〈ビルド済みイメージ設定〉への変更よりも、そこで可能である
限り、Debian アーカイブにあるパッケージを Live システムでよ
りよく動作させることを好みます。さらなる情報については、〈独
102 自化概要〉を見てください。

2.3 連絡先

- 103
- **メーリングリスト** : プロジェクトの第一の連絡先は
〈<http://lists.debian.org/debian-live/>〉のメーリングリストです。debian-
live@lists.debian.org 宛てのメールにより、メーリングリスト
に直接メールを送ることができます。メーリングリストのアー
カイブは 〈<http://lists.debian.org/debian-live/>〉で利用できます。
 - 104 • **IRC** : ユーザや開発者達が irc.debian.org (OFTC) の #debian-live
チャンネルにいます。IRC で質問するときは静かに回答を待つ
てください。回答が得られないときはメーリングリストにメール
で質問してください。
 - 105 • **BTS** : 〈[バグの報告](#)〉を見てください。

107 **Installation**108 **3. Installation**109 **3.1 Requirements**

110 Building live system images has very few system requirements:

- 111 • Superuser (root) access
- 112 • An up-to-date version of *live-build*
- 113 • A POSIX-compliant shell, such as *bash* or *dash*
- 114 • python3
- 115 • *debootstrap* or *cdebootstrap*
- 116 • Linux 2.6 or newer.

117 Note that using Debian or a Debian-derived distribution is not required - *live-build* will run on almost any distribution with the above requirements.

118 **3.2 Installing live-build**

119 You can install *live-build* in a number of different ways:

- 120 • From the Debian repository
- 121 • From source
- 122 • From snapshots

123 If you are using Debian, the recommended way is to install *live-build* via the Debian repository.

124 **3.2.1 From the Debian repository**

Simply install *live-build* like any other package:

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

127 **3.2.2 From source**

live-build is developed using the Git version control system. On Debian based systems, this is provided by the *git* package. To check out the latest code, execute:

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

You can build and install your own Debian package by executing:

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

Now install whichever of the freshly built *.deb* files you were interested in, e.g.

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

You can also install *live-build* directly to your system by execut-

ing:

144

135

```
# make install
```

136

and uninstall it with:

137

```
# make uninstall
```

138

3.2.3 From ‘snapshots’

139

If you do not wish to build or install *live-build* from source, you can use snapshots. These are built automatically from the latest version in Git and are available on <http://live-systems.org/debian/>.

140

3.3 Installing live-boot and live-config

141

Note: You do not need to install *live-boot* or *live-config* on your system to create customized live systems. However, doing so will do no harm and is useful for reference purposes. If you only want the documentation, you may now install the *live-boot-doc* and *live-config-doc* packages separately.

142

3.3.1 From the Debian repository

143

Both *live-boot* and *live-config* are available from the Debian repository as per [Installing live-build](#).

3.3.2 From source

To use the latest source from git, you can follow the process below. Please ensure you are familiar with the terms mentioned in [Terms](#).

145

- Checkout the *live-boot* and *live-config* sources

146

147

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

Consult the *live-boot* and *live-config* man pages for details on customizing if that is your reason for building these packages from source.

148

- Build *live-boot* and *live-config* .deb files

149

You must build either on your target distribution or in a chroot containing your target platform: this means if your target is **jessie** then you should build against **jessie**.

150

Use a personal builder such as *pbuilder* or *sbuild* if you need to build *live-boot* for a target distribution that differs from your build system. For example, for **jessie** live images, build *live-boot* in a **jessie** chroot. If your target distribution happens to match your build system distribution, you may build directly on the build system using `dpkg-buildpackage` (provided by the *dpkg-dev* package):

151

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

152

- Use applicable generated .deb files

153

154 As *live-boot* and *live-config* are installed by *live-build* system, installing the packages in the host system is not sufficient: you should treat the generated .deb files like any other custom packages. Since your purpose for building from source is likely to test new things over the short term before the official release, follow [<Installing modified or third-party packages>](#) to temporarily include the relevant files in your configuration. In particular, notice that both packages are divided into a generic part, a documentation part and one or more back-ends. Include the generic part, only one back-end matching your configuration, and optionally the documentation. Assuming you are building a live image in the current directory and have generated all .deb files for a single version of both packages in the directory above, these bash commands would copy all of the relevant packages including default back-ends:

155

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

156 3.3.3 From 'snapshots'

157 You can let *live-build* automatically use the latest snapshots of *live-boot* and *live-config* by configuring the package repository on live-systems.org as a third-party repository in your *live-build* configuration directory.

158 基本

159 4. 基本

160 この章ではビルドプロセスの概要と最も広く利用されている 3 種類
 のイメージの使用手順について簡単に述べます。最も汎用性の
 高い形式のイメージである iso-hybrid は、仮想マシンや光学メ
 ディア、USB ポータブルストレージ機器上で利用できます。特に
 変わった状況では後述のように、hdd 形式の方が適するかもしれ
 ません。この章では netboot 形式のイメージをビルド、利用する
 手順を記載しています。この形式はサーバ上で必要とする準備の
 ためにやや複雑になります。これは netboot についてまだ不慣れ
 な人にとってはわずかに高度な話題となりますが、その準備さえ
 できればローカルネットワーク上でブートするためのイメージを
 テスト、展開するのに非常に便利な方法で、難なくイメージのメ
 ディアを扱うことができるため、ここに収録しています。

161 この節は「ウェブブート」の簡単な手引きで終わっています。これ
 は恐らく異なる目的の異なるイメージを必要に応じて切り替え
 て使う最も簡単な方法で、手段としてインターネットを使いま
 す。

162 この章全体を通して、*live-build* により作成されるデフォルトの
 ファイル名を頻繁に参照しています。「ビルド済みイメージをダ
 ウンロード」した場合、実際のファイル名は異なる場合があります。

163 4.1 Live システムとは何?

164 Live システムとは、通常 CD-ROM や USB メモリ等の取り外し可
 能メディア、あるいはネットワークからコンピュータ上でブート
 されるオペレーティングシステムを意味し、普通のドライブに何
 もインストールせずに利用でき、実行時に自動設定が行われます
 (「用語」参照)。

165 Live システムはオペレーティングシステムで、サポートしている
 うちの単一のアーキテクチャ(現在 amd64 と i386) 向けにビルド
 されています。以下から構成されています。

- **Linux** カーネルイメージ、通常 `mlinuz*` という名前です 166
- 初期 **RAM** ディスクイメージ (**initrd**) : Linux ブート用に用意さ
 167 れた RAM ディスクで、システムのイメージをマウントするの
 に必要となる可能性があるモジュールとマウントするためのス
 クリプトをいくつか収録しています。
- システムイメージ : オペレーティングシステムのファイルシ
 168 ステムのイメージです。通常、Live システムのイメージサイズを
 最小限にするため、SquashFS 圧縮ファイルシステムが利用さ
 れています。このファイルシステムは読み込み専用であること
 に注意してください。そのため、ブート処理中は Live システム
 は RAM ディスクと「ユニオン」機構を利用して実行中のシス
 テム中でファイルを書き込むことができますようにしています。
 ただし、オプションの保持機能を使っていない限り、変更は全
 てシャットダウンにより失われます(「保持機能」参照)。
- ブートローダ : 選択したメディアからブートするように作ら
 169 れた短いコードの集合で、オプション/設定を選択できるプロ
 ンプトやメニューを恐らく提示します。Linux カーネルとその
`initrd` を読み込んでそのシステムのファイルシステム上で実行
 します。前に言及した構成要素を収録する対象メディアやファ
 イルシステムの形式によっては別の方法があります。`isolinux`
 では ISO9660 形式の CD や DVD からのブート、`syslinux` では
 HDD や USB ドライブの VFAT パーティションからのブート、
`extlinux` では ext2/3/4 や btrfs パーティション、`pxelinux` では
 PXE netboot、GRUB では ext2/3/4 パーティション、等。

170 *live-build* を使って Linux カーネル、`initrd`、それを実行するための
 ブートローダを独自仕様で用意して全て 1 つのメディア特有の
 形式 (ISO9660 イメージやディスクイメージ等) でシステムのイ
 メージをビルドできます。

171 4.2 ビルド済みイメージのダウンロード

172 このマニュアルの対象は自分の live イメージの開発やビルドですが、使い方の手引き、あるいは自分でビルドする代わりにビルド済みイメージを簡単に試してみたいこともあるでしょう。[〈live-images の git リポジトリ〉](#) と公式の安定版 (stable) リリースを使ってビルドされたイメージが [〈http://www.debian.org/CD/live/〉](http://www.debian.org/CD/live/) で公開されています。さらに、古いものや今後のリリース、non-free ファームウェアを収録する非公式のイメージ、あるいはドライバが [〈http://live-systems.org/cdimage/release/〉](http://live-systems.org/cdimage/release/) から利用できるようになっています。

173 4.3 ウェブ Live イメージビルダーの利用

174 コミュニティへのサービスとして、ウェブベースの live イメージビルダーサービスを [〈http://live-build.debian.net/〉](http://live-build.debian.net/) で運営しています。このサイトはベストエフォートの方針で保守されています。つまり、最新でいつでも使える状態の維持に努め、大規模な運用停止については問題を告知しますが、100% いつでも使えることやイメージの高速なビルドを保証することはできず、サービスについて解決に時間を要する問題が時々あるかもしれないということです。サービスについて問題や疑問があれば、問題のあるビルドへのリンクを添えて [〈連絡〉](#) してください。

175 4.3.1 ウェブビルダーの使い方と注意

176 ウェブインターフェイスでは現在、オプションの不正な組み合わせを避ける対策を何も取っていません。また、特に、変更すると通常ウェブフォームにある他のオプションのデフォルト値 (つまり *live-build* を直接使った場合の値) が変わるオプションを変更した場合にウェブビルダーはそのデフォルト値を変更しません。最も顕著な例として、`--architectures` をデフォルトの `i386` から `amd64` に変更すると対応するオプション `--linux-flavours` を

デフォルトの `486` から `amd64` に変更する必要があります。ウェブビルダーにインストールされている *live-build* のバージョンやさらなる詳細については `lb_config man` ページを見てください。*live-build* のバージョン番号はウェブビルダーのページ下部に記載されています。

ウェブビルダーにより提示される時間の推定は条件を考慮しない推定であり、実際にビルドにかかる時間を反映していないかもしれません。表示された後に更新もされません。それについては我慢してください。ビルド条件を送信した後にこのページを更新しないでください。更新すると同一のパラメータで再び新たにビルドを送信することになります。ビルドの通知をただの一度も受け取っておらず、十分な時間が確実に過ぎて、通知メールが自分の spam メールフィルタに引っかかかっていないことを確認した場合、[〈連絡〉](#) してください。

ウェブビルダーがビルドできるイメージの種類は限定されています。これにより、利用や保守を簡単、能率的に維持できます。ウェブインターフェイスで提供されていない独自化を行いたい場合は、*live-build* を使って自分のイメージをビルドする方法をこのマニュアルの残りで説明しています。

179 4.4 最初の段階: ISO hybrid イメージのビルド

180 イメージの種類を問わず、イメージをビルドするのに同一の基礎手順を毎回実行する必要があります。最初の例ではビルド用のディレクトリを作成して、このディレクトリに移動してから *live-build* コマンドを以下の順で実行し、X.org のないデフォルトの live システムを収録する基本的な ISO hybrid イメージを作成します。このイメージは CD や DVD メディアへの書き込み、さらに USB メモリへの複製にも適しています。

181 作業ディレクトリの名前は完全に自由ですが、*live-manual* 全体で利用されている例を参考にする場合、特に異なる種類のイメージについて作業、実験している場合、各ディレクトリで作業している

イメージの識別を支援する名前を使うのは良い方法です。ここではデフォルトのシステムをビルドするとして、例えば live-default と呼びましょう。

182

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

Then, run the `lb config` commands. This will create a “config/” hierarchy in the current directory for use by other commands:

184

```
$ lb config
```

No parameters are passed to these commands, so defaults for all of their various options will be used. See <[The `lb config` command](#)> for more details.

これで「config/」階層ができました。 `lb build` コマンドでイメージをビルドします。

187

```
# lb build
```

コンピュータやネットワーク接続の速度により、このプロセスには少々時間がかかるかもしれません。完了すると、live-image-i386.hybrid.iso イメージファイルが使える状態で現在のディレクトリにできているはずです。

4.5 ISO hybrid Live イメージの利用

ISO hybrid イメージをビルド、または <<http://www.debian.org/CD/live/>> にあるものをダウンロードした後、通常は次にブート用メディア

として CD-R(W) や DVD-R(W) の光学メディアか USB メモリを用意します。

4.5.1 ISO イメージの実際のメディアへの書き込み

191

ISO イメージの書き込みは簡単です。 `xorriso` をインストールしてそれをコマンドラインから使ってイメージを書き込むだけです。例えば:

192

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

193

4.5.2 ISO hybrid イメージの USB メモリへのコピー

194

`xorriso` で作られた ISO イメージは `cp` プログラムや同等プログラムを使って単純に USB メモリにコピーすることができます。イメージファイルを置けるだけの十分に大きなサイズの USB メモリを差し込んでそれがどのデバイスなのか決定します。以後 `#{USB メモリ}` として参照します。これは例えば `/dev/sdb` といった USB メモリのデバイスファイルで、例えば `/dev/sdb1` といったパーティションではありません! USB メモリを差し込んでから `dmesg` か、もっと良いのは `ls -l /dev/disk/by-id` の出力を見ると正しいデバイス名を調べることができます。

195

正しいデバイス名を得られたことを確信できたら `cp` コマンドを使ってイメージを USB メモリにコピーします。これを実行すると以前その **USB** メモリにあった内容は全て確実に上書きされます!

196

197

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

198 注意: `sync` コマンドはイメージのコピー中にカーネルによりメモ
 199 リに記憶されているデータが全て USB メモリに書き込まれた
 200 ことを保証するのに有用です。

4.5.3 USB メモリの空きスペースの利用

200 `live-image-i386.hybrid.iso` を USB メモリにコピーすると、最
 201 初のパーティションは Live システムで埋められます。残った空き
 202 スペースを利用するには、`gparted` や `parted` といったパーティシ
 203 ョン作業ツールを使ってその USB メモリに新しいパーティション
 204 を作成します。

```
# gparted ${USB メモリ}
```

202 パーティションの作成後にはファイルシステムを作成する必要が
 203 あります。選択肢には `ext4` 等があります。`${パーティション}` に
 204 は例えば `/dev/sdb2` 等パーティションの名前が入ります。

```
# mkfs.ext4 ${パーティション}
```

204 注意: 余った容量を Windows で使いたい場合ですが、この OS
 205 では最初のパーティション以外にアクセスすることは通常できま
 206 せん。この問題に対する解決策が `<メーリングリスト>` でいくらか
 207 議論されていますが、簡単な解はないようです。

205 **Remember:** 新しい `live-image-i386.hybrid.iso` を USB メモリに
 206 インストールする度に、パーティションテーブルがイメージの
 207 内容で上書きされるために USB メモリにあるデータは全て失われ

るので、追加パーティションをまずバックアップしてから、Live
 イメージの更新後に復帰させるようにしてください。

4.5.4 Live メディアのブート

206 Live メディア CD、DVD、USB メモリ、あるいは PXE ブートでの
 207 初回ブート時に、そのコンピュータの BIOS をまず設定する必要
 208 があるかもしれません。BIOS により機能やキーの割り当てが大き
 209 く異なるため、ここではそれについて深くは触れません。BIOS
 210 によってはブートするデバイスのメニューをブート時に提示させ
 211 るキー割り当てを提供しているものがあり、そのシステムでこれ
 が利用できる場合は最も簡単な方法でしょう。それがない場合は
 BIOS 設定メニューに入って Live システムのブートデバイスを通
 常のブートデバイスよりも前に配置するようにブート順を変更す
 る必要があります。

208 メディアをブートするとブートメニューが表示されているでしょ
 う。ここで単に `enter` を押すと、システムはデフォルトの項目
 Live とデフォルトのオプションを使ってブートします。ブートオ
 プションのさらなる情報については、メニューの「ヘルプ」の項
 目や Live システム内にある `live-boot` 及び `live-config` の man ペ
 ージを見てください。

209 Live を選択してデフォルトのデスクトップ Live イメージをブ
 210 ートしたとして、ブートメッセージが流れた後、自動的に user ア
 211 カウントにログインし、デスクトップがすぐに使える状態で見
 えているはずですが、`<ビルド済みイメージ>` の `standard` や `rescue` 等
 コンソールだけのイメージをブートした場合はコンソールで自動
 的に user アカウントにログインし、シェルプロンプトがすぐに
 使える状態で見えているはずですが。

