

Debian Live Manual

Debian Live Project <debian-live@lists.debian.org>

November 21, 2011

Copyright © 2006-2011 Debian Live 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.

Contents

Debian Live Manual **1**

About **1**

1. About this manual 1

 1.1 For the impatient 1

 1.2 Terms 1

 1.3 Authors 2

 1.4 Contributing to this document 2

 1.4.1 Applying patches 3

 1.4.2 Translation 3

2. About the Debian Live Project 4

 2.1 Motivation 4

 2.1.1 What is wrong with current live systems 4

 2.1.2 Why create our own live system? 4

 2.2 Philosophy 4

 2.2.1 Only unchanged packages from Debian
 “main” 4

 2.2.2 No package configuration of the live system 5

 2.3 Contact 5

User **5**

3. Installation 5

 3.1 Requirements 5

 3.2 Installing live-build 6

 3.2.1 From the Debian repository 6

 3.2.2 From source 6

 3.2.3 From `snapshots' 6

 3.3 live-boot and live-config 6

 3.3.1 From the Debian repository 6

 3.3.2 From source 6

 3.3.3 From `snapshots' 7

4. The basics 7

 4.1 What is a live system? 7

 4.2 First steps: building an ISO hybrid image 8

 4.3 Using an ISO hybrid live image 8

 4.3.1 Burning an ISO image to a physical medium 8

 4.3.2 Copying an ISO hybrid image to a USB stick 9

 4.3.3 Booting the live media 9

 4.4 Using a virtual machine for testing 9

 4.4.1 Testing an ISO image with QEMU 10

 4.4.2 Testing an ISO image with virtualbox-ose 10

 4.5 Building a USB/HDD image 10

 4.6 Using a USB/HDD image 10

 4.6.1 Testing a USB/HDD image with Qemu 11

 4.6.2 Using the space left on a USB stick 11

 4.7 Building a netboot image 11

 4.7.1 DHCP server 12

 4.7.2 TFTP server 12

 4.7.3 NFS server 12

 4.7.4 Netboot testing HowTo 12

 4.7.5 Qemu 13

 4.7.6 VMWare Player 13

5. Overview of tools 13

 5.1 live-build 13

 5.1.1 The lb config command 14

 5.1.2 The lb build command 14

 5.1.3 The lb clean command 15

 5.2 The live-boot package 15

 5.3 The live-config package 15

6. Managing a configuration 15

 6.1 Use auto to manage configuration changes 15

 6.2 Example auto scripts 16

7. Customization overview 16

 7.1 Build time vs. boot time configuration 16

7.2 Stages of the build	17	9. Customizing contents	24
7.3 Supplement lb config with files	17	9.1 Includes	25
7.4 Customization tasks	17	9.1.1 Live/chroot local includes	25
8. Customizing package installation	17	9.1.2 Binary local includes	25
8.1 Package sources	18	9.1.3 Binary includes	26
8.1.1 Distribution, archive areas and mode	18	9.2 Hooks	26
8.1.2 Distribution mirrors	18	9.2.1 Live/chroot local hooks	26
8.1.3 Distribution mirrors used at build time	18	9.2.2 Boot-time hooks	26
8.1.4 Distribution mirrors used at run time	18	9.2.3 Binary local hooks	26
8.1.5 Additional repositories	19	9.3 Preseeding Debconf questions	26
8.2 Choosing packages to install	19	10. Customizing run time behaviours	26
8.2.1 Package lists	19	10.1 Customizing the live user	27
8.2.2 Predefined package lists	20	10.2 Customizing locale and language	27
8.2.3 Local package lists	20	10.3 Persistence	28
8.2.4 Local binary package lists	20	10.3.1 Full persistence	28
8.2.5 Extending a provided package list using includes	20	10.3.2 Home automounting	29
8.2.6 Using conditionals inside package lists	20	10.3.3 Snapshots	29
8.2.7 Tasks	21	10.3.4 Persistent SubText	29
8.2.8 Desktop and language tasks	21	10.3.5 Partial remastering	29
8.3 Installing modified or third-party packages	22	11. Customizing the binary image	29
8.3.1 Using packages.chroot to install custom packages	22	11.1 Bootloader	29
8.3.2 Using an APT repository to install custom packages	22	11.2 ISO metadata	29
8.3.3 Custom packages and APT	22	12. Customizing Debian Installer	30
8.4 Configuring APT at build time	23	12.1 Types of Debian Installer	30
8.4.1 Choosing apt or aptitude	23	12.2 Customizing Debian Installer by preseeding	31
8.4.2 Using a proxy with APT	23	12.3 Customizing Debian Installer content	31
8.4.3 Tweaking APT to save space	23	Project	31
8.4.4 Passing options to apt or aptitude	24	13. Reporting bugs	31
8.4.5 APT pinning	24	13.1 Known issues	32
		13.2 Rebuild from scratch	32
		13.3 Use up-to-date packages	32
		13.4 Collect information	32

