

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

About	2
このマニュアルについて	3
1. このマニュアルについて	3
1.1 せっかちな人向け	3
1.2 用語	4
1.3 著者	5
1.4 この文書への貢献	5
1.4.1 変更の適用	6
1.4.2 翻訳	7
Live システムプロジェクトへの貢献	9
2. Live システムプロジェクトについて	9
2.1 動機	9
2.1.1 現在の Live システムの問題点	9
2.1.2 自身の Live システムを作成する理由	9
2.2 哲学	10
2.2.1 Debian 「main」 の変更しないパッケージしか使いません	10
2.2.2 Live システム固有のパッケージ設定はありません	10
2.3 連絡先	10
ユーザ	11
インストール	12
3. インストール	12
3.1 要件	12
3.2 live-build のインストール	12
3.2.1 Debian リポジトリから	12
3.2.2 ソースから	12
3.2.3 「スナップショット」 から	13
3.3 live-boot と live-config のインストール	13
3.3.1 Debian リポジトリから	14
3.3.2 ソースから	14
3.3.3 「スナップショット」 から	15

基本	16
4. 基本	16
4.1 Live システムとは何？	16
4.2 ビルド済みイメージのダウンロード	17
4.3 ウェブ Live イメージビルダーの利用	17
4.3.1 ウェブビルダーの使い方と注意	17
4.4 最初の段階: ISO hybrid イメージのビルド	18
4.5 ISO hybrid Live イメージの利用	19
4.5.1 ISO イメージの実際のメディアへの書き込み	19
4.5.2 ISO hybrid イメージの USB メモリへのコピー	19
4.5.3 USB メモリの空きスペースの利用	20
4.5.4 Live メディアのブート	20
4.6 仮想マシンを利用したテスト	21
4.6.1 QEMU での ISO イメージのテスト	21
4.6.2 VirtualBox での ISO イメージのテスト	22
4.7 HDD イメージのビルド及び利用	22
4.8 netboot イメージのビルド	23
4.8.1 DHCP サーバ	24
4.8.2 TFTP サーバ	25
4.8.3 NFS サーバ	25
4.8.4 ネットワーク経由のブートをテストする方法	26
4.8.5 Qemu	26
4.9 ウェブブート	26
4.9.1 ウェブブートファイルの取得	27
4.9.2 ウェブブートイメージの起動	27
ツールの概要	29
5. ツールの概要	29
5.1 live-build パッケージ	29
5.1.1 lb config コマンド	29
5.1.2 lb build コマンド	30
5.1.3 lb clean コマンド	30
5.2 live-boot パッケージ	31
5.3 live-config パッケージ	31
設定の管理	32
6. 設定の管理	32
6.1 設定変更への対応	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？	32
6.1.2 自動化スクリプトの使用例	32

6.2 Git 経由で公開されている設定の複製	33
収録内容の独自化	35
7. 独自化の概要	35
7.1 ビルド時とブート時の設定	35
7.2 ビルド段階	35
7.3 ファイルによる lb config の補完	36
7.4 独自化タスク	36
インストールするパッケージの独自化	37
8. インストールするパッケージの独自化	37
8.1 パッケージソース	37
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	37
8.1.2 ディストリビューションミラー	38
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	38
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	38
8.1.5 追加リポジトリ	39
8.2 インストールするパッケージの選択	39
8.2.1 パッケージ一覧	40
8.2.2 メタパッケージの利用	40
8.2.3 ローカルパッケージ一覧	41
8.2.4 ローカルバイナリパッケージ一覧	41
8.2.5 生成されたパッケージ一覧	41
8.2.6 条件付き内部パッケージ一覧の利用	42
8.2.7 インストール時のパッケージの削除	42
8.2.8 デスクトップ及び言語タスク	43
8.2.9 カーネルのフレーバー(種類) とバージョン	43
8.2.10 独自のカーネル	44
8.3 変更したあるいはサードパーティ製パッケージのインストール	44
8.3.1 packages.chroot を利用した独自のパッケージのインストール	45
8.3.2 APT リポジトリを利用した独自パッケージのインストール	45
8.3.3 独自パッケージと APT	45
8.4 ビルド時の APT 設定	46
8.4.1 apt と aptitude の選択	46
8.4.2 APT でのプロキシの利用	46
8.4.3 APT の調整による容量節約	46
8.4.4 apt や aptitude へのオプションの受け渡し	47
8.4.5 APT のピン止め	48

収録内容の独自化	50
9. 収録内容の独自化	50
9.1 Includes	50
9.1.1 Live/chroot ローカルインクルード	50
9.1.2 バイナリローカルインクルード	51
9.2 フック	51
9.2.1 Live/chroot ローカルフック	51
9.2.2 ブート時フック	52
9.2.3 バイナリローカルフック	52
9.3 Debconf 質問の preseed	52
実行時の挙動の独自化	53
10. 実行時の挙動の独自化	53
10.1 live ユーザの独自化	53
10.2 ロケールと言語の独自化	53
10.3 保持機能	55
10.3.1 persistence.conf ファイル	57
10.3.2 保持先を複数使いたい場合	57
10.4 暗号化した保持先の利用	58
バイナリイメージの独自化	61
11. バイナリイメージの独自化	61
11.1 ブートローダ	61
11.2 ISO メタ情報	61
Debian インストーラの独自化	63
12. Debian インストーラの独自化	63
12.1 Debian インストーラの種類	63
12.2 preseed による Debian インストーラの独自化	64
12.3 Debian インストーラの収録内容の独自化	64
プロジェクト	65

プロジェクトへの貢献	66
13. プロジェクトへの貢献	66
13.1 変更を加える	66
バグの報告	68
14. バグの報告	68
14.1 既知の問題	68
14.2 最初から再ビルド	68
14.3 最新のパッケージを使う	69
14.4 情報収集	69
14.5 可能であれば失敗している状況を分離する	70
14.6 正しいパッケージに対してバグを報告する	70
14.6.1 ビルド時のパッケージ収集中	70
14.6.2 ビルド時のパッケージインストール中	70
14.6.3 ブート時	71
14.6.4 実行時	71
14.7 調査してください	71
14.8 バグの報告先	71
コーディングスタイル	73
15. コーディングスタイル	73
15.1 互換性	73
15.2 インデント	73
15.3 改行	73
15.4 変数	74
15.5 その他	75
手順	76
16. 手順	76
16.1 主要リリース	76
16.2 ポイントリリース	76
16.2.1 ある Debian リリースの最後のポイントリリース	76
16.2.2 ポイントリリース告知用テンプレート	76

Git リポジトリ	78
17. Git リポジトリ	78
17.1 リポジトリを複数処理	79
 例	 80
 例	 81
18. 例	81
18.1 例の使用	81
18.2 チュートリアル 1: デフォルトイメージ	81
18.3 チュートリアル 2: ウェブブラウザユーティリティ	82
18.4 チュートリアル 3: 私的イメージ	83
18.4.1 最初の改訂	83
18.4.2 2 回目の改訂	84
18.5 VNC 公衆クライアント	85
18.6 128MB USB メモリ向けの基本イメージ	86
18.7 地域化した GNOME デスクトップとインストーラ	87
 付録	 89
 スタイルガイド	 90
19. スタイルガイド	90
19.1 著者向けガイドライン	90
19.1.1 言語特性	90
19.1.2 手順	92
19.2 翻訳者向けガイドライン	94
19.2.1 翻訳の手がかり	94
 SiSU Metadata, document information	 97

Live システムマニュアル

1

About

2

このマニュアルについて

1. このマニュアルについて

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。

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題は扱いませんが、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-build* 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。

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。

1.1 せっかちな人向け

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。

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ます。<例の使用>を最初に読み、引き続いて<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。

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。

1.2 用語

- **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改変しません。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。13
- **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が置かれている任意の場所を指すこともあります。14
- **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。15
- **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。16
- **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。17
- ***live-boot***：Live システムのブートに利用するスクリプト集です。18
- ***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。19
- ***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。20
- ***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。21
- ***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。22
- **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。23
- **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。24
- **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。25
- **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。26
- **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。27

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

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が、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
```

サポートされている全言語のマニュアルをビルドするの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都合がかもしれません。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:

```
$ 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は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 (l10n) 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。