4.6 仮想マシンを利用したテスト

211 Live イメージを仮想マシン (VM) 内で実行すると開発の面で大き

な時間の節約になるかもしれません。これには注意事項がないというわけではありません:

- 212 • VM の実行にはゲスト OS とホスト OS の両方に十分な RAM
が必要で、CPU には仮想化をハードウェアでサポートして
いるものを推奨します。
- 213 • VM 上での実行には、例えばビデオ性能が低いこと、エミュレー
するハードウェアの選択肢が限られていること等内在する制限
がいくらかあります。
- 214 • 特定のハードウェア向けの開発ではそのハードウェア自体での
実行に代わるものではありません。
- 215 • VM での実行にのみ関連するバグが時々あります。その疑いが
あるときはイメージをハードウェアで直接テストしてくださ
い。
- 216 こういった制約があることを理解した上で利用可能な VM ソフト
ウェアを調べて要件に合うものを選択してください。

217 4.6.1 QEMU での ISO イメージのテスト

218 Debian で最も汎用性の高い VM は QEMU です。プロセッサが
仮想化をハードウェアでサポートしている場合は *qemu-kvm* パッ
ッケージを使ってください。 *qemu-kvm* パッケージの説明に要件の
簡単な一覧があります。

219 プロセッサがサポートしている場合はまず *qemu-kvm* をインス
トールしてください。サポートしている場合は *qemu* をインス
トールしてください。以下の例ではどちらの場合もプログラム名
は *kvm* ではなく *qemu* とします。 *qemu-utils* パッケージもあると
qemu-img で仮想ディスクのイメージを作成するのによいでしょ
う。

220

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

ISO イメージのブートは簡単です:

221

222

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

詳細については *man* ページを見てください。

223

4.6.2 VirtualBox での ISO イメージのテスト

224

virtualbox で ISO をテストするには:

225

226

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

新しい仮想マシンを作成し、*live-image-i386.hybrid.iso* を CD/
DVD デバイスとして利用するようにストレージ設定を変更して
仮想マシンを起動します。

227

注意: X.org を収録している Live システムを *virtualbox* でテスト
したい場合は *live-build* 設定に VirtualBox X.org ドライバパッケ
ージ *virtualbox-guest-dkms* 及び *virtualbox-guest-x11* を収録
するとよいでしょう。収録しない場合、解像度は 800x600 に限定
されます。

228

229

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

dkms パッケージを機能させるためには、そのイメージで利用し

230

ているカーネルの種類のカーネルヘッダもインストールする必要があります。正しいパッケージの選択は上記で作成したパッケージ一覧に正しい *linux-headers* パッケージを手作業により列挙する代わりに *live-build* により自動的に行うことができます。

231

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

4.7 HDD イメージのビルド及び利用

HDD イメージのビルドは全面的に ISO hybrid イメージのビルドと似ていて、`-b hdd` を指定することと出来上がりのファイル名が `live-image-i386.img` で光学メディアに書き込んで使うことができないという点が異なります。このイメージは USB メモリや USB ハードドライブ、その他様々な他のポータブルストレージデバイスからのブートに適しています。通常、この目的には ISO hybrid イメージを代わりに使えますが、BIOS が hybrid イメージを適切に処理できない場合は HDD イメージが必要となります。

注意: 前の例で ISO hybrid イメージを作成している場合 `lb clean` コマンド ([<lb clean コマンド>](#) 参照) で作業ディレクトリをきれいにする必要があります:

235

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

前と同様に `lb config` コマンドを実行します。今回はイメージの種類に HDD を指定する点が異なります:

237

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

それから `lb build` コマンドでイメージをビルドします:

238

239

```
# lb build
```

ビルドが完了すると現在のディレクトリに `live-image-i386.img` ファイルができています。

240

生成されたバイナリイメージには VFAT パーティションと `syslinux` ブートローダが収録され、そのまま USB 機器に書きこめます。繰り返しますが HDD イメージの使い方は USB で ISO hybrid イメージを使うのと同様です。[<ISO hybrid Live イメージの利用>](#) の指示に従ってください。`live-image-i386.hybrid.iso` に代えて `live-image-i386.img` をファイル名に使う点異なります。

241

同様に、Qemu で HDD イメージをテストするには上記の [<Qemu での ISO イメージのテスト>](#) で説明しているように `qemu` をインストールしてください。それから `kvm` か `qemu` のホストシステムで必要バージョンを実行し、最初のハードドライブとして `live-image-i386.img` を指定します。

242

243

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

4.8 netboot イメージのビルド

244

以下の順でコマンドを実行すると X.org のないデフォルトの Live システムを収録する基本的な netboot イメージを作成します。ネットワーク越しのブートに適しています。

245

注意: 前に示した例からどれかを実行した場合、作業ディレクトリを `lb clean` コマンドできれいにする必要があります:

246

247

```
# lb clean
```

248 この特定の場合必要な段階の掃除が `lb clean --binary` では不十分です。netboot イメージのビルドで *live-build* が netboot の準備を自動的に実行するにあたって異なる `initramfs` 設定が必要なことがその原因です。`initramfs` の作成は `chroot` の段階で行われるため、既存のビルドディレクトリで netboot に切り替えるということは `chroot` の段階も再ビルドするということになります。したがって、`lb clean` (これは `chroot` の段階も削除します) を使う必要があります。

249 `lb config` コマンドを以下のように実行してイメージを netboot 用に設定します:

250

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

251 ISO 及び HDD イメージとは対照的に netboot 自体ではクライアントに対してファイルシステムのイメージを提供しないため、ファイルを NFS 経由で提供する必要があります。`lb config` で異なるネットワークファイルシステムを選択することもできます。`--net-root-path` 及び `--net-root-server` オプションはそれぞれ、ブート時にファイルシステムのイメージが置かれる NFS サーバの位置とサーバを指定します。ネットワークやサーバに合う適切な値がセットされていることを確認してください。

252 それから `lb build` コマンドでイメージをビルドします:

253

```
# lb build
```

254 ネットワーク経由のブートでは、クライアントは通常 Ethernet

カードの EPROM にある小さなソフトウェアを実行します。このプログラムは DHCP リクエストを送り、IP アドレスと次に行うことについての情報を取得します。次の段階は通常、TFTP プロトコルを経由した高レベルブートローダの取得です。これには `pxelinux` や GRUB、さらには直接 Linux のようなオペレーティングシステムをブートすることもできます。

例えば生成された `live-image-i386.netboot.tar` アーカイブを `/srv/debian-live` ディレクトリに展開すると、`live/filesystem.squashfs` にファイルシステムのイメージ、カーネルや `initrd`、`pxelinux` ブートローダが `tftpbboot/` にあることがわかるでしょう。

ネットワーク経由でのブートをできるようにするにはサーバ上でサービスを 3 つ、DHCP サーバ、TFTP サーバ、NFS サーバを設定する必要があります。

4.8.1 DHCP サーバ

ネットワーク経由でブートするクライアントシステムに対して確実に IP アドレスを 1 つ与え、PXE ブートローダの位置を通知するようにネットワークの DHCP サーバを設定する必要があります。

イメージしやすいように `/etc/dhcp/dhcpd.conf` 設定ファイルで設定する ISC DHCP サーバ `isc-dhcp-server` 向けに書かれた例を示します:

```
# /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;
}

```

261 4.8.2 TFTP サーバ

262 これはカーネルと初期 RAM ディスクをシステム実行時に提供し
 263 ます。

263 *tftpd-hpa* パッケージをインストールすべきです。これはルート
 ディレクトリ、通常 `/srv/tftp` 内にある全ファイルを提供できま
 264 す。`/srv/debian-live/tftpboot` 内にあるファイルを提供させる
 には `root` で

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

265 を実行し、*tftp* サーバの新しいディレクトリについて聞かれたら
 回答します。

266 4.8.3 NFS サーバ

267 ゲストコンピュータが Linux カーネルをダウンロード、ブートし
 て `initrd` を読み込むと、NFS サーバ経由で Live ファイルシステムの
 イメージをマウントしようとします。

268 *nfs-kernel-server* パッケージをインストールする必要があるいま

す。

それから `/etc/exports` に

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

271 のような行を追記してファイルシステムのイメージを NFS 経由
 で利用できるようにし、この新しいエクスポートについて NFS
 サーバに知らせます:

```
# exportfs -rv
```

273 この 3 つのサービスの設定にはやや注意が必要かもしれません。
 全て協調して機能させるまでには忍耐がいくらか必要かもしれ
 ません。さらなる情報については <http://www.syslinux.org/wiki/index.php/PXELINUX> にある
 syslinux wiki や <http://d-i.alioth.debian.org/manual/ja.i386/ch04s05.html> にある
 Debian インストーラマニュアルの TFTP ネットブート節を見てください。
 方法はとても似ているので手助けになるかもしれません。

274 4.8.4 ネットワーク経由のブートをテストする方法

275 Netboot イメージの作成は *live-build* により簡単になりましたが、
 イメージを実際のマシンでテストするのは本当に時間がかかるも
 のとなるかもしれません。

276 日常を楽にするために仮想化を利用できます。

277 4.8.5 Qemu

278 • *qemu*、*bridge-utils*、*sudo* をインストールします。

279 /etc/qemu-ifup を編集します:

280

```
#!/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
```

281 grub-floppy-netboot を取得またはビルドします。

282 「-net nic,vlan=0 -net tap,vlan=0,ifname=tun0」を引数にして
gemu を実行します

283 4.9 ウェブブート

284 ウェブブートは手段としてインターネットを使い Live システム
をブートするための便利な方法です。ウェブブートの要件はと
ても少なくなっています。ある言い方をすれば必要なのはブー
トローダと初期 RAM ディスク、カーネルを収録したメディアで
す。別の言い方をすれば必要なのはファイルシステムを収録する
squashfs ファイルを置くウェブサーバです。

285 4.9.1 ウェブブートファイルの取得

286 いつものように、イメージを自分でビルドすることも、プロ
ジェクトのホームページ <<http://live-systems.org/>> から取得できるビ
ルド済みファイルを利用することも可能です。自身の必要に
応じて微調整ができるまでの初期テストにはビルド済みイメ
ージの利用が手軽でしょう。Live イメージのビルド後ならウェブ
ブートに必要なファイルは binary/live/ 下のビルドディレクト

リで見つけられるでしょう。ファイルは vmlinuz、initrd.img、
filesystem.squashfs と呼ばれます。

必要なファイルを既に存在する ISO イメージから抽出すること
も可能です。そのためには以下のようにしてそのイメージをルー
プバックマウントします:

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

288
289 ファイルは live/ ディレクトリで見つけられます。この例の場合
は /mnt/live/ になります。この方法にはそのイメージをマウン
トするのに root になる必要があるという欠点があります。しかし
これには簡単に定型処理、つまり自動化できるという利点があり
ます。

290 しかし疑いようもなく、ISO イメージからファイルを抽出すると
同時にウェブサーバにアップロードするのに最も簡単なのはミ
ッドナイトコマンドーや mc の利用でしょう。genisoimage パッケ
ージをインストールしていれば 2 ペインのファイルマネージャによ
り ISO ファイルの内容を確認しながらもう 1 つのペインでは ftp
経由でファイルをアップロードできます。この方法は手作業の介
入が必要とはなりませんが root 権限を必要としません。

291 4.9.2 ウェブブートイメージの起動

292 ユーザによってはウェブブートのテストに仮想化を好みますがこ
こでは以下の活用事例に合わせて実際のハードウェアについて言
及します。あくまで例だと思ってください。

293 ウェブブートイメージの起動は上記で示した構成要素、つまり
vmlinuz と initrd.img を USB メモリの live/ ディレクトリ以下
に書き込み、ブートローダとして syslinux をインストールすれ
ば十分です。そして USB メモリからブートしてブートオプショ

ンに `fetch=URL/ファイル/への/パス` を入力します。`live-boot` は `squashfs` ファイルを取得して RAM に格納します。こうして、ダウンロードした圧縮ファイルシステムを普通の Live システムとして使えるようになります。例えば:

294

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

295

活用事例: ウェブサーバがあり、`squashfs` ファイルが 2 つ、1 つは例えば `gnome` のようなデスクトップ環境一式を収録したものともう 1 つは復旧用が置かれているとします。あるマシンでグラフィカル環境が必要であれば USB メモリを差し込んで `gnome` 用イメージをウェブブートできます。後者のイメージに収録されている復旧用ツールが別のマシン等で必要になった場合は復旧用のイメージをウェブブートできます。

296 **Overview of tools**297 **5. Overview of tools**

298 This chapter contains an overview of the three main tools used in building live systems: *live-build*, *live-boot* and *live-config*.

299 **5.1 The live-build package**

300 *live-build* is a collection of scripts to build live systems. These scripts are also referred to as “commands” .

301 The idea behind *live-build* is to be a framework that uses a configuration directory to completely automate and customize all aspects of building a Live image.

302 Many concepts are similar to those used to build Debian packages with *debhelper*:

- 303 • The scripts have a central location for configuring their operation. In *debhelper*, this is the `debian/` subdirectory of a package tree. For example, `dh_install` will look, among others, for a file called `debian/install` to determine which files should exist in a particular binary package. In much the same way, *live-build* stores its configuration entirely under a `config/` subdirectory.
- 304 • The scripts are independent - that is to say, it is always safe to run each command.

305 Unlike *debhelper*, *live-build* provides the tools to generate a skeleton configuration directory. This could be considered to be similar to tools such as *dh-make*. For more information about these tools, read on, since the remainder of this section discusses the four most important commands. Note that the preceding `lb` is a generic wrapper for *live-build* commands.

306 • **lb config** : Responsible for initializing and configuring a Live system configuration directory. See The `lb config` command for more information.

• **lb build** : Responsible for starting a Live system build. See <The **lb build command**> for more information. 307

• **lb clean** : Responsible for removing parts of a Live system build. See <The **lb clean command**> for more information. 308

309 **5.1.1 The lb config command**

310 As discussed in <**live-build**>, the scripts that make up *live-build* read their configuration with the `source` command from a single directory named `config/`. As constructing this directory by hand would be time-consuming and error-prone, the `lb config` command can be used to create the initial skeleton configuration tree.

311 The `lb config` command creates the following directories inside `config/`: `hooks/`, `includes/`, several other includes subdirectories for each stage of the build process and `package-lists/`. The latter includes a list of several important live packages like *live-boot*, *live-config* and *live-config-sysvinit*.

312 Issuing `lb config` without any arguments completes the `config/` subdirectory which it populates with some default settings in configuration files, and two skeleton trees named `auto/` and `local/`.

```
313 $ lb config
[2014-04-25 17:14:34] lb config
P: Updating config tree for a debian/wheezy/i386 system
```

314 Using `lb config` without any arguments would be suitable for

users who need a very basic image, or who intend to provide a more complete configuration via `auto/config` later (see [<Managing a configuration>](#) for details).

315 Normally, you will want to specify some options. For example, to specify which package manager to use while building the image:

316

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

317 It is possible to specify many options, such as:

318

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

319 A full list of options is available in the `lb_config` man page.

320 5.1.2 The `lb build` command

321 The `lb build` command reads in your configuration from the `config/` directory. It then runs the lower level commands needed to build your Live system.

322 5.1.3 The `lb clean` command

323 It is the job of the `lb clean` command to remove various parts of a build so subsequent builds can start from a clean state. By default, `chroot`, `binary` and `source` stages are cleaned, but the cache is left intact. Also, individual stages can be cleaned. For example, if you have made changes that only affect the binary stage, use `lb clean --binary` prior to building a new binary. If your changes

invalidate the bootstrap and/or package caches, e.g. changes to `--mode`, `--architecture`, or `--bootstrap`, you must use `lb clean --purge`. See the `lb_clean` man page for a full list of options.

5.2 The live-boot package 324

live-boot is a collection of scripts providing hooks for the *initramfs-tools*, used to generate an *initramfs* capable of booting live systems, such as those created by *live-build*. This includes the live system ISOs, netboot tarballs, and USB stick images. 325

At boot time it will look for read-only media containing a `/live/` directory where a root filesystem (often a compressed filesystem image like `squashfs`) is stored. If found, it will create a writable environment, using `aufs`, for Debian like systems to boot from. 326

More information on initial ramfs in Debian can be found in the Debian Linux Kernel Handbook at [<http://kernel-handbook.alioth.debian.org/>](http://kernel-handbook.alioth.debian.org/) in the chapter on *initramfs*. 327

5.3 The live-config package 328

live-config consists of the scripts that run at boot time after *live-boot* to configure the live system automatically. It handles such tasks as setting the hostname, locales and timezone, creating the live user, inhibiting cron jobs and performing autologin of the live user. 329

330 設定の管理

331 6. 設定の管理

332 この章では *live-build* ソフトウェアと Live イメージ自体の両方について、最初の作成から継続的な改訂、継続的なリリースを通して Live 設定を管理する方法を説明します。

333 6.1 設定変更への対応

334 Live 設定が最初の試行で全て上手くいくのはまれです。一度だけビルドするのならコマンドラインから `lb config` オプションを渡すだけで済むかもしれませんが、満足のいくまでオプションを改訂してビルドを繰り返す方が標準的です。そうした変更を支援するには設定を確実に一貫した状態に保つ自動化スクリプトが必要となるでしょう。

335 6.1.1 自動化スクリプトを使う理由は？ それは何をやるもの？

336 `lb config` コマンドに渡されたオプションはされている他の多数のオプションと共にデフォルト値として `config/*` ファイルに保管されます。その後に `lb config` を実行した場合最初のオプションを基にしてデフォルトのオプションはリセットされません。そのため、例えば `--binary-images` に新しい値を指定して再び `lb config` を実行した場合以前指定していた種類のイメージに依存しているオプションのデフォルト値は新しく指定した種類のイメージでは使えなくなるかもしれません。そのファイルが読み取りや変更の対象からも外れているかもしれません。これを使うと 100 以上のオプションの値を保管するため、実際に指定されたオプションを誰でも確認できます。最後に、`lb config` を実行した後に *live-build* をアップグレードして、オプションの名前が変更されていた場合、`config/*` には古いオプションが有効ではなく

なった後に名付けられた変数も収録されます。

337 以上に挙げた理由により `auto/*` スクリプトにより楽が出来るようになります。このスクリプト群は `lb config` や `lb build`、`lb clean` コマンドの単純なラッパーで、設定の管理を支援するように設計されています。`auto/config` スクリプトは `lb config` コマンドと希望したオプションを全て保管し、`auto/clean` スクリプトは設定用変数値を収録するファイルを削除し、`auto/build` スクリプトは各ビルドの `build.log` を維持します。このスクリプト群はそれぞれ対応する `lb` コマンドの実行時に自動的に実行されます。このスクリプト群を利用することで設定が見やすくなり、改訂を越えて内部的に一貫した状態が維持されます。また、更新された文書を読んだ後に *live-build* をアップグレードすれば変更の必要があるオプションを識別、修正するのははるかに容易になるでしょう。

516 6.1.2 自動化スクリプトの使用例

339 便宜のため *live-build* には例の自動化シェルスクリプトが付属していてコピーして編集できるようになっています。デフォルトの設定を新しく作成してから例をコピーしましょう:

```
340 $ mkdir mylive && cd mylive && lb config
$ mkdir auto
$ cp /usr/share/doc/live-build/examples/auto/* auto/
```

341 `auto/config` を編集して、希望に合わせてオプションを追加します。例えば:

```
342 #!/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/ \
"${@}"
```

343 これ、lb config を使うたびに auto/config がそのオプションを基にして設定をリセットします。オプションを変更したいときには lb config に渡すのではなくこのファイルに書かれているものを編集します。lb clean を使うと auto/clean は config/* ファイルを、ビルドした他のものとあわせて削除します。最後に、lb build を使うとビルド時のログは auto/build により build.log に書かれます。

344 注意: ここで特別な noauto パラメータを使い、auto/config を別に呼び出すことのないようにして無限再帰を回避しています。編集時に不注意で削除することのないようにしてください。また、読みやすくするために上記の例で示したように lb config コマンドを複数行に分割する場合は次の行に続く各行末のバックスラッシュ(を忘れることのないようにしてください。

345 6.2 Git 経由で公開されている設定の複製

346 lb config --config オプションを使って Live システムの設定を収録している Git リポジトリを複製します。Live システムプロジェクトにより保守されている設定を基にしたい場合は <http://live-systems.org/gitweb/> の Packages カテゴリーの live-images という名前のリポジトリに目を通してみてください。このリポジトリには Live システムの **ビルド済みイメージ** 用の設定を収録しています。

347 例えばレスキュー用のイメージをビルドするには live-images リポジトリを使って

348

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

のようにし、必要に応じて auto/config やその他 config ツリーにあるものを必要なだけ編集します。例えば非公式の non-free ビルド済みイメージは単純に --archive-areas “main contrib non-free” を追加することで作成されます。

オプションとして Git 設定でショートカットを定義することもできます。\${HOME}/.gitconfig に

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

を追加すると live-systems.org にある git リポジトリのアドレスを指定する必要があるところで lso: を使えるようになります。さらにオプションで末尾の .git も省くと、この設定を使って新しいイメージの作成を始めるのはこれだけ簡単になります:

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

live-images リポジトリ全体を複製すると数種類のイメージの設定を取得します。最初のイメージが出来上がってから異なるイメージをビルドしたい場合は別のディレクトリに移動して繰り返し、必要に応じて変更を加えてください。

どの場合も、イメージのビルド lb build は毎回スーパーユーザーで行う必要があることを覚えておいてください。

356 収録内容の独自化

357 7. 独自化の概要

358 この章では Live システムを独自化できる様々な方法について概要を示します。

359 7.1 ビルド時とブート時の設定

360 Live システムの設定オプションはビルド時に適用されるビルド時オプションとブート時に適用されるブート時オプションとに分けられます。ブート時オプションはさらに、*live-boot* パッケージにより適用され、ブートの早い段階で起きるものと *live-config* パッケージにより適用され、ブートの遅い段階で起きるものとに分けられます。ブート時オプションはどれも、ユーザがブートプロンプトで指定することで変更できます。イメージは、デフォルトのブートパラメータを指定してビルドし、デフォルト値を全て適応する場合オプションをユーザが何も指定せずに普通に Live システムを直接ブートするようにもできます。特に、`lb --bootappend-live` への引数は設定の維持やキーボードレイアウト、タイムゾーン等、Live システムのカーネルコマンドラインオプションのデフォルト値で構成されます。例については **<ロケールと言語の独自化>** をご覧ください。361 Build-time configuration options are described in the `lb config man` pages. Boot-time options are described in the man pages for *live-boot* and *live-config*. Although the *live-boot* and *live-config* packages are installed within the live system you are building, it is recommended that you also install them on your build system for easy reference when you are working on your configuration. It is safe to do so, as none of the scripts contained within them are executed unless the system is configured as a live system.

362 7.2 ビルド段階

363 ビルドプロセスは段階ごとに分けられ、様々な独自化がそれぞれ順に適用されます。実行の最初の段階は `*{パッケージ収集}*` 段階です。この初期段階では `chroot` ディレクトリを作成して Debian システムの骨子を構成するパッケージを集めます。引き続いて `*{chroot}*` 段階があり、`chroot` ディレクトリの構成を完了させ、他の内容とともに設定に列挙されているパッケージを全て収集します。収録内容の独自化はほとんどがこの段階で起こります。Live イメージの準備の最終段階は `*{バイナリ}*` 段階で、ブート可能なイメージをビルドします。`chroot` ディレクトリの内容を使って Live システムのルートファイルシステムを作成し、インストーラと対象メディアの Live システムのファイルシステム外に配置する、他の追加の内容を全て収録します。Live イメージをビルドした後は、有効化されている場合はソースの tar アーカイブを `*{ソース}*` 段階で作成します。364 各段階で、コマンドの適用には特定の順序があります。そのように配置することで、独自化を合理的に階層化できるようになります。例えば `chroot` 段階ではどのパッケージをインストールするよりも前に `preseed` が適用され、ローカルに収録したどのファイルをコピーするよりも前にパッケージをインストールし、フックはその後に、収録内容を全て配置してから実行されます。365 7.3 ファイルによる `lb config` の補完366 `lb config` は設定の骨格を `config/` ディレクトリに作成しますが、目標を実現するには `config/` サブディレクトリ以下に追加のファイルを提供する必要があるかもしれません。設定のどこにファイルを置くかにより、Live システムのファイルシステムやバイナリイメージのファイルシステムにコピーされるか、コマンドラインオプションとして渡す方法では扱いにくいビルド時のシステム設定を提供することになります。独自のパッケージ一覧やネットワーク、あるいはビルド時またはブート時に実行するフックスク

リプト等を収録し、debian-live は既にかかなりの柔軟性がありますが、自身のコードでそれを後押しすることができます。

367 7.4 独自化タスク

368 以下の章ではユーザがよく行う類の独自化タスクをほんの一部ですがまとめています: <インストールするパッケージの独自化> <収録内容の独自化> <ロケールと言語の独自化>

369 インストールするパッケージの独自化

370 8. インストールするパッケージの独自化

371 恐らく Live システムの最も基本的な独自化はイメージに収録するパッケージの選択でしょう。この章では *live-build* でのパッケージのインストールの独自化のためのビルド時の様々なオプションを見ていきます。イメージへのインストールに利用可能なパッケージに関して最も影響が大きい選択はディストリビューションとアーカイブ領域です。まともなダウンロード速度を確保するため、近いディストリビューションミラーを選択してください。backports や experimental あるいは独自のパッケージのある自身専用のリポジトリを追加することもできます。また、ファイルを直接パッケージとして収録することもできます。特定のデスクトップや言語のパッケージ等、多数の関連パッケージを同時にインストールするメタパッケージを含め、パッケージ一覧は定義できます。最後に、ビルドでパッケージをインストールするときに *apt* や好みにより *aptitude* を制御するオプションもいくつかあります。プロキシを使っていたり、推奨パッケージのインストールを無効にして容量を節約したい、インストールするパッケージのバージョンを APT ピン経由で制御する必要がある、等便利だと思える場面があるかもしれません。

372 8.1 パッケージソース

373 8.1.1 ディストリビューション、アーカイブ領域とモード

374 ディストリビューションの選択は Live イメージへの収録に利用できるパッケージに最も影響があります。コード名を指定してください。**jessie** バージョンの *live-build* では **jessie** がデフォルトになっています。現在アーカイブにある任意のディストリビューションをコード名で指定できます (詳細については [条件](#) 参照)。`--distribution` オプションはアーカイブ内のパッケージソース

に影響するだけでなく、サポートしている各ディストリビューションをビルドするのに必要な動作をするよう *live-build* に指示します。例えば `*{unstable}*` リリースである `sid` に対してビルドする場合は:

```
375 $ lb config --distribution sid
```

376 のように指定します。ディストリビューションアーカイブ内で、アーカイブ領域はアーカイブを大きく分類します。Debian では `#{main}#`、`contrib`、`non-free` となっています。main に収録されるソフトウェアだけが Debian ディストリビューションの一部であり、したがってそれがデフォルトとなっています。複数指定することもできます。例えば

```
377 $ lb config --archive-areas "main contrib non-free"
```

378 Debian 派生物によっては `--mode` オプションの実験的サポートが利用できるものがあります。このオプションは Debian または未知のシステムでビルドしている場合にのみデフォルトで `debian` がセットされています。サポートしている派生物のどれかで `lb config` を実行した場合のデフォルト値はその派生物のイメージを作成するための値になります。`lb config` を例えば `ubuntu` モードで実行すると Debian 向けに代えて指定された派生物のディストリビューション名やアーカイブ領域がサポートされます。このモードはその派生物に合うように *live-build* の挙動も変更します。

379 注意: モードを追加したプロジェクトの設定については主にそのオプションのサポートユーザの担当です。Live システムプロジェクトは最善の努力をもって開発サポートを提供はしますが、私たちは派生物を自ら開発あるいはサポートしているわけではない

め、あくまで派生物プロジェクトからのフィードバックが基になります。

380 8.1.2 ディストリビューションミラー

381 Debian アーカイブは世界中の巨大なネットワークミラーにまたがって複製されているため、各地域の人が最高のダウンロード速度を求めて近いミラーを選択できます。--mirror-* オプションはそれぞれ、ビルドの様々な段階でどのディストリビューションミラーを利用するのかを決定します。<ビルド段階>の繰り返しになりますが *{パッケージ収集}* 段階は最小限のシステムで *debootstrap* により最初に *chroot* を構成する段階、**chroot** 段階は *chroot* を使って Live システムのファイルシステムをビルドする段階です。ミラーはこの各段階ごとにそれぞれのオプションで指定するため *{バイナリ}* の段階では --mirror-binary や --mirror-binary-security の値が採用され、早い段階で使っていたミラーは置き換えられることとなります。

382 8.1.3 ビルド時に利用するディストリビューションミラー

383 ビルド時に利用するディストリビューションミラーにローカルミラーを指定するには、--mirror-bootstrap、--mirror-chroot-security、--mirror-chroot-backports を以下のように指定するだけです。

```
384 $ lb config --mirror-bootstrap http://localhost/debian/ \
    --mirror-chroot-security http://localhost/debian-security/ \
    --mirror-chroot-backports http://localhost/debian-backports/
```

385 --mirror-chroot で指定する *chroot* ミラーのデフォルト値は --mirror-bootstrap の値になっています。

8.1.4 実行時に利用するディストリビューションミラー

387 --mirror-binary* オプションはバイナリイメージ中のディストリビューションミラーの位置を決定します。このオプションは Live システムの実行中に追加のパッケージをインストールする際に利用できます。デフォルトは [http.debian.net](http://debian.net) で、利用できるミラーの中から特にユーザの IP アドレスを基にして地理的に近いミラーを選択するサービスになっています。これはその Live システムを利用するユーザにとって最適なミラーを予測できない場合に適切な選択です。以下の例に示すように自分専用の値を指定することもできます。この設定でビルドされたイメージはその“ミラー”が到達可能なネットワークにいるユーザにとってのみ適します。

```
388 $ lb config --mirror-binary http://mirror/debian/ \
    --mirror-binary-security http://mirror/debian-security/ \
    --mirror-binary-backports http://mirror/debian-backports/
```

8.1.5 追加リポジトリ

389 You may add more repositories, broadening your package choices beyond what is available in your target distribution. These may be, for example, for backports, experimental or custom packages. To configure additional repositories, create config/archives/your-repository.list.chroot, and/ or config/archives/your-repository.list.binary files. As with the --mirror-* options, these govern the repositories used in the **chroot** stage when building the image, and in the **binary** stage, i.e. for use when running the live system.

391 For example, config/archives/live.list.chroot allows you to install packages from the debian-live snapshot repository at live

system build time.

392

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

393

If you add the same line to `config/archives/live.list.binary`, the repository will be added to your live system's `/etc/apt/sources.list.d/` directory.

394

If such files exist, they will be picked up automatically.

395

You should also put the GPG key used to sign the repository into `config/archives/your-repository.key.{binary, chroot}` files.

396

Should you need custom APT pinning, such APT preferences snippets can be placed in `config/archives/your-repository.pref.{binary, chroot}` files and will be automatically added to your live system's `/etc/apt/preferences.d/` directory.

397

8.2 Choosing packages to install

398

There are a number of ways to choose which packages *live-build* will install in your image, covering a variety of different needs. You can simply name individual packages to install in a package list. You can also use metapackages in those lists, or select them using package control file fields. And finally, you may place package files in your `config/` tree, which is well suited to testing of new or experimental packages before they are available from a repository.

399

8.2.1 Package lists

400

Package lists are a powerful way of expressing which packages

should be installed. The list syntax supports conditional sections which makes it easy to build lists and adapt them for use in multiple configurations. Package names may also be injected into the list using shell helpers at build time.

Note: The behaviour of *live-build* when specifying a package that does not exist is determined by your choice of APT utility. See [<Choosing apt or aptitude>](#) for more details.

401

8.2.2 Using metapackages

402

The simplest way to populate your package list is to use a task metapackage maintained by your distribution. For example:

403

404

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

This supercedes the older predefined list method supported in *live-build* 2.x. Unlike predefined lists, task metapackages are not specific to the Live System project. Instead, they are maintained by specialist working groups within the distribution and therefore reflect the consensus of each group about which packages best serve the needs of the intended users. They also cover a much broader range of use cases than the predefined lists they replace.

405

All task metapackages are prefixed `task-`, so a quick way to determine which are available (though it may contain a handful of false hits that match the name but aren't metapackages) is to match on the package name with:

406

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

407

408 In addition to these, you will find other metapackages with vari- 414
 ous purposes. Some are subsets of broader task packages, like
 gnome-core, while others are individual specialized parts of a De-
 bian Pure Blend, such as the education-* metapackages. To list
 all metapackages in the archive, install the debtags package and
 list all packages with the role::metapackage tag as follows:

409

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

410 8.2.3 Local package lists

411 Whether you list metapackages, individual packages, or a combi-
 nation of both, all local package lists are stored in config/package-
 lists/. Since more than one list can be used, this lends itself well
 to modular designs. For example, you may decide to devote one
 list to a particular choice of desktop, another to a collection of re-
 lated packages that might as easily be used on top of a different
 desktop. This allows you to experiment with different combinations
 of sets of packages with a minimum of fuss, sharing common lists
 between different live image projects.