Live システムプロジェクトへの貢献 72

2. Live システムプロジェクトについて 73

2.1 動機 74

2.1.1 現在の Live システムの問題点 75

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。 76

- 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。 77
- 異なるディストリビューション、例えば `*{安定版 (testing)}*` と `*{不安定版 (unstable)}*` を混ぜて使っている。 78
- サポートしているのが i386 だけ。 79
- 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。 80
- Debian アーカイブ外のパッケージを収録している。 81
- 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。 82
- 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。 83
- 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。 84

2.1.2 自身の Live システムを作成する理由 85

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: 86

- これは Debian のサブプロジェクトです。 87
- 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
- 可能な限り多くのアーキテクチャで動作します。 89
- 変更しない Debian パッケージだけで構成されます。 90
- Debian アーカイブにないパッケージは何も含まれません。 91
- 改変しない Debian のカーネルを追加パッチなしで利用します。 92

2.2 哲学

2.2.1 Debian 「main」 の変更しないパッケージしか使いません

「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。

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。

例外として、*live-boot* や *live-build*、*live-config* 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。

2.2.2 Live システム固有のパッケージ設定はありません

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。

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メンテナと調整を行います。

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、〈独自化概要〉を見てください。

2.3 連絡先

- **メーリングリスト**：プロジェクトの第一の連絡先は <https://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 <https://lists.debian.org/debian-live/> で利用できます。
- **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ってください。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。
- **BTS**：〈バグの報告〉を見てください。

ユーザ

106

インストール 107

3. インストール 108

3.1 要件 109

Live システムイメージのビルドにはわずかながらシステム要件があります: 110

- スーパーユーザ (root) 権限 111
- 最新版の *live-build* 112
- *bash* や *dash* 等の POSIX に準拠したシェル 113
- *debootstrap* または *cdebootstrap* 114
- Linux 2.6 以降。 115

Debian や Debian 派生ディストリビューションの利用は必須ではないことに注意してください - *live-build* は上記の要件を満たすほぼありとあらゆるディストリビューションで動作します。 116

3.2 *live-build* のインストール 117

live-build のインストールにはいくつか方法があります: 118

- Debian リポジトリから 119
- ソースから 120
- スナップショットから 121

Debian を使っている場合に推奨するのは Debian リポジトリからの *live-build* のインストールです。 122

3.2.1 Debian リポジトリから 123

他のあらゆるパッケージと同様に、単に *live-build* をインストールします: 124

125

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

3.2.2 ソースから 126

live-build は Git バージョン管理システムを使って開発されています。Debian ベースのシステムでは *git* パッケージで提供されています。最新のコードを取得するには 127

128

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

を実行します。Debian パッケージを自分でビルド、インストールすることもできます。

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

を実行し、新しくできた .deb ファイルから対象のものをインストールします。例えば

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

システムに *live-build* を直接インストールすることもできます:

```
# make install
```

アンインストールは:

```
# make uninstall
```

3.2.3 「スナップショット」から

live-build をソースからビルドあるいはインストールしたくない場合、スナップショットを利用できます。スナップショットは Git の最新版から自動的にビルドされ、<http://live-systems.org/debian/> から利用できるようになっています。

3.3 live-boot と live-config のインストール

注意: 独自の Live システムを作成するためにシステムに *live-boot* や *live-config* をインストールする必要はありません。インストールは無害で、参照目的で有用でもあります。文書だけを望む場合には *live-boot-doc* や *live-config-doc* パッケージを別々にインストールできるようになっています。

3.3.1 Debian リポジトリから

live-boot と *live-config* はどちらも、*live-build* のインストールにあるように Debian リポジトリから利用できるようになっています。

3.3.2 ソースから

git から最新のソースを利用するには以下の処理を追ってください。〈用語〉で触れている用語について必ずよく理解しておくようにしてください。

- *live-boot* 及び *live-config* のソースの取得

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

パッケージをソースからビルドする理由が独自化である場合は、独自化の詳細について *live-boot* や *live-config* の man ページを参考にしてください。

- *live-boot* 及び *live-config* の .deb ファイルのビルド

ビルドは対象ディストリビューションまたは対象のプラットフォームを収録している chroot で行う必要があります: これはつまり、対象が **jessie** であれば **jessie** に対してビルドすべきだということです。

ビルドシステムとは異なるディストリビューションを対象とする *live-boot* をビルドする必要がある場合は *pbuilder* や *sbuild* といった個人向けビルダーを使ってください。例えば **jessie** の Live イメージであれば *live-boot* を **jessie** の chroot でビルドしてください。対象のディストリビューションがビルドシステムのディストリビューションと一致している場合はビルドシステムで直接 (*dpkg-dev* パッケージにより提供される) *dpkg-buildpackage* を使ってビルドできます:

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

- 件の .deb ファイルの利用

live-boot と *live-config* は *live-build* システムによりインストールされるため、ホストシステムでパッケージをインストールするだけでは十分ではありません: 生成された .deb ファイルを他の独自パッケージと同じように扱う必要があります。ソースからビルドする目的は恐らく公式リリース前の短期間に新しいものをテストすることなので、〈変更したあるいはサードパーティ製パッケージのインストール〉に従って、関連するファイルを設定に一時的に収録するようにしてください。特に、どちらのパッケージも一般的な部分、文書、そしてバックエンドに分割されていることに注意して

ください。一般的な部分と設定に合うバックエンドをただ 1 つ、オプションで文書を収録してください。Live イメージを現在のディレクトリでビルドし、前述のディレクトリに両方のパッケージの単一バージョンの .deb ファイルを全て生成したものと仮定して、以下の bash コマンドでデフォルトのバックエンドを含めて関連するパッケージを全てコピーします:

154

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

3.3.3 「スナップショット」から

155

live-build の設定ディレクトリで live-systems.org のパッケージリポジトリをサードパーティリポジトリとして設定することで、*live-build* が自動的に *live-boot* と *live-config* の最新のスナップショットを利用するようにできます。

156

基本

157

4. 基本

158

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。

159

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。

160

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。

161

4.1 Live システムとは何?

162

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）。

163

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。

164

- **Linux** カーネルイメージ、通常 `vmlinux*` という名前です
- 初期 **RAM** ディスクイメージ (**initrd**) : Linux ブート用に用意された 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るようにしています。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）。

165

166

167

- 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。

`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。

4.2 ビルド済みイメージのダウンロード

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 <<https://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/>> から利用できるようになっています。

4.3 ウェブ Live イメージビルダーの利用

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/>> 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。

4.3.1 ウェブビルダーの使い方と注意

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っている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。

4.4 最初の段階: ISO hybrid イメージのビルド

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。

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。

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

それから *lb config* コマンドを実行します。これにより他のコマンドが利用する「config/」階層を現在のディレクトリに作成します。

```
$ lb config
```

上記のコマンドにはパラメータが渡されていないので、様々な選択肢についてそれぞれのデフォルト値が使われます。さらなる詳細については〈*lb config* コマンド〉を見てください。

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。

```
# 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。

注意: amd64 システムでビルドした場合は、出来上がるイメージの名前は live-image-amd64.hybrid.iso となります。マニュアル全体でこの慣例を採用していることに留意してください。

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 <https://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

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:

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

4.5.2 ISO hybrid イメージの USB メモリへのコピー

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。

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!

```
$ cp live-image-i386.hybrid.iso ${USBメモリ}
$ sync
```

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

`live-image-i386.hybrid.iso` を USB メモリにコピーすると、最初のパーティションは Live システムで埋められます。残った空きスペースを利用するには、`gparted` や `parted` といったパーティシ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を作成します。

```
# gparted ${USBメモリ}
```

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あります。選択肢には `ext4` 等があります。`${パーティション}` には例えば `/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}
```

注意: 余った容量を Windows で使いたい場合ですが、この OS 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」でいくらか議論されていますが、簡単な解はないようです。

Remember: 新しい `live-image-i386.hybrid.iso` を USB メモリに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

Live メディア CD、DVD、USB メモリ、あるいは PXE ブートでの初回ブート時に、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 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。それが無い場合は 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。

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。

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。〈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。

4.6 仮想マシンを利用したテスト

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：

- 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。
- 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。
- 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。
- 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。

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。

4.6.1 QEMU での ISO イメージのテスト

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。

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。

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

ISO イメージのブートは簡単です：

```
$ 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。

4.6.2 VirtualBox での ISO イメージのテスト

virtualbox で ISO をテストするには:

```
# 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。

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 *virtualboxbox-guest-dkms* 及び *virtualboxbox-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。

```
$ 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。

```
$ 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:

```
# 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:

```
$ 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:

```
# 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るはずです。

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が異なります。

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。

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

4.8 netboot イメージのビルド

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。

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:


```
# lb clean
```

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。

`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:

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

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。

それから `lb build` コマンドでイメージをビルドします:

```
# lb build
```

ネットワーク経由のブートでは、クライアントは通常イーサネットカードの 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 `tftpboot/` 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;
}
```

4.8.2 TFTP サーバ

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ます。

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
```

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。

4.8.3 NFS サーバ

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ています。

nfs-kernel-server パッケージをインストールする必要があります。

それから /etc/exports に

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

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
```

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。

4.8.4 ネットワーク経由のブートをテストする方法

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。

日常を楽にするために仮想化を利用できます。

4.8.5 Qemu

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

/etc/qemu-ifup を編集します:

```
#!/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
```

grub-floppy-netboot を取得またはビルドします。

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

4.9 ウェブブート

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。

4.9.1 ウェブブートファイルの取得

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
```

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。

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すが `root` 権限を必要としません。

4.9.2 ウェブブートイメージの起動

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。

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：

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

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。

295

ツールの概要

5. ツールの概要

この章では Live システムのビルドに使う 3 つの主要ツール *live-build*、*live-boot*、*live-config* の概要をまとめています。

5.1 *live-build* パッケージ

live-build は Live システムをビルドするためのスクリプト集です。収録されているスクリプトは「コマンド」としても言及されています。

live-build の背景となる考え方は、設定ディレクトリを使って Live イメージのビルドに関するあらゆる面を完全に自動化、独自化するフレームワークにしようということです。

その多くの概念は *debhelper* で Debian パッケージをビルドするのと同様です：

- スクリプトには操作を制御するために中心となる位置があります。*debhelper* ではパッケージツリーの *debian/* サブディレクトリがこれにあたります。例えば *dh_install* は、他にもありますが、*debian/install* というファイルを探して特定のバイナリパッケージに収録すべきファイルを判断します。ほぼ同様に *live-build* は設定全体を *config/* サブディレクトリ以下に保管します。
- スクリプトは独立しています - つまり、各コマンドの実行は常に安全です。

debhelper とは異なり、*live-build* は設定ディレクトリの骨格を生成するツールを提供しています。これは *dh-make* 等のツールに似ていると言っても良いでしょう。こういったツールのさらなる情報については、この節の残りで最も重要な 4 つのコマンドについて触れているので続きを読んでください。各コマンドで先行している *lb* は *live-build* コマンドのラッパーであることに注意してください。

- lb config** : Live システム設定ディレクトリの初期化を担当します。さらなる情報については `<lb config コマンド>` を見てください。
- lb build** : Live システムのビルドの開始を担当します。さらなる情報については `<lb build>` を見てください。
- lb clean** : Live システムでビルドされた部分の掃除を担当します。さらなる情報については `<lb clean コマンド>` を見てください。

5.1.1 *lb config* コマンド

`<live-build>` で説明しているように、*live-build* を構成するスクリプトは *config/* という名の単一のディレクトリから *source* コマンドで指定された設定を読み込みます。このディレクトリを手作業により構成するのは時間がかかる上に誤りの元となりやすいため、*lb config* コマンドを使って初期設定ツリーの骨格を作成できるようになっています。

引数を付けずに `lb config` を実行すると `config/` サブディレクトリを作成してデフォルト設定がいくつか指定された設定ファイルを配置し、`auto/` 及び `local/` という骨格となる 2 つのツリーを作成します。

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

とても基本的なイメージを必要とするユーザや後で `auto/config` を使ってもっと全面的な設定を行おうと考えている場合は `lb config` を引数無しで使うのが適切でしょう (詳細は「設定管理」参照)。

通常はオプションをいくつか指定します。例えばイメージのビルド時に利用するパッケージマネージャーを指定する場合:

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

多数のオプションを指定することもできます:

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

利用可能なオプションの全容は `lb_config man` ページにあります。

5.1.2 lb build コマンド

`lb build` コマンドは `config/` ディレクトリから設定を読み込みます。それから Live システムのビルドに必要な低レベルコマンドを実行します。

5.1.3 lb clean コマンド

ビルドによる様々な生成物を削除するのは `lb clean` コマンドの役目で、それによりその後のビルドがきれいな状態から始められるようになります。デフォルトで `chroot`、バイナリ、ソースの段階が対象となりますが、キャッシュはそのまま残されます。また、個々の段階を個別に掃除することもできます。例えばバイナリ段階にのみ影響のある変更を加えた場合は新しいバイナリをビルドする前に `lb clean --binary` を実行します。変更がパッケージ収集やパッケージのキャッシュを無効化するようなもの、例えば `--mode` や `--architecture`、`--bootstrap` を変更した場合は `lb clean --purge` を実行しないといけません。オプションの全容については `lb_clean man` ページを見てください。

5.2 live-boot パッケージ

323

live-boot は *live-build* 等により作成される Live システムのブート時に使用する *initramfs* の生成に利用する *initramfs-tools* のフックを提供するスクリプト集です。Live システムの ISO やネットワーク経由のブートに利用する tar アーカイブ、USB メモリ向けイメージも対象です。

324

ブート時にはルートファイルシステム (squashfs のような圧縮ファイルシステムのイメージであることが多い) が保存されている `/live/` ディレクトリを収録する読み取り専用メディアを探します。見つかった場合は Debian 類似システムでそのメディアからブートできるように、*aufs* を使って書き込み可能な環境を作成します。

325

Debian の初期 RAM ファイルシステムについてのさらなる情報は <http://kernel-handbook.alioth.debian.org/> にある Debian Linux カーネルハンドブックの *initramfs* の章にあります。

326

5.3 live-config パッケージ

327

live-config はブート時に *live-boot* の後に実行して Live システムを自動的に設定するためのスクリプト集で構成されています。ホスト名やロケール、タイムゾーンの設定や *live* ユーザの作成、*cron* ジョブの抑制、*live* ユーザの自動ログイン等のタスクを処理します。

328

設定の管理

329

6. 設定の管理

330

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。

331

6.1 設定変更への対応

332

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。

333

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？

334

`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。

335

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。

336

6.1.2 自動化スクリプトの使用例

337

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

338

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:

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

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:

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

これで、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。

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。

6.2 Git 経由で公開されている設定の複製

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。

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

```
$ 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。 348

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 349

```
[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: 350

```
$ 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。 351

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で行う必要があるこ
とを覚えておいてください。 352

収録内容の独自化

355

7. 独自化の概要

356

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。

357

7.1 ビルド時とブート時の設定

358

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」を見てください。

359

ビルド時設定オプションは `lb config` の man ページで説明されています。ブート時オプションは *live-boot* と *live-config* の man ページで説明されています。*live-boot* 及び *live-config* パッケージはビルドする Live システム内にインストールされますが、設定作業時に参照しやすいようにビルドシステムにもインストールすることを勧めます。収録されているスクリプトはどれも、そのシステムが Live システムとして設定されていないと実行されないため、ビルドシステムへのインストールは安全です。

360

7.2 ビルド段階

361

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。

362

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

363

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。

7.3 ファイルによる `lb config` の補完

`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アートワーク、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。

7.4 独自化タスク

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>

インストールするパッケージの独自化

368

8. インストールするパッケージの独自化

369

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 のピン経由で制御する必要がある、等便利だと思う場面があるかもしれません。

370

8.1 パッケージソース

371

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

372

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:

373

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

374

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

375

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

376

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。

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ため、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。

8.1.2 ディストリビューションミラー

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になります。

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

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。

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

`--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

`--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` 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。