412 Package lists that exist in this directory need to have a .list suf-
 fix in order to be processed, and then an additional stage suffix,
 .chroot or .binary to indicate which stage the list is for.

413 **Note:** If you don't specify the stage suffix, the list will be used for
 both stages. Normally, you want to specify .list.chroot so that
 the packages will only be installed in the live filesystem and not
 have an extra copy of the .deb placed on the medium.

8.2.4 Local binary package lists

To make a binary stage list, place a file suffixed with .list.binary 415
 in config/package-lists/. These packages are not installed in
 the live filesystem, but are included on the live medium under
 pool/. You would typically use such a list with one of the non-live
 installer variants. As mentioned above, if you want this list to be
 the same as your chroot stage list, simply use the .list suffix by
 itself.

8.2.5 Generated package lists 416

It sometimes happens that the best way to compose a list is to 417
 generate it with a script. Any line starting with an exclamation point
 indicates a command to be executed within the chroot when the
 image is built. For example, one might include the line ! grep-
 aptavail -n -sPackage -FPriority standard |sort in a pack-
 age list to produce a sorted list of available packages with Pri-
 ority: standard.

In fact, selecting packages with the grep-aptavail command (from 418
 the dctrl-tools package) is so useful that live-build provides a
 Packages helper script as a convenience. This script takes two
 arguments: field and pattern. Thus, you can create a list with
 the following contents:

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

419

8.2.6 Using conditionals inside package lists

421 Any of the *live-build* configuration variables stored in `config/*` (minus the `LB_` prefix) may be used in conditional statements in package lists. Generally, this means any `lb config` option uppercased and with dashes changed to underscores. But in practice, it is only the ones that influence package selection that make sense, such as `DISTRIBUTION`, `ARCHITECTURES` or `ARCHIVE_AREAS`.

422 For example, to install `ia32-libs` if the `--architectures amd64` is specified:

423

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

424 You may test for any one of a number of values, e.g. to install `memtest86+` if either `--architectures i386` or `--architectures amd64` is specified:

425

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

426 You may also test against variables that may contain more than one value, e.g. to install `vrms` if either `contrib` or `non-free` is specified via `--archive-areas`:

427

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

420 The nesting of conditionals is not supported.

428

8.2.7 Removing packages at install time

429

You can list packages in files with `.list.chroot_live` and `.list.chroot_install` suffixes inside the `config/package-lists` directory. If both a live and an install list exist, the packages in the `.list.chroot_live` list are removed with a hook after the installation (if the user uses the installer). The packages in the `.list.chroot_install` list are present both in the live system and in the installed system. This is a special tweak for the installer and may be useful if you have `--debian-installer live` set in your config, and wish to remove live system-specific packages at install time.

430

8.2.8 Desktop and language tasks

431

Desktop and language tasks are special cases that need some extra planning and configuration. Live images are different from Debian Installer images in this respect. In the Debian Installer, if the medium was prepared for a particular desktop environment flavour, the corresponding task will be automatically installed. Thus, there are internal `gnome-desktop`, `kde-desktop`, `lxde-desktop` and `xfce-desktop` tasks, none of which are offered in `tasksel`'s menu. Likewise, there are no menu entries for tasks for languages, but the user's language choice during the install influences the selection of corresponding language tasks.

432

When developing a desktop live image, the image typically boots directly to a working desktop, the choices of both desktop and default language having been made at build time, not at run time as in the case of the Debian Installer. That's not to say that a live image couldn't be built to support multiple desktops or multiple languages

433

and offer the user a choice, but that is not *live-build*'s default behaviour.

434 Because there is no provision made automatically for language tasks, which include such things as language-specific fonts and input-method packages, if you want them, you need to specify them in your configuration. For example, a GNOME desktop image containing support for German might include these task metapackages:

435

```
$ 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
```

436 8.2.9 Kernel flavour and version

437 One or more kernel flavours will be included in your image by default, depending on the architecture. You can choose different flavours via the `--linux-flavours` option. Each flavour is suffixed to the default stub `linux-image` to form each metapackage name which in turn depends on an exact kernel package to be included in your image.

438 Thus by default, an amd64 architecture image will include the `linux-image-amd64` flavour metapackage, and an i386 architecture image will include the `linux-image-486` and `linux-image-686-pae` metapackages. At time of writing, these packages depend on `linux-image-3.2.0-4-amd64`, `linux-image-3.2.0-4-486` and `linux-image-3.2.0-4-686-pae`, respectively.

439 When more than one kernel package version is available in your configured archives, you can specify a different kernel package

name stub with the `--linux-packages` option. For example, supposing you are building an amd64 architecture image and add the experimental archive for testing purposes so you can install the `linux-image-3.7-trunk-amd64` kernel. You would configure that image as follows:

440

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

8.2.10 Custom kernels

441

442 You can build and include your own custom kernels, so long as they are integrated within the Debian package management system. The *live-build* system does not support kernels not built as `.deb` packages.

443 The proper and recommended way to deploy your own kernel packages is to follow the instructions in the `kernel-handbook`. Remember to modify the ABI and flavour suffixes appropriately, then include a complete build of the `linux` and matching `linux-latest` packages in your repository.

444 If you opt to build the kernel packages without the matching metapackages, you need to specify an appropriate `--linux-packages` stub as discussed in [Kernel flavour and version](#). As we explain in [Installing modified or third-party packages](#), it is best if you include your custom kernel packages in your own repository, though the alternatives discussed in that section work as well.

445 It is beyond the scope of this document to give advice on how to customize your kernel. However, you must at least ensure your configuration satisfies these minimum requirements:

- 446 • Use an initial ramdisk.
- 447 • Include the union filesystem module (i.e. usually aufs).
- 448 • Include any other filesystem modules required by your configuration (i.e. usually squashfs).

449 8.3 Installing modified or third-party packages

450 While it is against the philosophy of a live system, it may sometimes be necessary to build a live system with modified versions of packages that are in the Debian repository. This may be to modify or support additional features, languages and branding, or even to remove elements of existing packages that are undesirable. Similarly, “third-party” packages may be used to add bespoke and/or proprietary functionality.

451 This section does not cover advice regarding building or maintaining modified packages. Joachim Breitner’s ‘How to fork privately’ method from <http://www.joachim-breitner.de/blog/archives/282-How-to-fork-privately.html> may be of interest, however. The creation of bespoke packages is covered in the Debian New Maintainers’ Guide at <http://www.debian.org/doc/maint-guide/> and elsewhere.

452 There are two ways of installing modified custom packages:

- 453 • `packages.chroot`
- 454 • Using a custom APT repository

455 Using `packages.chroot` is simpler to achieve and useful for “one-off” customizations but has a number of drawbacks, while using a custom APT repository is more time-consuming to set up.

456 8.3.1 Using `packages.chroot` to install custom packages

457 To install a custom package, simply copy it to the `config/-`

`packages.chroot/` directory. Packages that are inside this directory will be automatically installed into the live system during build - you do not need to specify them elsewhere.

Packages **must** be named in the prescribed way. One simple way to do this is to use `dpkg-name`. 458

Using `packages.chroot` for installation of custom packages has disadvantages: 459

- 460 • It is not possible to use secure APT.
- 461 • You must install all appropriate packages in the `config/packages.chroot/` directory.
- 462 • It does not lend itself to storing live system configurations in revision control.

463 8.3.2 Using an APT repository to install custom packages

464 Unlike using `packages.chroot`, when using a custom APT repository you must ensure that you specify the packages elsewhere. See [Choosing packages to install](#) for details.

465 While it may seem unnecessary effort to create an APT repository to install custom packages, the infrastructure can be easily re-used at a later date to offer updates of the modified packages.

466 8.3.3 Custom packages and APT

467 *live-build* uses APT to install all packages into the live system so will therefore inherit behaviours from this program. One relevant example is that (assuming a default configuration) given a package available in two different repositories with different version numbers, APT will elect to install the package with the higher version number.

468 Because of this, you may wish to increment the version number in
your custom packages' `debian/changelog` files to ensure that your
modified version is installed over one in the official Debian repos-
itories. This may also be achieved by altering the live system's
APT pinning preferences - see [<APT pinning>](#) for more informa-
tion.

469 8.4 Configuring APT at build time

470 You can configure APT through a number of options applied only
at build time. (APT configuration used in the running live system
may be configured in the normal way for live system contents, that
is, by including the appropriate configurations through `config/-
includes.chroot/`.) For a complete list, look for options starting
with `apt` in the `lb_config` man page.

471 8.4.1 Choosing `apt` or `aptitude`

472 You can elect to use either *apt* or *aptitude* when installing pack-
ages at build time. Which utility is used is governed by the `--apt`
argument to `lb config`. Choose the method implementing the pre-
ferred behaviour for package installation, the notable difference be-
ing how missing packages are handled.

- 473 • `apt`: With this method, if a missing package is specified, the pack-
age installation will fail. This is the default setting.
- 474 • `aptitude`: With this method, if a missing package is specified,
the package installation will succeed.

475 8.4.2 Using a proxy with APT

476 One commonly required APT configuration is to deal with build-
ing an image behind a proxy. You may specify your APT proxy

with the `--apt-ftp-proxy` or `--apt-http-proxy` options as needed,
e.g.

```
477 $ lb config --apt-http-proxy http://proxy/
```

478 8.4.3 Tweaking APT to save space

479 You may find yourself needing to save some space on the image
medium, in which case one or the other or both of the following
options may be of interest.

480 If you don't want to include APT indices in the image, you can omit
those with:

```
481 $ lb config --apt-indices false
```

482 This will not influence the entries in `/etc/apt/sources.list`, but
merely whether `/var/lib/apt` contains the indices files or not. The
tradeoff is that APT needs those indices in order to operate in the
live system, so before performing `apt-cache search` or `apt-get`
`install`, for instance, the user must `apt-get update` first to create
those indices.

483 If you find the installation of recommended packages bloats your
image too much, provided you are prepared to deal with the con-
sequences discussed below, you may disable that default option
of APT with:

```
484 $ lb config --apt-recommends false
```

485 The most important consequence of turning off recommends is that live-boot and live-config themselves recommend some packages that provide important functionality used by most Live configurations, such as user-setup which live-config recommends and is used to create the live user. In all but the most exceptional circumstances you need to add back at least some of these recommends to your package lists or else your image will not work as expected, if at all. Look at the recommended packages for each of the live-* packages included in your build and if you are not certain you can omit them, add them back into your package lists.

486 The more general consequence is that if you don't install recommended packages for any given package, that is, "packages that would be found together with this one in all but unusual installations" (Debian Policy Manual, section 7.2), some packages that users of your Live system actually need may be omitted. Therefore, we suggest you review the difference turning off recommends makes to your packages list (see the binary.packages file generated by lb build) and re-include in your list any missing packages that you still want installed. Alternatively, if you find you only want a small number of recommended packages left out, leave recommends enabled and set a negative APT pin priority on selected packages to prevent them from being installed, as explained in <APT pinning>.

487 8.4.4 Passing options to apt or aptitude

488 If there is not a lb config option to alter APT's behaviour in the way you need, use --apt-options or --aptitude-options to pass any options through to your configured APT tool. See the man pages for apt and aptitude for details. Note that both options have default values that you will need to retain in addition to any overrides you may provide. So, for example, suppose you have included something from snapshot.debian.org for testing purposes

and want to specify Acquire::Check-Valid-Until=false to make APT happy with the stale Release file, you would do so as per the following example, appending the new option after the default value --yes:

```
489 $ lb config --apt-options "--yes -oAcquire::Check-Valid-Until=false"
```

490 Please check the man pages to fully understand these options and when to use them. This is an example only and should not be construed as advice to configure your image this way. This option would not be appropriate for, say, a final release of a live image.

491 For more complicated APT configurations involving apt.conf options you might want to create a config/apt/apt.conf file instead. See also the other apt-* options for a few convenient shortcuts for frequently needed options.

492 8.4.5 APT pinning

493 For background, please first read the apt_preferences(5) man page. APT pinning can be configured either for build time, or else for run time. For the former, create config/archives/*.pref, config/archives/*.pref.chroot, and config/apt/preferences. For the latter, create config/includes.chroot/etc/apt/preferences.

494 Let's say you are building a **jessie** live system but need all the live packages that end up in the binary image to be installed from **sid** at build time. You need to add **sid** to your APT sources and pin the live packages from it higher, but all other packages from it lower, than the default priority. Thus, only the packages you want are installed from **sid** at build time and all others are taken from the

target system distribution, **jessie** . The following will accomplish this:

495

```
$ 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
```

496

Negative pin priorities will prevent a package from being installed, as in the case where you do not want a package that is recommended by another package. Suppose you are building an LXDE image using `task-lxde-desktop` in `config/package-lists/desktop.list.chroot`, but don't want the user prompted to store wifi passwords in the keyring. This metapackage depends on *lxde-core*, which recommends *gksu*, which in turn recommends *gnome-keyring*. So you want to omit the recommended *gnome-keyring* package. This can be done by adding the following stanza to `config/apt/preferences`:

497

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

498 収録内容の独自化

499 **9. 収録内容の独自化**

500 この章では収録するパッケージを単に選択だけにとどまらない、
 501 微調整まで含めた Live システムの収録内容の独自化について説
 502 明します。インクルードにより live システムイメージの任意の
 ファイルを追加、置換できるようになり、フックによりビルド時
 及びブート時の異なる段階で任意のコマンドを実行できるよう
 になり、preseed が debconf の質問に対する回答を提供すること
 でパッケージのインストール時に設定できるようになります。

501 **9.1 Includes**

502 理想的なのは変更されていないパッケージにより提供されるファ
 イルを Live システムで完全に収録することではありますが、ファ
 イルを使って内容をいくらか提供あるいは変更することが便利な
 こともあります。インクルードを使うと Live システムイメージ
 中の任意のファイルを追加 (または置換) することができるよ
 うになります。live-build ではこれを使う仕組みを 2 つ提供してい
 ます:

- 503 • Chroot ローカルインクルード: chroot/Live ファイルシステムに
 対してファイルの追加や置換ができるようになります。さらな
 る情報については、**〈Live/chroot ローカルインクルード〉**を見
 てください。
- 504 • バイナリローカルインクルード: バイナリイメージ中のファ
 イルの追加や置換ができるようになります。さらなる情報につ
 いては、**〈バイナリローカルインクルード〉**を見てください。

505 「Live」及び「バイナリ」イメージの違いについてのさらなる情
 報は、**〈用語〉**を見てください。

506 **9.1.1 Live/chroot ローカルインクルード**

507 Chroot ローカルインクルードを使って chroot/Live ファイルシ
 ステム中のファイルの追加や置換を行い、それを Live システムで
 利用することができます。代表的な使い方として Live システムで
 利用するユーザディレクトリ (/etc/skel) の骨格を構成させ、live
 ユーザのホームディレクトリを作成するということがあります。
 別の使い方としては設定ファイルを提供し、そのまま加工せずイ
 メージ中に追加または置換するということがあります。加工が必
 要な場合は **〈Live/chroot ローカルフック〉**を見てください。

508 ファイルを収録するには config/includes.chroot ディレクトリ
 に単純に追加します。このディレクトリが Live システムのルート
 ディレクトリ / に対応します。例えば Live システムにファイル
 /var/www/index.html を追加する場合:

```
509 $ mkdir -p config/includes.chroot/var/www
$ cp /path/to/my/index.html config/includes.chroot/var/www
```

それから設定は以下の配置になっているでしょう:

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

512 Chroot ローカルインクルードはパッケージがインストールされ
 た後にインストールされるので、パッケージによりインストール
 されたファイルは上書きされます。

513 **9.1.2 バイナリローカルインクルード**

514 文書やビデオ等の内容をメディアのファイルシステムに収録して、メディアを差し込んで Live システムをブートしなくてもすぐにアクセスできるようにするのにバイナリローカルインクルードを使えます。これは chroot ローカルインクルードと同様の方法で動作します。例えばファイル `~/video_demo.*` が live システムの実演ビデオで、リンク先の HTML 索引ページでそれを説明しているものと仮定しましょう。単純に内容を `config/includes.binary/` にコピーします:

515

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

516 これでファイルは live メディアのルートディレクトリに現れます。

517 **9.2 フック**

518 フックではビルドの chroot 及びバイナリの段階でコマンドを実行し、イメージを独自化できます。

519 **9.2.1 Live/chroot ローカルフック**

520 chroot の段階でコマンドを実行するにはファイル名末尾が `.hook.chroot` でコマンドを収録するフックスクリプトを `config/hooks/` ディレクトリに作成します。フックは残りの chroot 設定の適用後に chroot 内で実行されるため、フックの実行に必要なパッケージやファイルを全て確実に設定に収録することを忘れないようにしてください。代表的な chroot の様々な独自化タスクについて `/usr/share/doc/live-build/examples/hooks` で提供されている chroot フックスクリプトの例を確認してください。この

例からコピーやシンボリックリンクを作成して自分の設定で使えます。

521 **9.2.2 ブート時フック**

522 ブート時にコマンドを実行するために man ページの「独自化」節で説明されている `live-config` フックを提供することができます。`/lib/live/config/` で提供している `live-config` 独自のフックを、実行順を示す頭の番号に注意して調べてください。それから自分のフックに実行順を示す適切な番号を頭に付けて、`config/includes.chroot/lib/live/config/` 内の chroot ローカルインクルードか、**変更した、またはサードパーティのパッケージのインストール** で説明している独自パッケージとして提供してください。

523 **9.2.3 バイナリローカルフック**

524 バイナリ段階でコマンドを実行するには、コマンドを収録するフックスクリプトを、末尾に `.hook.binary` を付けて `config/hooks/` ディレクトリに作成します。このフックは他の `binary_checksums` を除いたバイナリコマンドを全て実行した後の、バイナリコマンドのほぼ最後に実行されます。フック内のコマンドは chroot 内で実行されるのではないため、ビルドツリー外のファイルを変更することのないように注意してください。変更するとビルドシステムが機能しなくなるかもしれません! 代表的なバイナリ独自化タスクについて `/usr/share/doc/live-build/examples/hooks` で提供されているバイナリフックスクリプトの例を確認してください。この例からコピーやシンボリックリンクを作成して自分の設定で使えます。

525 **9.3 Debconf 質問の preseed**

526 `config/preseed/` ディレクトリにある、末尾が段階 (`.chroot` か

.binary) に続いて .cfg で終わるファイルは debconf の preseed ファイルと見なされ、対応する段階で *live-build* により debconf-set-selections を使ってインストールされます。

527

debconf のさらなる情報については、*debconf* パッケージの `debconf(7)` を見てください。

528 **Customizing run time behaviours**529 **10. Customizing run time behaviours**

530 All configuration that is done during run time is done by *live-config*. Here are some of the most common options of *live-config* that users are interested in. A full list of all possibilities can be found in the man page of *live-config*.

531 **10.1 Customizing the live user**

532 One important consideration is that the live user is created by *live-boot* at boot time, not by *live-build* at build time. This not only influences where materials relating to the live user are introduced in your build, as discussed in [Live/chroot local includes](#), but also any groups and permissions associated with the live user.

533 You can specify additional groups that the live user will belong to by using any of the possibilities to configure *live-config*. For example, to add the live user to the `fuse` group, you can either add the following file in `config/includes.chroot/etc/live/config/user-setup.conf`:

534

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

535 or use `live-config.user-default-groups=audio,cdrom,dip,floppy,video,plugdev,netdev,powerdev,scanner,bluetooth,fuse` as a boot parameter.

536 It is also possible to change the default username “user” and the default password “live”. If you want to do that for any reason, you can easily achieve it as follows:

537 To change the default username you can simply specify it in your

config:

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

538 One possible way of changing the default password is by means of a hook as described in [Boot-time hooks](#). In order to do that you can use the “passwd” hook from `/usr/share/doc/live-config/examples/hooks`, prefix it accordingly (e.g. `2000-passwd`) and add it to `config/includes.chroot/lib/live/config/`

540 **10.2 Customizing locale and language**

541 When the live system boots, language is involved in two steps:

- the locale generation 542
- setting the keyboard configuration 543

544 The default locale when building a Live system is `locales=en_US.UTF-8`. To define the locale that should be generated, use the `locales` parameter in the `--bootappend-live` option of `lb config`, e.g.

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

545 Multiple locales may be specified as a comma-delimited list. 546

547 This parameter, as well as the keyboard configuration parameters indicated below, can also be used at the kernel command line. You can specify a locale by `language_country` (in which case the default encoding is used) or the full `language_country.encoding`

word. A list of supported locales and the encoding for each can be found in `/usr/share/i18n/SUPPORTED`.

548 Both the console and X keyboard configuration are performed by `live-config` using the `console-setup` package. To configure them, use the `keyboard-layouts`, `keyboard-variants`, `keyboard-options` and `keyboard-model` boot parameters via the `--bootappend-live` option. Valid options for these can be found in `/usr/share/X11/xkb/rules/base.lst`. To find layouts and variants for a given language, try searching for the English name of the language and/or the country where the language is spoken, e.g:

549

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

550 Note that each variant lists the layout to which it applies in the description.

551 Often, only the layout needs to be configured. For example, to get the locale files for German and Swiss German keyboard layout in X use:

552

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

553 However, for very specific use cases, you may wish to include other

parameters. For example, to set up a French system with a French-Dvorak layout (called Bepo) on a TypeMatrix EZ-Reach 2030 USB keyboard, use:

554

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

Multiple values may be specified as comma-delimited lists for each of the `keyboard-*` options, with the exception of `keyboard-model`, which accepts only one value. Please see the `keyboard(5)` man page for details and examples of `XKBMODEL`, `XKBLAYOUT`, `XKBVARIANT` and `XKBOPTIONS` variables. If multiple `keyboard-variants` values are given, they will be matched one-to-one with `keyboard-layouts` values (see `setxkbmap(1)` `-variant` option). Empty values are allowed; e.g. to define two layouts, the default being US QWERTY and the other being US Dvorak, use:

555

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

556

10.3 Persistence

557

A live cd paradigm is a pre-installed system which runs from read-only media, like a cdrom, where writes and modifications do not survive reboots of the host hardware which runs it.

558

A live system is a generalization of this paradigm and thus supports other media in addition to CDs; but still, in its default behaviour, it should be considered read-only and all the run-time evolutions of the system are lost at shutdown.

559

560 ‘Persistence’ is a common name for different kinds of solutions for saving across reboots some, or all, of this run-time evolution of the system. To understand how it works it would be handy to know that even if the system is booted and run from read-only media, modifications to the files and directories are written on writable media, typically a ram disk (tmpfs) and ram disks’ data do not survive reboots.

561 The data stored on this ramdisk should be saved on a writable persistent medium like local storage media, a network share or even a session of a multisession (re)writable CD/DVD. All these media are supported in live systems in different ways, and all but the last one require a special boot parameter to be specified at boot time: persistence.

562 If the boot parameter `persistence` is set (and `nopersistence` is not set), local storage media (e.g. hard disks, USB drives) will be probed for persistence volumes during boot. It is possible to restrict which types of persistence volumes to use by specifying certain boot parameters described in the *live-boot(7)* man page. A persistence volume is any of the following:

- 563 • a partition, identified by its GPT name.
- 564 • a filesystem, identified by its filesystem label.
- 565 • an image file located on the root of any readable filesystem (even an NTFS partition of a foreign OS), identified by its filename.

566 The volume label for overlays must be `persistence` but it will be ignored unless it contains in its root a file named `persistence.conf` which is used to fully customize the volume’s persistence, this is to say, specifying the directories that you want to save in your persistence volume after a reboot. See [<The persistence.conf file>](#) for more details.

567 Here are some examples of how to prepare a volume to be used

for persistence. It can be, for instance, an ext4 partition on a hard disk or on a usb key created with, e.g.:

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

See also [<Using the space left on a USB stick>](#).

If you already have a partition on your device, you could just change the label with one of the following:

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

Here’s an example of how to create an ext4-based image file to be used for persistence:

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

Once the image file is created, as an example, to make `/usr` persistent but only saving the changes you make to that directory and not all the contents of `/usr`, you can use the “union” option. If the image file is located in your home directory, copy it to the root of your hard drive’s filesystem and mount it in `/mnt` as follows:

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

Then, create the `persistence.conf` file adding content and un-

mount the image file.

577

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

578 Now, reboot into your live medium with the boot parameter “persistence” .

579 10.3.1 The persistence.conf file

580 A volume with the label `persistence` must be configured by means of the `persistence.conf` file to make arbitrary directories persistent. That file, located on the volume’s filesystem root, controls which directories it makes persistent, and in which way.

581 How custom overlay mounts are configured is described in full detail in the `persistence.conf(5)` man page, but a simple example should be sufficient for most uses. Let’s say we want to make our home directory and APT cache persistent in an ext4 filesystem on the `/dev/sdb1` partition:

582

```
# 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
```

583 Then we reboot. During the first boot the contents of `/home` and `/var/cache/apt` will be copied into the persistence volume, and from then on all changes to these directories will live in the persistence volume. Please note that any paths listed in the `persistence.conf` file cannot contain white spaces or the special `.` and

`..` path components. Also, neither `/lib`, `/lib/live` (or any of their sub-directories) nor `/` can be made persistent using custom mounts. As a workaround for this limitation you can add `/ union` to your `persistence.conf` file to achieve full persistence.

10.3.2 Using more than one persistence store

584

585 There are different methods of using multiple persistence store for different use cases. For instance, using several volumes at the same time or selecting only one, among various, for very specific purposes.

586 Several different custom overlay volumes (with their own `persistence.conf` files) can be used at the same time, but if several volumes make the same directory persistent, only one of them will be used. If any two mounts are “nested” (i.e. one is a sub-directory of the other) the parent will be mounted before the child so no mount will be hidden by the other. Nested custom mounts are problematic if they are listed in the same `persistence.conf` file. See the `persistence.conf(5)` man page for how to handle that case if you really need it (hint: you usually don’t).

587 One possible use case: If you wish to store the user data i.e. `/home` and the superuser data i.e. `/root` in different partitions, create two partitions with the `persistence` label and add a `persistence.conf` file in each one like this, `# echo “/home” > persistence.conf` for the first partition that will save the user’s files and `# echo “/root” > persistence.conf` for the second partition which will store the superuser’s files. Finally, use the `persistence` boot parameter.

588 If a user would need multiple persistence store of the same type for different locations or testing, such as `private` and `work`, the boot parameter `persistence-label` used in conjunction with the boot parameter `persistence` will allow for multiple but unique persistence media. An example would be if a user wanted to

use a persistence partition labeled `private` for personal data like browser bookmarks or other types, they would use the boot parameters: `persistence persistence-label=private`. And to store work related data, like documents, research projects or other types, they would use the boot parameters: `persistence persistence-label=work`.

589 It is important to remember that each of these volumes, `private` and `work`, also needs a `persistence.conf` file in its root. The *live-boot* man page contains more information about how to use these labels with legacy names.

590 10.4 Using persistence with encryption

591 Using the persistence feature means that some sensible data might get exposed to risk. Especially if the persistent data is stored on a portable device such as a usb stick or an external hard drive. That is when encryption comes in handy. Even if the entire procedure might seem complicated because of the number of steps to be taken, it is really easy to handle encrypted partitions with *live-boot*. In order to use **luks**, which is the supported encryption type, you need to install *cryptsetup* both on the machine you are creating the encrypted partition with and also in the live system you are going to use the encrypted persistent partition with.

592 To install *cryptsetup* on your machine:

593

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

594 To install *cryptsetup* in your live system, add it to your package-lists:

595

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

Once you have your live system with *cryptsetup*, you basically only need to create a new partition, encrypt it and boot with the `persistence` and `persistence-encryption=luks` parameters. We could have already anticipated this step and added the boot parameters following the usual procedure:

596

597

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

Let's go into the details for all of those who are not familiar with encryption. In the following example we are going to use a partition on a usb stick which corresponds to `/dev/sdc2`. Please be warned that you need to determine which partition is the one you are going to use in your specific case.

598

The first step is plugging in your usb stick and determine which device it is. The recommended method of listing devices in *live-manual* is using `ls -l /dev/disk/by-id`. After that, create a new partition and then, encrypt it with a passphrase as follows:

599

600

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

Then open the luks partition in the virtual device mapper. Use any name you like. We use **live** here as an example:

601

602

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

603 The next step is filling the device with zeros before creating the 613
604 filesystem:

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

605 Now, we are ready to create the filesystem. Notice that we are
606 adding the label persistence so that the device is mounted as per-
607 sistence store at boot time.

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

607 To continue with our setup, we need to mount the device, for ex-
608 ample in /mnt.

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

609 And create the persistence.conf file in the root of the partition.
610 This is, as explained before, strictly necessary. See <The persis-
611 tence.conf file>.

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

611 Then unmount the mount point:

612

```
# umount /mnt
```

And optionally, although it might be a good way of securing the data
we have just added to the partition, we can close the device:

614

```
# cryptsetup luksClose live
```

615 Let's summarize the process. So far, we have created an encryp-
616 tion capable live system, which can be copied to a usb stick as ex-
617 plained in <Copying an ISO hybrid image to a USB stick>. We have
618 also created an encrypted partition, which can be located in the
619 same usb stick to carry it around and we have configured the en-
620 crypted partition to be used as persistence store. So now, we only
621 need to boot the live system. At boot time, *live-boot* will prompt
622 us for the passphrase and will mount the encrypted partition to be
623 used for persistence.

616 バイナリイメージの独自化

617 11. バイナリイメージの独自化

618 11.1 ブートローダ

619 *live-build* は *syslinux* や (イメージの種類により) その派生物の一部をブートローダとしてデフォルトで利用します。これは要件に合わせて簡単に独自化できます。

620 全面的なテーマを使うには `/usr/share/live/build/bootloaders` を `config/bootloaders` にコピーしてその中のファイルを編集します。サポートしているブートローダ全部の設定変更を望まない場合は、ブートローダの 1 つ、例えば `config/bootloaders/-isolinux` にある **isolinux** だけを局所的に地域化したものを提供するのでも、活用方法によりますが十分です。

621 のようになります。デフォルトのテーマを変更してブートメニューとともに表示される背景画像に個別のものを使いたい場合は 640x480 ピクセルの画像を `splash.png` というファイル名で追加し、`splash.svg` ファイルを削除します。

622 変更を加えるに至る要因は多々あります。例えば *syslinux* 派生物ではデフォルトでタイムアウト時間が 0 に設定されていて、この場合はスプラッシュ画面でキーが押されるまでいつまでも一時停止状態で止まっているということになります。

623 デフォルトの `iso-hybrid` イメージのブート時のタイムアウト時間を変更する方法は、デフォルトの **isolinux.cfg** ファイルを編集して 1/10 秒単位でタイムアウト時間を指定するだけです。5 秒後にブートするように **isolinux.cfg** を変更する場合は

624

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

timeout 50

11.2 ISO メタ情報

ISO9660 バイナリイメージの作成時に以下のオプションを使って、テキストの様々なメタ情報をイメージに追加できます。これはイメージのバージョンや設定をブートせずに簡単に識別する手助けとなります。

- 627 • `LB_ISO_APPLICATION/--iso-application NAME`: これにはイメージ上に置かれる、アプリケーションの説明を記述します。最大長は 128 文字です。
- 628 • `LB_ISO_PREPARER/--iso-preparer NAME`: これはイメージの作成者を説明し、通常連絡先の詳細をいくらか含めます。このオプションのデフォルト値は作成に利用した *live-build* のバージョンで、後でデバッグするときに手がかりとなることを意図しています。最大長は 128 文字です。
- 629 • `LB_ISO_PUBLISHER/--iso-publisher NAME`: これはイメージの発行者を説明し、通常連絡先の詳細をいくらか含めます。最大長は 128 文字です。
- 630 • `LB_ISO_VOLUME/--iso-volume NAME`: これはイメージのボリューム ID を指定します。Windows や Apple Mac OS 等一部のプラットフォームではユーザから見えるラベルとして利用されます。最大長は 32 文字です。

631 **Debian インストーラの独自化**632 **12. Debian インストーラの独自化**

633 Live システムのイメージは Debian インストーラと統合できます。
 インストールには収録内容やインストーラの動作方法によりいく
 つもの異なる種類があります。

634 この節で「Debian インストーラ」と大文字を使った表記で参照
 しているところに注意してください - この表記の場合には公式の
 Debian システム用インストーラを明示的に指していて、他の何
 かではありません。「d-i」と短縮することもよくあります。

635 **12.1 Debian インストーラの種類**

636 インストーラの主な 3 つの種類:

637 「通常の」**Debian** インストーラ：これは通常の live システム
 のイメージで、(適切なブートローダからそれを選択した場合に)
 Debian の CD イメージをダウンロードしてそれをブートしたの
 と同様に標準の Debian インストーラを起動するための別個の
 カーネルと `initrd` を収録しています。live システムとこういった
 別個の独立したインストーラを収録するイメージはよく「複合イ
 メージ」と呼ばれます。

638 こういったイメージでは、`debootstrap` を使ってローカルメディ
 アやネットワークから `.deb` パッケージを取得、インストールす
 ることで Debian がインストールされます。結果としてはデフォ
 ルトの Debian システムがハードディスクにインストールされま
 す。

639 このプロセス全体で、いくつかの方法で `preseed` を使って独自化
 できます。さらなる情報については Debian インストーラマニ
 ュアルの関連するページを見てください。機能する `preseed` ファイル
 が得られたら `live-build` が自動的にイメージに取り込んで使え
 るようになります。

640 「**Live**」**Debian** インストーラ：これは live システムイメージで、
 (適切なブートローダからそれを選択した場合に) Debian インス
 トーラを起動するための別個のカーネルと `initrd` を収録していま
 す。

641 インストールは上記で説明した「通常の」インストールと全く同
 じように進みますが、実際にパッケージをインストールする段
 階で、`debootstrap` を使ってパッケージを取得、インストールす
 る代わりに、live ファイルシステムのイメージを対象にコピーし
 ます。これは `live-installer` という特別な `udeb` により行っていま
 す。

642 この段階の後には、Debian インストーラはインストールや、ブート
 ローダやローカルユーザ等の設定を通常どおり続けます。

643 注意：一つの live メディアのブートローダの項目で通常のイン
 ストーラと live インストーラの両方に対応するには、`live-
 installer/enable=false` という `preseed` により `live-installer` を
 無効化する必要があります。

644 「デスクトップ」**Debian** インストーラ：収録する Debian インス
 トーラの種類を問わず、デスクトップからアイコンをクリックす
 ることで `d-i` を起動できます。状況によってはこちらの方がユー
 ザからわかりやすいこともあります。これを使えるようにするに
 は `debian-installer-launcher` パッケージを収録する必要があります。

645 `live-build` は Debian インストーラのイメージをデフォルトではイ
 メージに収録しないことに注意してください。`lb config` により
 具体的に有効化する必要があります。さらに、「デスクトップ」イン
 ストーラが機能するようにするには live システムのカーネルが
 指定されたアーキテクチャで `d-i` が利用するカーネルと一致する
 必要があることに注意してください。例えば:

```
646 $ lb config --architectures i386 --linux-flavours 486 \
```



```
--debian-installer live
$ echo debian-installer-launcher >> config/package-lists/my.list.chroot
```

647 12.2 preseed による Debian インストーラの独自化

648 <<http://www.debian.org/releases/stable/i386/apb.html>> にある Debian イン
ストーラマニュアルの付録 B で説明されていますが「preseed は、
インストールの実行中に手作業により回答を入力せずに、インス
トールプロセス中の質問の回答を設定する方法を提供します。こ
れにより、ほとんどの方法のインストールを完全に自動化し、さ
らに通常のインストールでは利用できない特徴もあります」。こ
の種の独自化は *live-build* を使って設定を preseed.cfg ファイル
に書き、config/includes.installer/ に置くことで最も完成させ
ることができます。例えばロケールを en_US に設定する preseed
は:

649

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

650 12.3 Debian インストーラの収録内容の独自化

651 実験やデバッグの目的で、ローカルでビルドした d-i の一部で
ある udeb パッケージを収録したいことがあるかもしれません。
config/packages.binary/ にそれを配置してイメージに収録し
ます。<Live/chroot ローカルインクルード> と同じ方法で内容を
config/includes.installer/ に置くことで、追加または置換す
るファイルやディレクトリを同様にインストーラの initrd に収録
することもできます。

652 プロジェクト

プロジェクトへの貢献

13. プロジェクトへの貢献

貢献物の提出にあたっては著作権者を明確に識別し、適用するライセンス文を収録してください。受け入れられるためには、その貢献物はその文書の他の部分と同一の、GPL バージョン 3 以降というライセンスを採用する必要があることに注意してください。

翻訳やパッチといったプロジェクトへの貢献は大いに歓迎します。誰もがリポジトリに直接コミットできますが、大きな変更についてはまずメーリングリストに送って議論するようお願いいたします。さらなる情報については [連絡先](#) 節を見てください。

Live システムプロジェクトでは Git をソースコード管理用のバージョン管理システムとして利用しています。[Git リポジトリ](#) で説明しているように、開発用ブランチは **debian** と **debian-next** の 2 つあります。debian-next ブランチの *live-boot*、*live-build*、*live-config*、*live-images*、*live-manual*、*live-tools* リポジトリには誰でもコミットできます。

ただし、特定の制限があります。サーバは

- fast-forward ではないプッシュ
- マージコミット
- タグやブランチの追加や削除

を拒否します。あらゆるコミットを訂正できるとはいえ、自分の常識に従って、良いコミットメッセージを使って良いコミットを行うようお願いいたします。

- 完結した、有意な文で構成されるコミットメッセージを英語で書き、大文字から始めて句点で終わるようにしてください。通常、「Fixing/Adding/Removing/Correcting/Translating/...」のようなものから開始します。

- 良いコミットメッセージを書いてください。先頭行はそのコミットの内容を正確にまとめるようにしてください。これは changelog に収録されることとなります。何か説明がさらに必要であれば、先頭行の後に 1 行空けてから書き、各段落の後は新たな空行を空けてください。段落の行の長さは 80 文字を超えないようにしてください。

- コミットは小分けにしてください。これは関係のないものをまとめてコミットしないようにということです。各変更ごとに別個にコミットするようにしてください。

13.1 変更を加える

リポジトリに送るには、以下の手順に従う必要があります。ここでは *live-manual* を例として使うのでそれは作業したいリポジトリに置き換えてください。*live-manual* を変更する方法に関する詳細な情報については [この文書への貢献](#) を見てください。

- 公開コミットキーを取得します:

```
$ 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*
```

- openssh-client の設定に以下を追記します:

```
$ cat >> ~/.ssh/config << EOF
Host live-systems.org
  Hostname live-systems.org
  User git
  IdentitiesOnly yes
```

```
IdentityFile ~/.ssh/keys/git@live-systems.org
EOF
```

- 672 • ssh 経由で *live-manual* の複製を取得します:

673

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

- 674 • Git で作者とメールをセットしたことを確認してください:

675

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

676 **重要:** 変更はどれも **debian-next** ブランチにコミットする必要
があるということを忘れないでください

- 677 • 変更を加えます。この例ではまずパッチの適用を扱う新しい節
を書き、ファイルの追加をコミットする下準備をしてコミット
メッセージを

678

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

- 679 • のように書いてサーバにコミットを送ります:

680

```
$ git push
```

681 バグの報告

682 14. バグの報告

683 live システムは完璧にはほど遠いですが、可能な限り完璧に近づ
 けたいと思っています - あなたの支援とともに。バグの報告を躊
 躇わないでください。バグがあるのに報告されないよりも二重に
 報告される方がいいからです。この章ではバグ報告を提出するに
 あたっての推奨事項について説明します。

684 せっかちな人向け:

- 685 • 常にまず <http://live-systems.org/> にある私達のホームページにある
 イメージの更新状況により既知の問題を確認してください。
- 686 • バグ報告を提出する前に使用している *live-build*、*live-boot*、*live-*
config、*live-tools* のブランチの *{最新版}* (*live-build* 4 を使っ
 ているなら最新のバージョン 4.x の *live-build*) でそのバグを再現
 できるか常に確認します。
- 687 • バグについて *{できるだけ具体的な情報}* を提示するようにし
 てください。これには (最低限) 利用した *live-build*、*live-boot*、
live-config、*live-tools* のバージョンや Live システムをどのディ
 ストリビューションでビルドしたのか、等があります。

688 14.1 既知の問題

689 Debian テスト版 (**testing**) と Debian 不安定版 (**unstable**) ディ
 ストリビューションは変化しているのでこのどちらかを対象シス
 テムディストリビューションに指定している場合、ビルドが常に
 成功するとは限りません。

690 そのためあまり困難になる場合はビルドに *{テスト版 (testing)}*
*** や *{不安定版 (unstable)}* をベースにしたシステムではなく
{安定版 (stable)} を使ってください。 *live-build* は常に *{安定版*
(stable)} リリースをデフォルトとしています。

691 現在わかっている問題は <http://live-systems.org/> にある私達のホーム
 ページの「status」に一覧があります。

692 開発用ディストリビューションのパッケージにある問題を正しく
 693 識別、修正するための訓練はこのマニュアルの目的ではありません
 694 ませんが、常に確認できることが 2 つあります: テスト版 (**testing**)
 を対象ディストリビューションとしてビルドに失敗した場合に
{不安定版 (unstable)} で試してみるということです。不安定版
 (**unstable**) でもダメな場合は *{テスト版 (testing)}* に差し戻し、
 失敗しているパッケージのもっと新しいバージョンを *{不安定*
 695 *版 (unstable)}* から利用するようにしてみます (詳細については
 <APT の PIN 設定> 参照)。

693 14.2 最初から再ビルド

694 きれいではない環境でシステムがビルドされたことにより特定の
 バグが発生しているのではないことを保証するため、live システ
 ム全体を最初から再ビルドして、そのバグが再現するか常に確認
 695 してください。

695 14.3 最新のパッケージを使う

696 問題を再現 (最終的には修正) しようとするときに古くなったパッ
 ケージを使用すると重大な問題を引き起こす可能性があります。
 ビルドシステムが最新であること、同様にそのイメージに収録さ
 れているパッケージがどれも最新であることを確認してくださ
 い。

697 14.4 情報収集

698 報告では十分な情報を提供してください。最低でもそのバグが発
 生した *live-build* の正確なバージョンとそれを再現する手順を含

めてください。常識的に考えて問題解決の支援になりそうだと思う関連情報が何か他にあればそれも提供してください。

699 バグ報告を最大限に活用するため、最低限次の情報が必要です:

- 700 • ホストシステムのアーキテクチャ
- 701 • ホストシステムのディストリビューション
- 702 • ホストシステムの *live-build* のバージョン
- 703 • ホストシステムの Python のバージョン
- 704 • ホストシステムの *debootstrap* や *cdebootstrap* のバージョン
- 705 • Live システムのアーキテクチャ
- 706 • Live システムのディストリビューション
- 707 • Live システムの *live-boot* のバージョン
- 708 • Live システムの *live-config* のバージョン
- 709 • Live システムの *live-tools* のバージョン

710 tee コマンドを使ってビルドプロセスのログを生成することができます。auto/build スクリプトによりこれを自動的に行うことを推奨します (詳細は [設定管理](#) 参照)。

711

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

712 ブート時に、*live-boot* と *live-config* はログファイルを /var/log/live/ に保存します。エラーメッセージはここを確認してください。

713 さらに、他のエラーを除外するため、config/ ディレクトリを tar でまとめてどこかにアップロードするのは常に良い方法です (メーリングリストに添付として送ら *{ないでください}**)。それ

により、そのエラーの再現を試みる事が可能になります。それが (例えばサイズの問題により) 困難な場合は `lb config --dump` の出力を使ってください。これは設定ツリーのまとめです (つまり config/ のサブディレクトリにあるファイル一覧を列挙しますがファイル自体は収録しません)。

ログは全て英語のロケール設定で生成されたものを提示することを忘れないでください。例えば先頭に `LC_ALL=C` や `LC_ALL=en_US` を付けて *live-build* コマンドを実行してください。

714

14.5 可能であれば失敗している状況を分離する

715

可能であれば失敗している状況を可能な限りうまくいかなくなる最小の変更に分離してください。これは常に簡単だとは限らないので、報告の際にできないようであれば気にする必要はありません。しかし、開発サイクルを向上させたい場合、繰り返しのたびに変更する量を十分に小さくすると、実際の設定により近く、より単純な「ベース」設定を構成することによりうまくいかなる追加の変更点だけに問題を分離することができるかもしれません。どの変更によりうまくいなくなっているのか区別するのに苦労している場合、それぞれの変更点が多すぎる事が考えられ、その場合開発の進行は緩くなるはずです。

716

14.6 正しいパッケージに対してバグを報告する

717

どの構成要素がそのバグの原因なのかわからない、あるいはそのバグが Live システム全般に関係するバグである場合は *debian-live* 疑似パッケージに対するバグとして報告してください。

718

とはいうものの、バグの現れ方を元にその範囲を限定してくれると助かります。

719

14.6.1 ビルド時のパッケージ収集中

721 *live-build* は最初に *debootstrap* または *cdebootstrap* で Debian システムの基本的なパッケージを収集します。利用したパッケージ収集ツールやパッケージを収集した Debian ディストリビューションによっては失敗するかもしれません。バグがここで起きていると思われる場合は、そのエラーが特定の Debian パッケージに (ほとんどの場合こちらです) 関連するのか、パッケージ収集ツール自体に関連するものなのか確認してください。

722 どちらの場合でも、これは Live システムではなく Debian 自体のバグで、恐らく私達が直接修正することはできません。こういったバグはパッケージ収集ツールまたは失敗しているパッケージに対して報告してください。

723 14.6.2 ビルド時のパッケージインストール中

724 *live-build* は追加のパッケージを Debian アーカイブからインストールしているため、利用する Debian ディストリビューションとその日のアーカイブの状態によっては失敗するかもしれません。バグがここで起きていると思われる場合は、そのエラーが通常のシステムで再現できるか確認してください。

725 通常のシステムで再現できる場合これは Live システムではなく Debian のバグです - 失敗しているパッケージに対して報告してください。Live システムのビルドとは別に *debootstrap* を実行、あるいは *lbbootstrap --debug* を実行するとさらなる情報を得られるでしょう。

726 また、ローカルミラーやプロキシの類を使っていて問題が起きている場合はまず、公式ミラーからパッケージを収集した場合に再現するか常に確認してください。

727 14.6.3 ブート時

イメージがブートしない場合は **情報収集** で指定している情報を添えてメーリングリストに報告してください。そのイメージが正確にどのように/どの段階で失敗しているのか、仮想化を使っているのか実際のハードウェアなのか、ということについて忘れずに言及してください。何らかの仮想化技術を使っている場合はバグを報告する前に常に実際のハードウェアで実行してください。失敗しているときのスクリーンショットを提供することも、とても参考になります。

14.6.4 実行時

パッケージのインストールには成功したけれども Live システムを実際に行っている間に何か失敗している場合、これは恐らく Live システムのバグです。その場合:

14.7 調査してください

バグを報告する前に、問題の症状やそのエラーメッセージについてウェブを検索してください。その問題に遭っているのがあなた一人だけという可能性は非常に低いからです。他のどこかで議題に上り、解決できそうな方法やパッチ、回避策が提案されている可能性は常にあります。

Live システムのメーリングリストや同様にホームページには、最新の情報がある可能性があるため、特に注意を払ってください。そういった情報が存在する場合は、バグ報告で常に参照するようにしてください。

さらに、似たことが既に報告されていないか *live-build*、*live-boot*、*live-config*、*live-tools* の現在のバグ一覧を確認してください。

14.8 バグの報告先

735

- 736 Live システムプロジェクトではバグ追跡システム (BTS) に報告されたバグを全て追跡しています。このシステムの使い方についての情報は <http://bugs.debian.org/> をご覧ください。reportbug パッケージの同名コマンドを使ってバグを報告することもできます。
- 737 一般的に、ビルド時のエラーは *live-build* に、ブート時のエラーは *live-boot* に、実行時のエラーは *live-config* パッケージに対して報告してください。どのパッケージが適切なのかわからない、あるいはバグの報告前にもっと支援が必要だという場合は *debian-live* 疑似パッケージに対して報告してください。その場合は私達が調べて適切なものに割り当てし直します。
- 738 (Ubuntu その他の) Debian 派生ディストリビューションで見つかったバグは、それが公式の Debian パッケージを使っている Debian システムでも再現するものでない限り、Debian BTS に報告すべきでは *{ない}* ことに注意してください。

739 コーディングスタイル

740 **15. コーディングスタイル**

741 この章では live システムで利用されているコーディングスタイル
742 について述べます。

742 **15.1 互換性**

- 743 • Bash シェル固有の書式や記号を使わないでください。例えば
744 配列構造の利用など
- 744 • POSIX のサブセットだけを使ってください - 例えば 'foo' より
745 も \$(foo) を使ってください。
- 745 • 'sh -n' と 'checkbashisms' によりスクリプトをチェックできま
746 す。
- 746 • シェルコードが全て確実に 'set -e' で動作するようにしてくだ
747 さい。

747 **15.2 インデント**

- 748 • 常にスペースよりもタブを使います。

749 **15.3 改行**

- 750 • 通常、行は最大で 80 文字までです。
- 751 • 「Linux 式」で改行します：

752 悪い例:

753

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

良い例:

754

755

```
if foo
then
    bar
fi
```

- 関数についても同様です:

756

悪い例:

757

758

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

良い例:

759

760

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

15.4 変数

761

- 変数は常に大文字です。 762
- *live-build* で利用する変数は先頭を常に LB_ で始めます。 763

- *live-build* 内部の一時変数は `<=underscore>LB_` で始めます。
- *live-build* のローカル変数は `<=underscore><=underscore>LB_` で始めます。
- *live-config* 中のブートパラメータにつながる変数は `LIVE_` で始めます。
- *live-config* 中の他の変数は全て `_` で始めます。
- 変数は大括弧「`{}`」で囲みます。例えば `$FOO` ではなく `${FOO}` とします。
- 空白文字の可能性を考慮し、常に引用符を使って変数を保護します: `${FOO}` ではなく `"${FOO}"` とします。
- 一貫性を保つため、変数に値を割り当てるときは常に引用符を使います:

悪い例:

```
F00=bar
```

良い例:

```
F00="bar"
```

- 複数の変数を使うときは表現全体を引用符で囲みます:

悪い例:

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

良い例:

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

15.5 その他

- `sed` を呼び出すときは区切り文字に「`|`」を使います。例えば `sed -e `s|'|``
- 比較やテストには `test` コマンドを使わず、「`[`」や「`]`」を使います。例えば `if [-x /bin/foo]; ...` を使い、「`if test -x /bin/foo; ...`」は使いません。
- `test` よりも `case` の方が読みやすく実行速度も早いため、可能な部分ではこちらを使います。
- ユーザの環境と混ざる可能性を限定するため、関数の名前には大文字を使います。

785 手順

786 **16. 手順**

787 この章では、Debian の他のチームと協調する必要がある、Live システムプロジェクトの様々な作業の手順について触れます。

788 **16.1 主要リリース**

789 Debian の安定版の新しい主要バージョンのリリースでは、その完成のために多くの異なるチームが協調して作業しています。どこかの時点で、Live チームが参加して live システムのイメージをビルドします。そのための要件は

- 790 • リリースしたバージョンに該当する debian 及び debian-security アーカイブを収録している、debian-live の buildd からアクセスできるミラー
- 791 • イメージの名前は既知の形式である必要があります (例えば debian-live-バージョン-アーキテクチャ-収録デスクトップ環境等.iso)。
- 792 • debian-cd から来るデータを同期する必要があります (udeb 除外一覧)。
- 793 • ビルドされたイメージのミラーが cdimage.debian.org に置かれます。

794 **16.2 ポイントリリース**

- 795 • ここでも、debian と debian-security の更新されたミラーが必要です。
- 796 • ビルドされたイメージのミラーが cdimage.debian.org に置かれます。
- 797 • 告知メールを送ります

16.2.1 ある Debian リリースの最後のポイントリリース 798

799 ある Debian リリース向けの最後のイメージを ftp.debian.org から archive.debian.org に移動した後にビルドするときには、chroot とバイナリミラーの両方を調整することを忘れないでください。そうすることで、古いビルド済み live イメージをユーザが変更しなくてもそのまま続けて使えるようになります。

16.2.2 ポイントリリース告知用テンプレート 800

801 ポイントリリース用の告知メールはテンプレートと以下のコマンドを使って生成できます。

```
802 $ 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'
```

803 メールを送る前に注意深く確認し、他の人による校正を受けてください。

804

```
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:

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

Additionally it includes the following Live-specific changes:

- * [LIVE 固有の変更をここに]
- * [LIVE 固有の変更をここに]
- * [大きな問題については専用の節を作ることもあります]

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>.

805 **Git** リポジトリ806 **17. Git** リポジトリ

807 Live システムプロジェクトの利用可能な全リポジトリー覧は
 <<http://live-systems.org/gitweb/>> にあります。プロジェクトの git URL は:
 プロトコル://live-systems.org/git/リポジトリ という形式に
 なっています。したがって、*live-manual* を読み込み専用で複製す
 るには

```
808 $ git clone git://live-systems.org/git/live-manual.git
```

```
809 $ git clone https://live-systems.org/git/live-manual.git
```

```
810 $ git clone http://live-systems.org/git/live-manual.git
```

811 のどれかを実行します。書き込み権限のある複製には `git@live-`
`systems.org:/`リポジトリという形式のアドレスを使いま
 す。

812 なので、繰り返しますが *live-manual* を ssh 経由で複製するに
 は

```
813 $ git clone git@live-systems.org:live-manual.git
```

814 と入力する必要があります。git ツリーは複数の異なるブランチで
 できています。**debian** 及び **debian-next** ブランチは最終的には

新しいリリースそれぞれに収録される実際の作業を収録している
 ため特に注目すべきです。

815 既存のリポジトリのどれかを複製した後は **debian** ブランチに
 いるでしょう。これはプロジェクトの最新リリースの状態を確認す
 るには適切ですが、作業開始前に必ず **debian-next** ブランチに
 切り替える必要があります。切り替えには

```
816 $ git checkout debian-next
```

817 を実行します。**debian-next** ブランチは常に fast-forward とは限
 らず、あらゆる変更が **debian** ブランチにマージされる前にまず
 はここにコミットされます。例えて言えばテストの場のようなもの
 です。このブランチで作業していてサーバにある変更を取得する
 必要がある場合は `git pull --rebase` を実行する必要があります
 ます。それにより、サーバから取得するときにローカルでの変更が
 反映され、その変更が最上位に配置されます。

818 **17.1** リポジトリを複数処理

819 live システムのリポジトリを複数複製してすぐに、最新コードの
 確認、パッチ作成、あるいは翻訳での貢献等のために **debian-next**
 ブランチに切り替えたい場合、複数のリポジトリを扱いやすくす
 るための `mrconfig` ファイルを git サーバで提供していることを
 紫知っておくべきでしょう。これを使うには `mr` パッケージをイン
 ストールする必要があります。その後、

```
820 $ mr bootstrap http://live-systems.org/other/mr/mrconfig
```

821 を実行します。このコマンドは自動的に複製し、プロジェクト
 により作成される Debian パッケージの開発用リポジトリであ

る **debian-next** ブランチを取得します。これには、中でも *live-images* リポジトリがあり、プロジェクトが一般用途向けに公開しているビルド済みイメージで利用される設定を収録しています。このリポジトリの使い方に関するさらなる情報については、[〈Git 経由で公開されている設定の複製〉](#) をご覧ください。

822 例

823 例

824 **18. 例**

825 この章では特定の live システム活用事例向けの見本ビルドについて触れます。自分用の live システムイメージのビルドが初めてであれば、まず3つのチュートリアルを順に調べてみることを勧めます。それぞれで他の例の利用、理解を支援する新しい技術を学ぶようになっているためです。

826 **18.1 例の使用**

827 提示している例を利用するためには、ビルドするために〈要件〉に記載されている要件一覧に合致するシステムと、〈*live-build* のインストール〉で説明しているように *live-build* がインストールされていることが必要となります。

828 簡潔にするため、ここに挙げる例ではビルドで利用するローカルミラーを指定していないことに注意してください。ローカルミラーを利用するとダウンロード速度をかなり高速化できます。ビルド時に利用するディストリビューションのミラーで説明しているように、`lb config` を使った場合はオプションを指定することができます。ビルドシステムのデフォルト値を `/etc/live/build.conf` でセットするともっと便利になります。このファイルを単純に作成し、対応する `LB_MIRROR_*` 変数に望ましいミラーをセットしてください。ビルドで利用する他のミラーは全て、これにより設定した値をデフォルト値として使います。例えば:

829

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

18.2 チュートリアル 1: デフォルトイメージ 830

831 事例: 簡単な最初のイメージを作成して *live-build* の基礎を学びます。

832 このチュートリアルでは、*live-build* を利用した最初の演習として `base` パッケージ (Xorg は含まない) と *live* システムを支援するパッケージだけを収録する、デフォルトの ISO hybrid 形式の *live* システムイメージをビルドします。

833 これ以上簡単にすることはなかなかできないでしょう:

```
834 $ mkdir tutorial1 ; cd tutorial1 ; lb config
```

835 何か望むことがあれば `config/` ディレクトリの内容を調べてください。ここには概略の設定があり、すぐ独自化もできますが、ここではそのままデフォルトのイメージをビルドします。

836 スーパーユーザでイメージをビルドし、そのログを `tee` により保存します。

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

838 すべてがうまくいくとして、しばらくすると現在のディレクトリに `live-image-i386.hybrid.iso` が出来上がります。この ISO hybrid イメージは〈*Qemu* での ISO イメージのテスト〉や〈*VirtualBox* での ISO イメージのテスト〉で説明しているように仮想マシンで直接、あるいは〈物理メディアへの ISO イメージ書き込み〉や〈USB メモリへの ISO hybrid イメージのコピー〉で説明しているように光学メディアや USB フラッシュ機器に書き込んだイメージ、それぞれからブートできます。

18.3 チュートリアル 2: ウェブブラウザユーティリティ

8439

840 事例: ウェブブラウザユーティリティイメージを作成し、独自化
の適用方法を学びます。

841 このチュートリアルでは live システムイメージを独自化する方法
の紹介として、ウェブブラウザユーティリティとしての利用に適
するイメージを作成します。

842

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

843 この例で LXDE を選択しているのは最小限のデスクトップ環境
を提供するという私達の目的を反映しています。念頭に置いて
いるこのイメージの目的はただ一つ、ウェブブラウザだけだから
です。もっと細かく、config/includes.chroot/etc/iceweasel/-
profile/ でのウェブブラウザ向けデフォルト設定やウェブ上の
様々な種類の内容を表示するための追加のサポートパッケージを
提供することはできますが、それは読み手の演習として残してお
きます。

844 <チュートリアル 1> と同様、ここでもスーパーユーザでイメージ
をビルドし、ログを残します:

845

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

846 ここでも <チュートリアル 1> と同様、イメージがうまくできてい
るか検証し、テストします。

18.4 チュートリアル 3: 私的イメージ

848 事例: プロジェクトを作成して個人用イメージをビルドします。
USB メモリを使って好みのソフトウェアを自由に収録し、要求や
設定を変更しながらこのイメージを続けて改訂します。

849 この個人用イメージを何度も改訂し、変更を追跡しておいて実験
的に試してみてもうまくいかなかったときには差し戻せるようにし
たいため、人気のある git バージョン管理システムに設定を残し
ます。<設定管理> で説明している auto スクリプトによる自動設
定を経由した最善の実践も利用します。

18.4.1 最初の改訂

850

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

852 auto/config を以下のように変更します:

853

```
#!/bin/sh

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

854 Perform lb config to generate the config tree, using the auto/
config script you just created:

855

```
$ lb config
```

856 ここでローカルパッケージ一覧を設定します:

857

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

858 まず、`--architectures i386` により必ず `amd64` ビルドシステムでほとんどのマシンでの利用に適応する 32 ビット版をビルドするようにします。次に、相当に古いシステムでのこのイメージの利用を想定しないため `--linux-flavours 686-pae` を使います。`lxde task` メタパッケージを選択して最小限のデスクトップを揃えます。最後に、好みのパッケージの初期値として `iceweasel` と `xchat` を追加しています。

859 そして、イメージをビルドします:

860

```
# lb build
```

861 最初の 2 つのチュートリアルとは異なり、`2>&1 |tee build.log` は `auto/build` に書かれているため打ち込む必要がなくなっていることに注意してください。

862 (<チュートリアル 1>にあるように) イメージをテストしてうまく機能する確信を得たら `git` リポジトリを初期化し、作成したばかりの `auto` スクリプトだけを追加し、最初のコミットを行います:

863

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

18.4.2 2 回目の改訂

864

865 この改訂では、最初のビルドをきれいにし、`vlc` パッケージを設定に追加して再ビルド、テストコミットを行います。

866 `lb clean` コマンドは前のビルドで生成したファイルを、パッケージを再びダウンロードせずに済むようにキャッシュを除いて全てきれいにします。これにより以降の `lb build` が全段階で再び実行され、必ず新しい設定でファイルを再生成するようになります。

867

```
# lb clean
```

868 `vlc` パッケージを `config/package-lists/my.list.chroot` のローカルパッケージ一覧に追記します:

869

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

870 再びビルドします:

871

```
# lb build
```

872 テストして満足したら次の改訂としてコミットします:

873

```
$ git commit -a -m "Adding vlc media player."
```

874 もちろん、`config/` 以下のサブディレクトリにファイルを追加する等により設定をもっと複雑に変更することも可能です。新しい

改訂版をコミットする際、`config` の最上位にある、`LB_*` 変数を設定しているファイルもビルドされてきたもので、`lb clean` と、対応する `auto` スクリプトを経由して再作成した `lb config` により常に整理されるものなので、手で編集したりコミットすることのないように注意してください。

875 一連のチュートリアルもこれで終わりです。もっと多様な独自化はできますが、ここまでの簡単な例で見えてきた少しの機能を使うだけでも、イメージはほぼ無限の異なる組み合わせを作成することができます。この節の残りの例では、収集してきた `live` システムのユーザの経験を元にした他の事例についていくつか触れません。

876 18.5 VNC 公衆クライアント

877 事例: `live-build` を使って、ブートすると直接 VNC サーバに接続するイメージを作成します。

878 ビルド用ディレクトリを作ってそこに概略設定を作成し、推奨パッケージを無効にして最小限のシステムを作成します。それから初期パッケージ一覧を 2 つ作成します: 1 つ目は `live-build` により提供される `Packages` というスクリプト (生成されるパッケージ一覧参照) により生成し、2 つ目では `xorg`、`gdm3`、`metacity`、`xvnc4viewer` を収録します。

879

```
$ 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
```

880 <APT の調整による容量の節約> で説明しているように、イメージ

が適切に機能するためには推奨パッケージを再びいくらか追加する必要があるかもしれません。

推奨パッケージ一覧を調べるための簡単な方法として `apt-cache` の利用があります。例えば:

881

882

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

883 この例では `live-config` 及び `live-boot` により推奨されるパッケージを複数、再び収録する必要があることがわかっています: 自動ログインが機能するためには `user-setup`、システムをシャットダウンするための不可欠なプログラムとして `sudo`。他に、イメージを RAM にコピーできるようになる `live-tools` や `live` メディアを最終的に取り出す `eject` を追加しておくとう便利でしょう。それを反映すると:

884

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

885 その後ディレクトリ `/etc/skel` を `config/includes.chroot` に作成し、その中にデフォルトユーザ向けの独自の `.xsession` を置きます。このファイルは `metacity` を立ち上げて `xvncviewer` を起動し、`192.168.1.2` にあるサーバのポート `5901` に接続します:

886

```
$ 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

895

887 イメージをビルドします:

888