387

```
$ 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

388

リポジトリをさらに追加して、利用できるパッケージ選択の幅を対象ディストリビューション以外にも広げられます。それにより、例えば backports や experimental、そして独自のパッケージを利用できます。追加のリポジトリを設定するには config/archives/your-repository.list.chroot や config/archives/your-repository.list.binary ファイルを作成します。--mirror-* オプションにより、イメージのビルドの **chroot** 段階や **{バイナリ}** 段階、つまり Live システムの実行時に使用するリポジトリを決定できます。

389

例えば config/archives/live.list.chroot により Live システムのビルド時に debian-live スナップショットリポジトリからパッケージをインストールできます。

390

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

同一の行を config/archives/live.list.binary に追加すると、Live システムの /etc/apt/sources.list.d/ ディレクトリにそのリポジトリが追加されます。

392

こういったファイルが存在すれば自動的に処理されます。

393

リポジトリの署名に利用された GPG 鍵を config/archives/your-repository.key.{binary,chroot} ファイルに置くこともできます。

394

APT のピン止めが独自に必要な場合、APT 設定行等を config/archives/your-repository.pref.{binary,chroot} ファイルに配置すれば自動的に Live システムの /etc/apt/preferences.d/ ディレクトリに追加されます。

395

8.2 インストールするパッケージの選択

396

イメージに *live-build* がインストールするパッケージを選択する方法はいくつかあり、異なる様々な要求に対応します。インストールする個々のパッケージをパッケージ一覧で指定できます。その一覧でメタパッケージを利用あるいはパッケージ制御ファイル fields を使って選択できます。そして最後に、パッケージファイルを config/ ツ

397

リーに置く方法があります。この方法は、新しいあるいは試験的なパッケージをリポジトリから利用できるようになる前にテストするのに非常に適しています。

8.2.1 パッケージ一覧

パッケージ一覧はインストールするパッケージを明確にする強力な方法です。一覧の構文では条件付けをサポートし、一覧の生成や複数の設定への適合を容易にしています。ビルド時にシェルヘルパーを使って一覧にパッケージ名を差し込むこともできます。

注意: 存在しないパッケージが指定されたときの *live-build* の挙動は APT ユーティリティの選択により決定されます。さらなる詳細については「*apt* と *aptitude* の選択」を見てください。

8.2.2 メタパッケージの利用

パッケージ一覧を指示するもっとも簡単な方法は利用するディストリビューションで保守されているタスクのメタパッケージの利用です。例えば:

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

live-build 2.x でサポートされていた、一覧を事前定義する古い方法はこれで置き換えられました。一覧の事前定義とは異なり、タスクのメタパッケージは Live システムプロジェクト特有のものではありません。タスクのメタパッケージはディストリビューション内の専門の作業グループにより保守されているため、要求したユーザに対して提供する最善のパッケージについて、各グループでの合意を反映したものとなっています。また、一覧を事前定義する置き換えられた方法よりはるかに幅広い事例にも対応できます。

タスクのメタパッケージには全て先頭が *task-* から始まるため、利用できるものを簡単に判別する方法があります (名前が該当しても実際にはメタパッケージではないものもほんの一部とはいえありますが)。パッケージ名を前方一致で検索します:

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

以上に加え、他に様々な目的を持ったメタパッケージを見つけられるでしょう。*gnome-core* のように他のもっとも範囲の広いタスクパッケージの一部を構成するものや、*education-** メタパッケージのように Debian Pure Blend の中のある個々の専門分野に特化したものもあります。アーカイブにある全メタパッケージを列挙するには、*debtags* パッケージをインストールして *role::metapackage* タグの付けられたパッケージを全て列挙させます:

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

8.2.3 ローカルパッケージ一覧

409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

このディレクトリに存在するパッケージ一覧は、処理対象とするためには後ろに .list を付ける必要があり、さらにその一覧をどの段階の対象とするのか示すためには .chroot や .binary をその後に追加します。

注意: 対象とする段階の指定を追加しない場合、その一覧は両方の段階で利用されます。通常指定するのは .list.chroot で、この場合そのパッケージは Live ファイルシステムにのみインストールされ、メディア上に .deb の余計なコピーは置かれません。

8.2.4 ローカルバイナリパッケージ一覧

バイナリ段階の一覧を作成する場合はファイルの末尾を .list.binary にして config/package-lists/ に置きます。それにより指定したパッケージは Live ファイルシステムにはインストールされず、Live メディアの pool/ 以下に収録されます。Live ではないインストーラでこういった一覧を標準的に利用しているものもあります。バイナリ段階で chroot 段階と同一の一覧を利用したい場合は上述したように末尾を .list とします。

8.2.5 生成されたパッケージ一覧

一覧の構成はスクリプトで生成するのが最善の方法だということもあります。感嘆符から始まる行は全て、そのコマンドがイメージのビルド後に chroot 内で実行されることを示します。例えばパッケージ一覧に ! grep-aptavail -n -sPackage -FPriority standard |sort という行を書いておけば、Priority: standard で利用可能なパッケージをソートした一覧を生成できます。

実際、パッケージの選択に grep-aptavail コマンド (dctrl-tools パッケージに収録) はとても有用なので、live-build では便宜のため Packages 補助スクリプトを提供しています。このスクリプトは引数を #{フィールド}# と #{パターン}# の 2 つ取ります。一覧を作成する例:

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

8.2.6 条件付き内部パッケージ一覧の利用

config/* (LB_ が先頭に付くものは除く) に保存された *live-build* の設定変数はどれもパッケージ一覧の条件文で利用できます。一般には lb config オプションを大文字に、ダッシュ文字をアンダースコアに変更したものになります。しかし実際に意味があるのは、パッケージ選択に関わるもの、例えば DISTRIBUTION や ARCHITECTURES、ARCHIVE_AREAS だけです。

例えば --architectures amd64 が指定された場合に ia32-libs をインストールする場合:

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

任意の 1 つを条件の値とすることもできます。--architectures i386 または --architectures amd64 のどちらかが指定された場合に memtest86+ をインストールする場合の例:

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

値を複数指定できる変数を条件にすることもできます。--archive-areas で contrib または non-free のどちらかが指定されている場合に vrms をインストールする場合の例:

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

入り組んだ条件分岐はサポートしていません。

8.2.7 インストール時のパッケージの削除

config/package-lists ディレクトリの末尾が .list.chroot_live や .list.chroot_install のファイルにパッケージを列挙できます。Live 用とインストール用の両方の一覧が存在する場合、.list.chroot_live に列挙されているパッケージは (ユーザがイ

ンストローラを利用している場合) インストール後にフックにより削除されます。
.list.chroot_install に列挙されているパッケージは Live システムとインストールされたシステムの両方に存在することになります。これはインストーラ向けの特別な調整で、設定で --debian-installer live をセットしている場合や Live システム特有のパッケージをインストール時には削除したい場合に有用かもしれません。

8.2.8 デスクトップ及び言語タスク

デスクトップと言語のタスクは特別で、計画性や設定が追加で必要となります。Live イメージが Debian インストーライメージとは異なる点です。Debian インストーラでは、特定のデスクトップ環境向けに用意されたメディアでは対応するタスクは自動的にインストールされます。内部に gnome-desktop や kde-desktop、lxde-desktop、xfce-desktop タスクがあり、tasksel のメニューにはどれも出てきません。同様に言語向けタスクのメニュー項目はありませんが、インストール中にユーザが選択した言語が対応する言語タスクの選択に影響します。

デスクトップ向け Live イメージ開発時には、イメージは通常動作するデスクトップを直接ブートし、デスクトップやデフォルト言語はどちらも Debian インストーラの場合のように実行時に選択するのではなくビルド時に決められています。これは Live イメージでデスクトップや言語を複数サポートしてユーザに選択の機会を与えるようにできないわけではなく、それが *live-build* のデフォルトの挙動ではないというだけです。

言語特有のフォントや入力メソッドにどのパッケージを収録するのか、といった規定は言語タスクにはないので、特定のパッケージを収録したい場合は設定で指定する必要があります。ドイツ語サポートを収録した GNOME デスクトップイメージの場合に収録するタスクのメタパッケージの例:

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

8.2.9 カーネルのフレーバー(種類) とバージョン

アーキテクチャによっては、イメージに複数のカーネルをデフォルトで収録することができます。フレーバーは --linux-flavours オプションで選択できます。各フレーバーはデフォルトの短い linux-image に、イメージに収録される実際のカーネルパッケージに依存する各メタパッケージの名前を付加した形式になります。

そうして、デフォルトで amd64 アーキテクチャのイメージは linux-image-amd64 のメタパッケージを収録し、i386 アーキテクチャのイメージは linux-image-486 と linux-image-686-pae メタパッケージを収録します。これを書いている時点ではそれぞれ linux-image-3.2.0-4-amd64、linux-image-3.2.0-4-486、linux-image-3.2.0-4-686-pae に依存しています。

設定したアーカイブで複数バージョンのカーネルパッケージが利用できる場合、`--linux-packages` オプションでカーネルパッケージ名の前半部を指定できます。例えば amd64 アーキテクチャのイメージをビルドする際にテスト用に experimental アーカイブを追加すると `linux-image-3.7-trunk-amd64` カーネルをインストールできます。そのイメージの設定例:

```
$ 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 独自のカーネル

Debian パッケージ管理システムに組み入れられていれば、独自のカーネルを自分でビルド、収録できます。*live-build* システムは .deb パッケージでビルドされていないカーネルはサポートしていません。

自身のカーネルパッケージを配置するための適切で推奨する方法は `kernel-handbook` の指示に従うことです。パッケージ名の ABI とフレーバーの部分を忘れずに適切に変更し、リポジトリに `linux` の完全なビルドとそれに該当する `linux-latest` パッケージを収録してください。

該当するメタパッケージ無しでカーネルパッケージをビルドしたい場合は、`<カーネルのフレーバー(種類)とバージョン>` で説明しているように `--linux-packages` でパッケージ名の適切な前半部を指定する必要があります。`<変更したあるいはサードパーティ製パッケージのインストール>` で説明しているように、自身のリポジトリに独自のカーネルパッケージを収録する場合はそのようにするのが最善ですが、別の方法についても説明しています。

カーネルを独自化する方法はこの文書の対象範囲ではありません。とはいえ、設定が最低限満たさないといけない要件があります:

- 初期 RAM ディスクを利用する。
- 結合ファイルシステムモジュール (つまり通常は `aufs`) を収録する。
- 自分の設定で必要とする他のファイルシステムモジュール (つまり通常は `squashfs`) があればそれを収録する。

8.3 変更したあるいはサードパーティ製パッケージのインストール

Live システムの哲学には反しますが、Debian リポジトリにあるバージョンのパッケージを改変して Live システムをビルドする必要に迫られることもあります。それは機能や言語、商標を変更あるいは追加するものであったり、既存のパッケージから望ましくない要素を削除するものであるかもしれません。同様に求める機能や独自開発の機能を追加するのに「サードパーティ製」パッケージを利用できます。

この節では変更したパッケージのビルドや保守については対象としていません。

Joachim Breitner さんの <http://www.joachim-breitner.de/blog/archives/282-How-to-fork-privately.html> にある「How to fork privately」に書かれている方法が該当するのかもしれませんが。求める機能を収録したパッケージの作成については <https://www.debian.org/doc/maint-guide/> にある Debian 新メンテナガイドその他で説明されています。

変更した独自のパッケージをインストールする方法は 2 つあります:

- `packages.chroot`
- 独自 APT リポジトリの利用

`packages.chroot` を使う方法はより簡単に出来て「一度限りの」独自化には有用ですが欠点がいくつかあります。一方独自の APT リポジトリを使う方法はその準備に時間がかかります。

8.3.1 `packages.chroot` を利用した独自のパッケージのインストール

独自化したパッケージをインストールするには、単に `#{config/packages.chroot}/#` ディレクトリにコピーします。このディレクトリ内に置かれたパッケージはビルドの際に Live システムに自動的にインストールされます - 他のどこかを指定する必要はありません。

パッケージ名は規定の命名規則に従わ `*{ないといけません}*`。それを簡単に行う方法は `dpkg-name` の利用です。

`packages.chroot` を利用した独自のパッケージのインストールには欠点があります:

- `secure APT` を利用することができません。
- `config/packages.chroot/` ディレクトリに適切なパッケージを全てインストールしないといけません。
- Live システムの設定をリビジョン管理するには適しません。

8.3.2 APT リポジトリを利用した独自パッケージのインストール

`packages.chroot` を使う場合とは異なり、独自の APT リポジトリを使う場合は他のどこかでパッケージを指定する必要があります。詳細については「インストールするパッケージの選択」を見てください。

独自のパッケージをインストールするために APT リポジトリを作成するのは不要な手間だと思うかもしれませんが、その基盤は変更したパッケージを後で更新する際に簡単に再利用できます。

8.3.3 独自パッケージと APT

`live-build` は Live システムへのパッケージのインストールに全て APT を利用するため、そのプログラムの挙動を引き継ぎます。関連する一例としては (デフォルト設定

だと仮定して)、異なる 2 つのリポジトリでバージョン番号の異なるあるパッケージが利用可能な場合に、APT はバージョン番号の大きい方のパッケージをインストールに選択します。

そのため、独自パッケージの `debian/changeLog` ファイルでバージョン番号を増加させ、公式の Debian リポジトリにあるものよりも変更したバージョンが確実にインストールされるようにすると良いでしょう。Live システムの APT のピン設定を改変する方法もあります - さらに情報は、`<APT のピン止め>` を見てください。

8.4 ビルド時の APT 設定

ビルド時だけに適用されるオプションを使って APT を設定できます (実行中の Live システムで利用される APT の設定は Live システムの内容による通常の、`config/includes.chroot/` で適切な設定を収録する) 方法により設定できます。オプションの全容については `lb_config man` ページの `apt` で始まるオプションを見てください。

8.4.1 apt と aptitude の選択

ビルド時にパッケージをインストールする際に `apt` と `aptitude` のどちらを利用するのか選択できます。利用するユーティリティは `lb config` の `--apt` 引数で決定します。パッケージが欠けている場合の処理方法に顕著な違いがあることに着目し、パッケージのインストール時に望ましい挙動を実装している方を選択してください。

- `apt`: この方法では、欠けているパッケージが指定された場合にそのパッケージのインストールは失敗します。これはデフォルトの設定です。
- `aptitude`: この方法では、欠けているパッケージが指定された場合にそのパッケージのインストールは成功します。

8.4.2 APT でのプロキシの利用

よく要求される APT の設定として、プロキシの内側でのイメージのビルドへの対応があります。必要に応じて、`--apt-ftp-proxy` や `--apt-http-proxy` オプションにより APT プロキシを指定できます。例:

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

8.4.3 APT の調整による容量節約

イメージのメディアの容量をいくらか節約する必要があるかもしれません。その場合は以下に挙げるオプションを利用するといいかもかもしれません。

APT の索引をイメージに収録したくない場合は除外できます:

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

これは /etc/apt/sources.list 内の項目には影響せず、単に /var/lib/apt に索引ファイルを収録するかどうかだけに影響します。その代わりに、Live システムで APT を操作するためにはこの索引が必要なので、apt-cache search や apt-get install を実行する前にユーザは例えば apt-get update をまず実行して索引を作成しないといけません。

推奨パッケージのインストールによりイメージが肥大化しているような場合、以下で説明している影響を踏まえた上で APT のデフォルトオプションを無効にできます:

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

推奨パッケージを無効にした場合の最も重要な影響は、live-boot や live-config 自体が、ほとんどの Live 設定に利用している重要な機能を提供する一部のパッケージを推奨していることで、例えば live-config が推奨し、Live ユーザの作成に利用している user-setup があります。ほぼ例外なく、パッケージ一覧に追加しないといけないパッケージが推奨パッケージの中にいくらかあります。追加しない場合は、イメージが期待通りに動作しないということになります。ビルドに収録されている各 live-* パッケージの推奨パッケージを確認し、省略できると確信できない場合はパッケージ一覧に当該パッケージを追加するようにしてください。

あるパッケージの推奨パッケージをインストールしないことによるもっと一般的な影響は「異常なインストール状態でなければあるパッケージとともにあるはずの」(Debian ポリシーマニュアル 7.2 節) Live システムのユーザが実際に必要とする一部のパッケージが省略されているかもしれないということです。したがって、推奨パッケージのインストールを無効にした場合とのパッケージ一覧 (lb build により生成される binary.packages ファイル参照) の違いを確認して、欠けているパッケージの中にインストールしたいものがあれば一覧に収録することを推奨しています。推奨パッケージからほんの一部を除外したい場合は、推奨パッケージのインストールは有効なままにしておき、<APT のピン止め>で説明しているように APT のピン機能で、選択したパッケージについて負の優先度をセットしてインストールされないようにする方法があります。

8.4.4 apt や aptitude へのオプションの受け渡し

障害となる APT の挙動を変更するための lb config オプションがない場合、--apt-options や --aptitude-options により任意のオプションを選択した APT ツールに渡せます。詳細については apt や aptitude の man ページを見てください。どちらのオ

プションにも、優先設定に加えて保持しておく必要のあるデフォルト値があることに注意してください。そのため、例えば `snapshot.debian.org` からテスト用に何か取得する場合に `Acquire::Check-Valid-Until=false` を指定すると APT は古いままの Release ファイルを使い続けます。以下の例のように、デフォルト値 `--yes` の後に新しいオプションを付加します:

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

このオプションを利用する場合は `man` ページを確認して完全に理解するようにしてください。これはあくまで例であり、イメージをこの方法で設定するようという助言だと解釈することのないようにしてください。このオプションは例えば Live イメージの最終的なリリースには適しません。

`apt.conf` のオプションを利用するようなもっと複雑な APT 設定には代わりに `config/apt/apt.conf` ファイルを作成すると良いでしょう。少数ですが、よく必要とされるオプションへの有用なショートカットがあります。他の `apt-*` オプションについても参照してください。

8.4.5 APT のピン止め

背景として、`apt_preferences(5)` `man` ページをまず読んでください。APT のピン機能はビルド時用と実行時用に設定できます。前者については `config/archives/*.pref`、`config/archives/*.pref.chroot`、`config/apt/preferences` を、後者については `config/includes.chroot/etc/apt/preferences` を作成してください。

jessie の Live システムをビルドするとしましょう。その場合に必要な Live パッケージは全て、ビルド時にバイナリイメージを **sid** からインストールする必要があります。APT ソースに **sid** を追加して、ピン機能で **sid** の Live パッケージには高い優先度をセットし、他のパッケージには全てデフォルトよりも低い優先度をセットする必要があります。そうするとビルド時に望むパッケージだけを **sid** からインストールし、他は全て対象システムのディストリビューションである **jessie** から取得します。以下によりその動作になります:

```
$ 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
```

別のパッケージにより推奨されたパッケージを望まない場合に、ピン機能で優先度

に負の値をセットすることによりそのパッケージをインストールしないようにできます。config/package-lists/desktop.list.chroot で task-lxde-desktop を使って LXDE イメージをビルドしていて、wifi パスワードをキーリングに保存するかユーザに聞かないようにしたいと仮定しましょう。このメタパッケージは *lxde-core* に依存し、*lxde-core* は *gksu* を推奨し、*gksu* は *gnome-keyring* を推奨しています。そこで、推奨された *gnome-keyring* パッケージを除外したい場合、config/apt/preferences に以下を追加することで除外できます:

496

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

収録内容の独自化

9. 収録内容の独自化

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、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。

9.1 Includes

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:

- 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。
- 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。

「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。

9.1.1 Live/chroot ローカルインクルード

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。

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:

```
$ 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:

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

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。

9.1.2 バイナリローカルインクルード

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:

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

これでファイルは Live メディアの最上位ディレクトリに現れます。

9.2 フック

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。

9.2.1 Live/chroot ローカルフック

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。

9.2.2 ブート時フック

520

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、`<変更したあるい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。

521

9.2.3 バイナリローカルフック

522

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。

523

9.3 Debconf 質問の preseed

524

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。

525

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。

526

実行時の挙動の独自化

527

10. 実行時の挙動の独自化

528

実行時に行われる設定は全て *live-config* により行われます。ユーザが関心を持つであろう *live-config* の最も一般的なオプションから一部を説明します。オプションの全容は *live-config* の man ページにあります。

529

10.1 live ユーザの独自化

530

重要な検討事項が 1 つあり、live ユーザはブート時に *live-boot* により作成され、ビルド時に *live-build* により作成されるのではないということです。この影響は「Live/chroot ローカルインクルード」で説明しているように、ビルドで live ユーザに関連する内容が導入されるところだけにはとどまらず、live ユーザに関連するグループや権限にも影響します。

531

live-config を設定できる複数の方法で live ユーザの所属する追加のグループを指定できます。例えば live ユーザを fuse グループに追加するには `config/includes.-chroot/etc/live/config/user-setup.conf` ファイルに

532

533

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

を追加するかブートパラメータとして `live-config.user-default-groups=audio,cdrom,dip,floppy` と指定します。

デフォルトのユーザ名「user」やデフォルトのパスワード「live」を変更することもできます。何らかの理由で変更したい場合は以下のようにして簡単に変更できます:

535

デフォルトのユーザ名を変更するには単に設定で指定します:

536

537

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

デフォルトのパスワードを変更できる 1 つの方法は「ブート時フック」で説明しているフックを使います。そのためには `/usr/share/doc/live-config/examples/hooks` から「passwd」を使い、適当な名前 (例えば 2000-passwd) で保存してそれを `config/includes.chroot/lib/live/config/` に追加します。

538

10.2 ロケールと言語の独自化

539

Live システムがブートする際、2 つの段階で言語が関わってきます:

540

- ロケール生成

541

• キーボードの設定

Live システムをビルドする際のデフォルトのロケールは `locales=en_US.UTF-8` となっています。生成したいロケールの定義には `lb config` の `--bootappend-live` オプションで `locales` パラメータを指定します。例えば

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

ロケールをコンマで区切って複数指定することもできます。

このパラメータも以下に示すキーボード設定用パラメータと同様にカーネルコマンドラインで指定できます。ロケールは 言語 _ 国 (デフォルトのエンコーディングを使う場合) または完全な 言語 _ 国. エンコーディング の形式で指定できます。サポートしているロケールやそれぞれで利用されるエンコーディングの一覧は `/usr/share/i18n/SUPPORTED` にあります。

コンソールと X キーボードの設定はどちらも `live-config` により `console-setup` パッケージを使って行われます。設定には `--bootappend-live` オプション経由で `keyboard-layouts`、`keyboard-variants`、`keyboard-options`、`keyboard-model` ブートパラメータを利用します。それぞれの有効なオプションは `/usr/share/X11/xkb/rules/base.lst` にあります。ある言語向けのレイアウトや配列を見つけるには、その言語の英語名やその言語が話されている国を検索してみてください。例:

```
$ 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
```

それぞれの配列の説明に、適合するレイアウトが示されていることに注意してください。

レイアウトだけを設定する必要があることはよくあります。例えば X で利用するドイツ語のロケールファイル及びスイスのドイツ語のキーボードレイアウトを利用する場合:

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

非常に具体的な事例ですが他のパラメータを同時に指定することもできます。例えばフランス語のシステムを用意して TypeMatrix EZ-Reach 2030 USB キーボードで (Bepo と呼ばれる) フランス語用の Dvorak 配置を使う場合:

553

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

値を 1 つだけ受け付ける `keyboard-model` は例外ですが、他の `keyboard-*` オプションではそれぞれに値をコンマで区切って複数指定することもできます。`XKBMODEL` や `XKBLAYOUT`、`XKBVARIANT`、`XKBOPTIONS` 変数の詳細や例については `keyboard(5) man` ページを見てください。`keyboard-variants` に複数の値を指定した場合、1 つずつ `keyboard-layouts` の値 (`setxkbmap(1)` の `-variant` オプション参照) との照合が行われます。空白の値も使えます。例えばデフォルトとして米国向けの `QWERTY`、それとは別に米国向けの `Dvorak`、の 2 つの配列を指定する場合:

554

555

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

10.3 保持機能

556

典型的なライブ CD というものは CD-ROM 等の読み取り専用メディアから起動するインストール済みシステムで、書き込みや変更は起動したホストハードウェアの再起動により消え去ります。

557

Live システムはそれを一般化したものであり、CD 以外のメディアもサポートしますが、デフォルトの挙動としては読み取り専用であり、そのシステムで実行時に行ったことは全てシャットダウンにより失われるものだと考えるべきです。

558

「保持機能」というのは、実行時にシステムに行ったことの一部あるいは全てを保存し、リブート後に引き継ぐための様々な策の共通の呼び名です。それが機能する仕組みを理解するためには、読み取り専用メディアからブート、実行している場合でもファイルやディレクトリへの変更が書き込み可能メディア、標準的には RAM ディスク (`tmpfs`) に書かれ、RAM ディスクのデータはリブート後には残らないのだ、ということを知っておくと良いでしょう。

559

この RAM ディスクに保存されるデータは、ローカルストレージメディアやネットワーク共有、あるいはマルチセッションで (再) 書き込み可能な CD/DVD のセッション等の書き込み可能な保持用メディアに保存する必要があります。こういったメディアは Live システムで様々な方法でサポートされています。また、特別なブートパラメータ `persistence` をブート時に指定する必要があるということも重要です。

560

ブートパラメータ `persistence` がセットされている (と同時に `nopersistence` がセットされていない) 場合、保持用ボリューム用のローカルストレージメディア (例えばハードディスクや USB ドライブ) がブート中に調査されます。`live-boot(7) man` ページで説明している特定のブートパラメータを指定することにより、利用する保持用ボリュームの種類を制限できます。保持用ボリュームは以下のどれかになります:

561

- GPT 名により識別されるパーティション 562
- ファイルシステムラベルにより識別されるファイルシステム 563
- 読み取り可能な任意のファイルシステム (異質の OS の NTFS パーティションも利用可) の最上位に置かれている、ファイル名により識別されるイメージファイル 564

オーバーレイ用のボリュームラベルは persistence でないといけません。さらにその最上位に、ボリュームの保持を完全に制御するのに利用する persistence.conf というファイルが置かれていない限り無視されます。これは言うに及ばず、リブート後に保持用ボリュームに保存する対象となるディレクトリを指定します。さらなる詳細については「persistence.conf ファイル」をご覧ください。 565

保持用に利用するボリュームを用意する方法についていくつか例を示します。これは例えばハードディスクや USB メモリに例えば 566

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

により作成した ext4 パーティションを利用できます。「USB メモリの空きスペースの利用」もご覧ください。 568

デバイスに既存のパーティションがある場合は以下に示すどれかによりラベルを変更するだけで準備は終わりです: 569

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

保持用に利用する ext4 ベースのイメージファイルの作成例: 571

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

イメージファイル作成後、例えば /usr を保持させる場合に、保存するのはディレクトリに対する変更だけで、/usr の内容を丸ごと保存したいわけではない、という場合、「union」オプションを利用できます。イメージファイルがホームディレクトリに置かれている場合はハードドライブのファイルシステム最上位にコピーして /mnt にマウントします: 573

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

それから、内容を追加する persistence.conf ファイルを作成してイメージファイルのマウントを解除します。 575

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

ブートパラメータ「persistence」を指定して Live メディアでリブートします。

10.3.1 persistence.conf ファイル

persistence のラベルを付けられたボリュームは persistence.conf ファイルを使って、任意のディレクトリを保持するように設定します。ボリュームのファイルシステム最上位に置かれているこのファイルは保持するディレクトリや方法を制御します。

独自のオーバーレイマウントの設定方法は persistence.conf(5) man ページで詳細に説明されていますが、ほとんどの場合簡単な例で十分なはずです。/dev/sdb1 パーティションの ext4 ファイルシステムにホームディレクトリと APT キャッシュを保持させる場合:

```
# 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
```

それからリブートします。最初のブート中に /home と /var/cache/apt の内容が保持用ボリュームにコピーされ、以後のこのディレクトリへの変更は全て保持用ボリュームに残ります。persistence.conf ファイルに列挙するパスには空白文字や特別なパス構成要素 . and .. を含めることは一切できないことに注意してください。また、/lib や /lib/live (はそのサブディレクトリを含めて)、/ は独自マウントを使って保持させることができません。この制約の回避策として、persistence.conf ファイルに / union を追加すると完全な保持を実現できます。

10.3.2 保持先を複数使いたい場合

複数の保存先を使う方法は複数あります。目的の明確な例として、複数のボリュームを同時に使う場合と 1 つだけを選択して使う場合を取り上げます。

異なる (個別の persistence.conf ファイルを利用する) 独自オーバーレイボリュームを複数同時に利用できますが、同一のディレクトリを保持させる設定のボリュームが複数ある場合はその中から 1 つだけが利用されます。ある 2 つのマウントが「入り組んでいる」(つまり他にマウントしたディレクトリ以下のサブディレクトリとしてマウントする) ような場合には子孫側ディレクトリよりも前に親側ディレクトリがマウントされるため、他のマウントにより見えなくなるマウントは発生しません。入り組んでいる独自マウントが同一の persistence.conf ファイルで指定されている場合は問題があります。そういった状況が実際に必要な場合の処理方法につ

いては `persistence.conf(5)` man ページを見てください (ヒント: 通常は必要ありません)。

ありそうな事例: ユーザデータ、つまり `/home` とスーパーユーザのデータ、つまり `/root` を異なるパーティションに保存するため `persistence` ラベルの付いたパーティションを 2 つ作成し、それぞれに `persistence.conf` ファイルを 1 つはユーザのファイルを保存する最初のパーティション向けに `# echo "/home" > persistence.conf`、もう 1 つはスーパーユーザのファイルを保存する 2 つ目のパーティション向けに `# echo "/root" > persistence.conf` として作成します。最後に、ブートパラメータとして `persistence` を使います。

別の位置やテストのために例えば `private` と `work` のようにして同一の種類で複数の保持先が必要な場合、ブートパラメータ `persistence` と合わせてブートパラメータ `persistence-label` を使うと、複数でそれぞれ一意となる保持用メディアを使えるようになります。例として、ブラウザのブックマークその他の個人用データ用に `private` のラベルを付けられた保持用パーティションを使いたい場合、ブートパラメータとして `persistence persistence-label=private` を使います。そして文書や研究プロジェクトその他の仕事関係のデータ用にはブートパラメータとして `persistence persistence-label=work` を使います。

各ボリウム `private` 及び `work` にはそれぞれ最上位に `persistence.conf` ファイルが必要だということは重要ですので覚えておいてください。古い名前でこういったラベルを使う方法についてさらなる情報が `live-boot` man ページにあります。

10.4 暗号化した保持先の利用

保持機能を使うことには重要なデータが漏洩する危険がいくらか存在します。特に保持するデータが USB メモリや外付ハードドライブ等のポータブル機器に保存される場合は。そういった場合には暗号化が便利です。手順が多いために全体として複雑に見えるかもしれませんが、`live-boot` で暗号化したパーティションを扱うのは実際には簡単です。サポートしている **luks** という種類の暗号化を利用するためには、暗号化したパーティションを作成する側と暗号化した保持用パーティションを利用する Live システムの両方のマシンに `cryptsetup` をインストールする必要があります。

マシンへの `cryptsetup` のインストール:

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

Live システムに `cryptsetup` をインストールするには、パッケージ一覧に追加します:

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

`cryptsetup` を備えた Live システムが出来たら、基本的に必要なのは新しいパーティションを作成して暗号化し、`persistence` と `persistence-encryption=luks` パラメータを指定してブートするだけです。既にこの段階を予測し、通常の手順に沿ったブートパラメータを追加してあります:

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

暗号化についてよく理解していない人たちのために詳細を見て行きましょう。以下の例では `/dev/sdc2` に対応する USB メモリ上のパーティションを利用します。自分で使う際にはどのパーティションになるのか判断する必要があることに注意してください。

最初の段階は USB メモリを接続してそれがどのデバイスになるのか判断することです。*live-manual* で推奨するデバイス一覧方法では `ls -l /dev/disk/by-id` を使います。それから新しいパーティションを作成、さらにパスフレーズを使って暗号化します:

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

仮想デバイスマッパーから `luks` パーティションを開きます。好きな名前を使ってください。ここでは例として **live** を使います:

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

次の段階はファイルシステムを作成する前にデバイスをゼロで埋めることです:

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

これでファイルシステムを作成する準備ができました。ラベル `persistence` の指定を追加しているためこのデバイスはブート時に保持用としてマウントされるということに注意してください。

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

準備を続けるにはデバイスをマウントする必要があります。例として `/mnt` にマウントします。

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

そしてパーティション最上位に persistence.conf ファイルを作成します。これは前に説明したように必ず必要です。〈persistence.conf ファイル〉を見てください。

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

それからマウントポイントのマウントを解除します:

```
# umount /mnt
```

オプションですがパーティションに追加したばかりのデータを安全にしておくといいでしょう。デバイスを閉じます:

```
# cryptsetup luksClose live
```

手順をまとめます。これまでに、暗号化を有効化した Live システムを作成しました。これは「ISO hybrid イメージの USB メモリへのコピー」で説明しているように USB メモリにコピーできます。暗号化したパーティションも作成しました。これは同一の USB メモリに置いて持ち運べます。保持先として利用する暗号化パーティションを設定しました。あと必要なのは Live システムをブートするだけです。ブート時に live-boot は保持用として利用する暗号化したパーティションのパスフレーズを質問し、マウントします。

バイナリイメージの独自化

11. バイナリイメージの独自化

11.1 ブートローダ

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。

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。

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。

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。

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

```
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。

- `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。
- `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。

- `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つか含めます。最大長は 128 文字です。 628
- `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。 629

Debian インストーラの独自化

630

12. Debian インストーラの独自化

631

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かの異なる種類があります。

632

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。

633

12.1 Debian インストーラの種類

634

インストーラの主な 3 つの種類:

635

「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。

636

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。

637

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。

638

「**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。

639

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。

640

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。

641

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。

642

「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

643

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。

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:

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

12.2 preseed による Debian インストーラの独自化

<<https://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:

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

12.3 Debian インストーラの収録内容の独自化

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。

プロジェクト

651

プロジェクトへの貢献

652

13. プロジェクトへの貢献

653

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。

654

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。

655

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。

656

ただし、特定の制限があります。サーバは

657

- 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

658

659

660

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します。

661

- 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 に収録されることになります。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、各段落の後に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。

662

663

664

13.1 変更を加える

665

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。

666

- 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
```

- ssh 経由で *live-manual* の複製を取得します:

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

- 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:

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

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

- 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

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

- のように書いてサーバにコミットを送ります:

```
$ git push
```

バグの報告

14. バグの報告

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。

せっかちな人向け:

- 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。
- 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。
- 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。

14.1 既知の問題

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。

そのために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。

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。

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は **{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)。

14.2 最初から再ビルド

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してください。

14.3 最新のパッケージを使う

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。

14.4 情報収集

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。

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:

- ホストシステムのアーキテクチャ
- ホストシステムのディストリビューション
- ホストシステムの *live-build* のバージョン
- ホストシステムの Python のバージョン
- ホストシステムの *debootstrap* や *cdebootstrap* のバージョン
- Live システムのアーキテクチャ
- Live システムのディストリビューション
- Live システムの *live-boot* のバージョン
- Live システムの *live-config* のバージョン
- Live システムの *live-tools* のバージョン

`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)。

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

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。

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。

713

14.5 可能であれば失敗している状況を分離する

714

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か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ものが考えられ、その場合開発の進行は緩くなるはずです。

715

14.6 正しいパッケージに対してバグを報告する

716

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。

717

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。

718

14.6.1 ビルド時のパッケージ収集中

719

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。

720

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。

721

14.6.2 ビルド時のパッケージインストール中

722

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。

723

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 -

724

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。

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。

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だ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のも、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

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 <https://bugs.debian.org/> を見てください。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。

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

切なのかよくわからない、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。

(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。

737

コーディングスタイル

738

15. コーディングスタイル

739

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について述べます。

740

15.1 互換性

741

- Bash シェル固有の書式や記号を使わないでください。例えば配列構造の利用など
- POSIX のサブセットだけを使ってください - 例えば 'foo' よりも \$(foo) を使ってください。
- 'sh -n' と 'checkbashisms' によりスクリプトをチェックできます。
- 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さい。

742

743

744

745

15.2 インデント

746

- 常にスペースよりもタブを使います。

747

15.3 改行

748

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

749

750

悪い例:

751

752

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

良い例:

753

754

```
if foo
then
    bar
fi
```

- 関数についても同様です:

755

悪い例:

756

757

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

良い例:

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

15.4 変数

- 変数は常に大文字です。
- *live-build* で利用する変数は先頭を常に LB_ で始めます。
- *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 \$F00 ではなく \${F00} 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: \${F00} ではなく “\${F00}” 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 "${F00}"/foo/"${BAR}"/bar ]
then
    foobar
fi
```

良い例:

777

778

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

15.5 その他

779

- 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|'|`」780
- 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。781
- 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。782
- 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。783

手順

16. 手順

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。

16.1 主要リリース

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

- 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
- 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)。
- 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)。
- ビルドされたイメージのミラーが cdimage.debian.org に置かれます。

16.2 ポイントリリース

- ここでも、debian と debian-security の更新されたミラーが必要です。
- ビルドされたイメージのミラーが cdimage.debian.org に置かれます。
- 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

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

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。

```
$ 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'
```

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。

802

803

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

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

The images are available for download at:

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

and later at:

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

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

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

Additionally it includes the following Live-specific changes:

* [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>.
```

Git リポジトリ

17. Git リポジトリ

Live システムプロジェクトの利用可能な全リポジトリ一覧は <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

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

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

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

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。

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

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

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。

既存のリポジトリのどれかを複製した後は **debian** ブランチにいます。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

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

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。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。

17.1 リポジトリを複数処理

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、

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

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。

例

821

例

18. 例

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。

18.1 例の使用

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。

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:

```
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

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。

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。

これ以上簡単にすることはなかなかできないでしょう:

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

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。

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。

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

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

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。

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。

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

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。

「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:

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

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

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。

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

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

`auto/config` を以下のように変更します:

```
#!/bin/sh

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

`lb config` を実行して設定ツリーを生成し、生成された `auto/config` スクリプトを使います:

```
$ lb config
```

ここでローカルパッケージ一覧を設定します:

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

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` のタスク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。

そして、イメージをビルドします:

```
# lb build
```

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。

(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:

```
$ 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

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。

`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。

```
# lb clean
```

`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:

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

再びビルドします:

```
# lb build
```

テストして満足したら次の改訂としてコミットします:

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

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できたもので、`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。

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す。

18.5 VNC 公衆クライアント

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。

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。

```
$ 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
```

「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つ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:

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

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:

883

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

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:

884

885

```
$ 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
```

イメージをビルドします:

886

887

```
# lb build
```

楽しみましょう。

888

18.6 128MB USB メモリ向けの基本イメージ

889

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。

890

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。

891

892

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

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

893

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：

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

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。

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でさらに容量をいくらか節約します。そして最後に、 `--memtest none` によりメモリテストのインストールを抑制します。

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。

18.7 地域化した GNOME デスクトップとインストーラ

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

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のパッケージを全て収録します。

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:

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

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、

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

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:

```
$ 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:

```
$ 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
```

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。

付録

912

スタイルガイド 913

19. スタイルガイド 914

19.1 著者向けガイドライン 915

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。 916

注意: 著者はまず「この文書への貢献」を読んでください 917

19.1.1 言語特性 918

- 平易な英語を使う 919

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、引き続いて終止符を打ってください。 920

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。 921

- 英語の方言 922

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。 923

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。 924

- バランス良く 925

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。 926

- 政治的に正しく 927

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。 928

- 簡潔に 929

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。読み手の側にいくらか前提知識があることを仮定してください。 930

- 翻訳作業を最小限に

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。

- 一貫性を

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。

- 結束性を

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)。

- 記述的に

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。

- 辞書

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。

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ます。

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くだけた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。

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。

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。

- 空似言葉や熟語その他の慣習的な表現

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だけでもその意味や使い方が全く異なる「空似言葉」という罠にはまる可能性があることは避けられません。

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!

- 俗語や省略、短縮表現等は避けましょう

平易な、日常的な英語の使用を勧めるとはいっても、技術的文献は言語を正式に記録する類のものです。

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。

19.1.2 手順

- 書く前にテストを

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。

- 例

例示するとき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。

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を簡単に説明することもできます。

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。

- 外部リンク

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。

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。

- 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

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。

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。

- まず草稿を書き、改訂、変更して改善し、必要なら作り直す

- 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。

- 頭の中で何とか形ができたなら最初の草稿を書きます。

- 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。

- 章

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。

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。

- マークアップ

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
```

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

- 文字列の強調/太字:

```
*{foo}* または !{foo}!
```

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
```

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

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。

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。

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

- コメント

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。 996

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。 997

- *TN, Translator's Note* (翻訳者によるメモ) 998

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。 999

- 非人称の文を 1000

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られ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。 1001

- 空似言葉 1002

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。 1003

- マークアップ 1004

最初は **{pot}** ファイル、後には **{po}** 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。 1005

- コードブロック 1006

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。 1007

- 改行 1008

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 **\n** 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。 1009

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。 1010

- 翻訳できない、してはいけない文字列 1011

翻訳者が決して翻訳すべきでないもの:	1012
- リリースのコード名 (小文字で書くべき)	1013
- プログラムの名前	1014
- 例示するコマンド	1015
- メタ情報 (前後にコロンが置かれることが多い : メタ情報:)	1016
- リンク	1017
- パス	1018

SiSU Metadata, document information

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: 2014-10-25

Version Information

Sourcefile: live-manual.ssm.sst

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

Source Digest: SHA256(live-manual.ssm.sst)=c89c1989b8f7190efa7b806a748ee1bbab7aad4070830f63acf5c82b53833271-

Generated

Document (ao) last generated: 2014-10-25 13:16:36 +0000

Generated by: SiSU 5.7.1 of 2014w41/7 (2014-10-19)

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