```
# lb build
```

889 楽しみましょう。

890 18.6 128MB USB メモリ向けの基本イメージ

891 事例: 128MB USB メモリに収まるように構成要素をいくらか削除して、収まることがわかるように容量を少し空けたデフォルトのイメージの作成。

892 特定のメディア容量に収まるようにイメージを最適化する場合、イメージのサイズと機能はトレードオフになることを理解する必要があります。この例では削るだけにしているので 128MB のメディアサイズ内に何か追加する余地をできるだけ残していますが、*localepurge* パッケージによるロケールの完全削除や収録しているパッケージ内の一貫性は何も壊していません。また、その他の「押し付ける」ような最適化もしていません。特に注目すべきなのは、最小限のシステムを最初から作成するために `--debootstrap-options` を利用している点です。

893

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

894 イメージを適切に機能させるためには、最低でも `--apt-recommends false` オプションにより外されていた推奨パッケージを 2 つ追加しなおす必要があります。<APT の調整による容量の節約> を見てください。

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

ここで、普通の方法でイメージをビルドしてみます:

896

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

897

898 これを書いている時点の著者のシステムでは、上記の設定により 77MB のイメージができました。これを <チュートリアル 1> のデフォルト設定で作成された 177MB のイメージと都合良く比較してみましょう。

899 i386 アーキテクチャシステム上でデフォルトのイメージをビルドするのと比較して、ここで最もスペースの節約になったのはカーネルのアーキテクチャの種類をデフォルトの `-k "486 686-pae"` に代えて 486 だけを選択することでした。 `--apt-indices false` により APT の索引を省くことでもかなりの容量を節約していますが、その代わりに live システムで *apt* を使う前に `apt-get update` を実行する必要があります。 `--apt-recommends false` により推奨パッケージを除外することで、本来あるはずのパッケージをいくらか除外する代わりにいくらか追加で容量を節約します。 `--debootstrap-options "--variant=minbase"` で最初から最小限のシステムを構成します。 `--firmware-chroot false` でファームウェアパッケージを自動的に収録しないようにすることもさらに容量をいくらか節約します。そして最後に、 `--memtest none` によりメモリテスターのインストールを抑制します。

900 注意: 最小限のシステムの構成はフックを使って、例えば `/usr/share/doc/live-build/examples/hooks` にある `stripped.hook.chroot` でも実現できます。これは容量をさらに少し減らし、62MB のイメージを生成します。しかしこれはその実現のために、システムにインストールしたパッケージから文書

その他のファイルを削除しています。これはそうしたパッケージの完全性を破壊し、ヘッダで警告しているように思わぬ結果をもたらすかもしれません。それが、この目標のために推奨するのが最小限の *debootstrap* を利用する方法になっている理由です。

901 18.7 地域化した GNOME デスクトップとインストーラ

902 事例: GNOME デスクトップのイメージを作成し、スイス用の地域化とインストーラを収録する

903 好みのデスクトップを使った i386 アーキテクチャ向けの iso-hybrid イメージを作りたい。ここでは GNOME を使用して、GNOME 用の標準の Debian インストーラによりインストールされるのと同じパッケージを全て収録します。

904 最初の問題は適切な言語用タスクの名前を判断する方法です。現在 *live-build* はこれを支援できません。運良くこれを試行錯誤で見つけられるかもしれませんが、そのためのツールがあります。grep-dctrl を利用して tasksel-data にあるタスクの説明を見つけることができます。そのため、準備としてこの両方が揃っていることを確認してください。

905

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

906 これで適切なタスクを検索できるようになりました。まず、

907

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

908 というコマンドにより、呼ばれたタスクが、簡単に言うところでは

ドイツだということがわかります。次は関連タスクを見つけます:

909

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

ブート時に **de_CH.UTF-8** ロケールを生成して **ch** のキーボードレイアウトを選択します。一緒に見ていきましょう。<メタパッケージの利用> から、タスクのメタパッケージには先頭に **task-**が付くことを思いだしてください。こういった言語のブートパラメータを指定し、それから優先度が標準のパッケージと発見したタスクの全メタパッケージをパッケージ一覧に追加するだけです:

910

911

```
$ 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
```

912 のようになります。インストーラを live デスクトップから立ち上げるために *debian-installer-launcher* パッケージを収録し、さらに 486 用のカーネルを指定していることに注意してください。これは現在、インストーラを立ち上げる機能が適切に動作するため

にはインストーラと live システムのカーネルを一致させる必要があるためです。

914	スタイルガイド		
915	19. スタイルガイド		
916	19.1 著者向けガイドライン		
917	この節では live マニュアル向けの技術的文書を記述する際に一般的に考慮すべき事項を扱います。言語特性と推奨手順に分かれています。		
918	注意: 著者はまず <この文書への貢献> を読んでください		
919	19.1.1 言語特性		
920	• 平易な英語を使う		
921	読み手は英語が母国語ではない人の割合が高いことに留意してください。そのため、一般的規則として短く有意な文章を使い、引き続き終止符を打ってください。		
922	これは単純で幼稚な書き方をするように言っているわけではありません。英語が母国語ではない人にとって理解しにくい複雑な従属文にすることを可能な限り避けましょうという提案です。		
923	• 英語の方言		
924	最も広く使われている英語の方言はイギリス英語とアメリカ英語なので、ほとんどの著者が非常に高い率でこのどちらかを使っています。共同作業環境下で理想的なのは「国際英語」ですが、既存の全ての方言からどれを使うのが最善なのか決定するのは不可能とは言いませんが非常に困難です。		
925	誤解を生まずに複数の方言を混在させることもできるとは思いますが、一般論として一貫性を持たせるようにすべきで、また、イギリス英語やアメリカ英語、その他の英語の方言からどれを使うか自分の裁量で決める前に、他の人がどのように書いているのかを調べてそれを真似るようにしてください。		
	• バランス良く	926	
	偏見を持たないようにしてください。live マニュアルに全く関係のない思想への言及を引用することは避けてください。技術的文献は可能な限り中立であるべきです。科学的文献では中立こそが自然です。	927	
	• 政治的に正しく	928	
	性差を表す言葉を可能な限り避けるようにしてください。個人の第三者を持ち出す必要がある場合は「he (彼)」や「she (彼女)」、あるいは「s/he や s(he) 彼 (女)」などと複雑にするよりも「they (彼ら)」を使うのが好ましいです。	929	
	• 簡潔に	930	
	要点を直接述べ、回りくどい表現を使わないようにしてください。必要な情報は十分に提示ながらも、必要以上の余計な情報を提示するのはやめてください。これは不要な詳細を説明しないようにということです。読み手には理解力があります。読み手の側にくらか前提知識があることを仮定してください。	931	
	• 翻訳作業を最小限に	932	
	書かれたものは他の複数の言語に翻訳されることになるということに留意してください。これは無意味あるいは冗長な情報を追加するとその分余計な作業をする人が出てくるということを意味します。	933	
	• 一貫性を	934	
	前にも提案しましたが、共同作業の文書を標準化して全体を完全に統一することはほぼ不可能です。しかし、文書を書く際に全体を通して他の著者と一貫した書き方をすることを歓迎します。	935	
	• 結束性を	936	
	必要なだけ文脈形成句を使い、文章に結束性を持たせて明確にしてください (文脈形成句は接続語句等の言語標識です)。	937	

- 938 • 記述的に
- 939 標準的な「changelog」形式で文を単に羅列するよりも段落を使って要点を説明する方が好ましいです。描写してください! 読み手はそれを歓迎するでしょう。
- 940 • 辞書
- 941 英語で特定の概念を表現する方法がわからないときは辞書や百科事典でその語の意味を調べてください。ただし、辞書は最高の友ですが正しい使い方を知らなければ最悪の敵にもなることに留意してください。
- 942 英語には最大の語彙が存在する言語の一つです (100 万語以上)。この語の多くは他の言語から取り入れられたものです。単語の意味を二カ国語の辞書で調べる際、英語が母国語ではない人は母国語の言葉により似ているものを選択する傾向があります。このことにより、英語ではあまり自然に聞こえない、過度に形式ばった文体になりがちです。
- 943 原則として、ある概念が複数の異なる同義語により表現できるとき、辞書で最初に提示された語を選択するのが良い判断となるでしょう。疑問がある場合はゲルマン起源の語 (通常単音節の語) を選択すると多くの場合正しいとなります。この 2 つの技ではどちらかというところだけ表現になるかもしれないという点には注意が必要ですが、少なくとも広く使われていて通常受け入れられる語を選択することになります。
- 944 共起辞書の利用を勧めます。通常合わせて利用する語がわかるようになると極めて役に立ちます。
- 945 繰り返しますが、他の人の作業から学ぶことが最良の実践です。検索エンジンを使って他の著者が特定の表現をどのように使っているか確認することは大きな手助けとなるでしょう。
- 946 • 空似言葉や熟語その他の慣習的な表現
- 947 空似言葉に気をつけてください。外国語の熟練度を問わず、2 つの言語で同じように見える語けれどもその意味や使い方が全く異なる「空似言葉」という罠にはまることは避けられません。
- 948 熟語は可能な限り避けてください。「熟語」は個々の語が持っていた意味とは完全に異なる意味を表すことがあります。熟語は英語が母国語の人でさえ理解しにくいこともあります!
- 949 • 俗語や省略、短縮表現等は避けましょう
- 950 平易な、日常的な英語の使用を勧めるとはいつても、技術的文献は言語を正式に記録する類のものです。
- 951 俗語や通常使わない解釈困難な省略表現、特に母国語での表現を模倣するような短縮表現は避けてください。IRC や、家族や仲間内で使うような特有の表現での記述はしないでください。
- 952 **19.1.2 手順**
- 953 • 書く前にテストを
- 954 著者が live マニュアルに追加する前に例をテストして、全て確実に説明通りに動作するようにすることは重要です。きれいな chroot や VM 環境でのテストが良い起点となるでしょう。他に、それから異なるハードウェアを使っている異なるマシンでテストを実施し、起きるであろう問題を発見することができれば理想でしょう。
- 955 • 例
- 956 例示するときにはできるだけ具体的にするようにしてください。例は結局例でしかありませんから。
- 957 抽象的な表現で読み手を混乱させるよりも、特定の状況でのみ適用できるような書き方をする方がより良いことはよくあります。この場合は提示した例の効果簡単に説明することもできます。
- 958 使い方を誤ればデータ消失や類似の望ましくない影響を及ぼす可

能性のある、潜在的に危険なコマンド類の使用を例示する場合等、例外がいくらかあります。この場合は起こりうる副作用について十分な説明を提供すべきです。

959 • 外部リンク

960 外部サイトへのリンクは、そのサイトにある情報が特別な点を理解するために決定的な効果が期待できる場合にのみ利用すべきです。その場合でも、外部サイトへのリンクは可能な限り少なくしてください。インターネット上のリンクはその内容がほとんどが変更される可能性があるもので、その結果機能しないリンクができたり、論拠を不完全な状態にしてしまうことになります。

961 他に、インターネットに接続せずにそのマニュアルを読んでいる人にはそのリンクを追う機会がありません。

962 • 商標の主張やマニュアルの公開にあたって採用したライセンスに違反するものは避ける

963 商標の主張は可能な限り避けてください。記述した文書は他の下流のプロジェクトで使うことになるかもしれないことに留意してください。つまり、ある種の特定の内容を追加することは事態を複雑にすることになります。

964 *live-manual* は GNU GPL の条件下で使用を許可しています。これには、合わせて公開する (著作権のある画像やロゴを含むあらゆる種類の) 内容の配布物に適用する意味合いがいくつかあります。

965 • まず草稿を書き、改訂、変更して改善し、必要なら作り直す

966 - 案を引き出しましょう! まず論理的に順を追って考えを整理する必要があります。

967 - 頭の中で何とか形ができれば最初の草稿を書きます。

968 - 文法や書式、つづりを直します。リリースの正しい名前は **jessie** や **sid** で、これをコード名として参照するときは大文字にすべきではないことに留意してください。「spell」ターゲットを使っ

て、つまり `#{make spell}#` でつづりの誤りがないか確認できます。

- 記述を改善し、必要な部分があれば書き直します。

970 • 章

971 章や副題には慣習的な番号の付け方をしてください。例えば 1、1.1、1.1.1、1.1.2 ... 1.2、1.2.1、1.2.2 ... 2、2.1 ... などというようにです。以下のマークアップを見てください。

972 説明するのに一連の手順や段階を列挙する必要がある場合は、First (最初に)、second (2 つ目に)、third (3 つ目に) ... というように序数を使ったり、First (最初に)、Then (それから)、After that (その後)、Finally (最後に)、... あるいは箇条書きすることもできます。

973 • マークアップ

974 大事なことを言い忘れましたが、*live-manual* では `<SiSU>` を使ってテキストファイルを処理し、複数の形式の出力を生成しています。`<SiSU マニュアル>` を眺めてそのマークアップ方法をよく理解することを勧めます。代わりに

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

976 と入力する方法もあります。マークアップをいくらか例示してみます。有用だということはわかるかもしれません。

977 - 文字列の強調/太字:

```
*{foo}* または !{foo}!
```

979 は「**foo** または **foo**」となります。これは特定のキーワードを強

調するのに使います。

- 斜体:

```
/{foo}/
```

は *foo* となります。これは例えば Debian パッケージの名前に使います。

- 等幅:

```
#{foo}#
```

は `foo` となります。これは例えばコマンドの名前に使います。また、キーワードやパスのようなものの一部を強調するのにも使います。

- コードブロック:

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

は

```
$ foo
# bar
```

990 となります。タグの開始には `code{` を、終了には `}code` を使います。コードの各行には先頭に空白が必要だということを必ず覚えておいてください。

19.2 翻訳者向けガイドライン 991

992 この節では live マニュアルの内容を翻訳する際に一般的に考慮すべき事項を扱います。

993 一般的な推奨事項として、翻訳者は自分の言語に適用される翻訳規則を既に読んで理解しておくべきです。通常、翻訳用のグループやメーリングリストが Debian の品質標準に合致する翻訳物を作成する方法についての情報を提供しています。

994 注意: 翻訳者は [この文書への貢献](#) も読むべきです。特に [翻訳](#) 節を

19.2.1 翻訳の手がかり 995

• コメント 996

997 翻訳者の役割は元の著者により書かれた語や文、段落、そして文章の意味を可能な限り忠実に目標の言語で伝えることです。

998 そのため、個人的なコメントや自分の余計な情報の追加は控えるべきです。同一の文書について作業している他の翻訳者に向けてコメントを追加したい場合はそのために用意されている場合があります。これは **po** ファイルの番号記号 `#` に続く文字列のヘッダです。ほとんどの視覚的な翻訳用プログラムで自動的にこれをコメントの種類に属するものとして処理します。

• *TN, Translator's Note* (翻訳者によるメモ) 999

1000 完全に受け入れられるとはいえ、翻訳済みテキストの括弧「()」内に語や表現を含めることは、ややこしい語や表現の意味を読み手にとってより明確にする場合にのみ行ってください。翻訳者は括

	弧内に「(訳注)」等と記載して、その追記が翻訳者によるものであることを明確にすべきです。		
1001	• 非人称の文を		
1002	英語で書かれた文書は「you」を非人称として幅広く使います。他の言語にはこの特徴を共有しないものもあります。このことで、元の文が読み手に対して直接呼びかけているかのような誤った印象を実際にはそうではないのに与えてしまうかもしれません。翻訳者はこの点に注意して、可能な限り正確に自分の言語に反映する必要があります。		
1003	• 空似言葉		
1004	前に説明した「空似言葉」の罣は特に翻訳者に当てはまります。疑いがあれば、その疑わしい空似言葉の意味を再点検してください。		
1005	• マークアップ		
1006	最初は <code>*{pot}*</code> ファイル、後には <code>*{po}*</code> ファイルについて作業する翻訳者は多数のマークアップ機能を文字列に確認できるでしょう。文は翻訳できるものである限り翻訳できますが、それが元の英語版と全く同一のマークアップを採用しているということは極めて重要です。		
1007	• コードブロック		
1008	コードブロックは通常翻訳できませんが、翻訳にそれを含めることが、翻訳率 100% を達成する唯一の方法です。コードが変更されると翻訳者による介入が必要となるため最初は余計な作業になりますが、長期的に見るとこれが .po ファイルの整合性を確認したときに何が既に翻訳済みで何が未翻訳なのか識別する最善の方法です。		
1009	• 改行		
1010	翻訳文には元の文と全く同じだけの改行が必要です。元のファイルに改行があるときは注意して「Enter」キーを押すか <code>*{}*</code> を打ち		
		込んでください。改行は例えばコードブロック中でよく使われます。	
		勘違いしないでください。これは翻訳文を英語版と同一の長さにする必要がある、ということではありません。それはほぼ不可能です。	1011
		• 翻訳できない、してはいけない文字列	1012
		翻訳者が決して翻訳すべきでないもの:	1013
		- リリースのコード名 (小文字で書くべき)	1014
		- プログラムの名前	1015
		- 例示するコマンド	1016
		- メタ情報 (前後にコロンが置かれることが多い: メタ情報:)	1017
		- リンク	1018
		- パス	1019

SiSU Metadata, document information

Document Manifest @:

<http://complete.sisudoc.org/manual/manifest/live-manual.ja.html>

Title: Live システムマニュアル

Creator: Live システムプロジェクト <debian-live@lists.debian.org>

Rights: Copyright: Copyright (C) 2006-2014 Live Systems Project

License: This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

The complete text of the GNU General Public License can be found in /usr/share/common-licenses/GPL-3 file.

Publisher: Live システムプロジェクト <debian-live@lists.debian.org>

Date: 2014-08-27

Version Information

Sourcefile: live-manual.ssm.sst

Filetype: UTF-8 assumed, file encoding check program unavailable

Source Digest: SHA256(live-manual.ssm.sst)=c72478eecf9a8fdd9b26d2b5-de5b6cd2c350229babb9c4866964d4a5df6cd997

Generated

Document (ao) last generated: 2017-06-19 16:12:32 +0000

Generated by: SiSU 7.1.9 of 2016w30/7 (2016-07-31)

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