13.5 Isolate the failing case if possible	33	Metadata	42
13.6 Use the correct package to report the bug against	33	SiSU Metadata, document information	42
13.6.1 At build time whilst bootstrapping	33		
13.6.2 At build time whilst installing packages	33		
13.6.3 At boot time	33		
13.6.4 At run time	33		
13.7 Do the research	34		
13.8 Where to report bugs	34		
14. Coding Style	34		
14.1 Compatibility	34		
14.2 Indenting	34		
14.3 Wrapping	34		
14.4 Variables	35		
14.5 Miscellaneous	35		
15. Procedures	35		
15.1 Udeb Uploads	36		
15.2 Major Releases	36		
15.3 Point Releases	36		
15.3.1 Point release announcement template	36		
Examples	37		
16. Examples	37		
16.1 Using the examples	37		
16.2 Tutorial 1: A standard image	38		
16.3 Tutorial 2: A web browser utility	38		
16.4 Tutorial 3: A personalized image	38		
16.4.1 First revision	39		
16.4.2 Second revision	39		
16.5 A VNC Kiosk Client	40		
16.6 A base image for a 128M USB key	40		
16.7 A localized KDE desktop and installer	41		

1 Debian Live Manual

2 About

3 1. About this manual

4 The main goal of this manual is to serve as a single access point to all documentation related to the Debian Live project. While it is primarily focused on helping you build a live system and not on end-user topics, an end-user may find some useful information in these sections: <[The Basics](#)> covers preparing images to be booted from media or the network, and <[Customizing run time behaviours](#)> describes some options that may be specified at the boot prompt, such as selecting a keyboard layout and locale, and using persistence.

5 Some of the commands mentioned in the text must be executed with superuser privileges which can be obtained by becoming the root user via `su` or by using `sudo`. To distinguish between commands which may be executed by an unprivileged user and those requiring superuser privileges, commands are prepended by `$` or `#` respectively. This symbol is not a part of the command.

6 1.1 For the impatient

7 While we believe that everything in this manual is important to at least some of our users, we realize it is a lot of material to cover and that you may wish to experience early success using the software before delving into the details. Therefore, we have provided three tutorials in the <[Examples](#)> section designed to teach you image building and customization basics. Read <[Using the examples](#)> first, followed by <[Tutorial 1: A standard image](#)>, <[Tutorial 2: A web browser utility](#)> and finally <[Tutorial 3: A personalized image](#)>. By the end of these tutorials, you will have a taste of what

can be done with Debian Live. We encourage you to return to more in-depth study of the manual, perhaps next reading <[The basics](#)>, skimming or skipping <[Building a netboot image](#)>, and finishing by reading the <[Customization overview](#)> and the chapters that follow it. By this point, we hope you are thoroughly excited by what can be done with Debian Live and motivated to read the rest of the manual, cover-to-cover.

8 1.2 Terms

- 9 • **Live system** : An operating system that can boot without installation to a hard drive. Live systems do not alter local operating system(s) or file(s) already installed on the computer hard drive unless instructed to do so. Live systems are typically booted from media such as CDs, DVDs or USB sticks. Some may also boot over the network.
- 10 • **Debian Live** : The Debian sub-project which maintains the *live-boot*, *live-build*, *live-config*, and *live-manual* packages.
- 11 • **Debian Live system** : A live system that uses software from the Debian operating system that may be booted from CDs, DVDs, USB sticks, over the network (via netboot images), and over the Internet (via boot parameter `fetch=URL`).
- 12 • **Host system** : The environment used to create the live system.
- 13 • **Target system** : The environment used to run the live system.
- 14 • **live-boot** : A collection of scripts used to boot live systems. *live-boot* was formerly a part of *live-initramfs*.
- 15 • **live-build** : A collection of scripts used to build customized Debian Live systems. *live-build* was formerly known as *live-helper*, and even earlier known as *live-package*.

	• live-config : A collection of scripts used to configure a live system during the boot process. <i>live-config</i> was formerly a part of <i>live-initramfs</i> .	16			
17	• live-manual : This document is maintained in a package called <i>live-manual</i> .			1.3 Authors	25
18	• Debian Installer (d-i) : The official installation system for the Debian distribution.			A list of authors (in alphabetical order):	26
19	• Boot parameters : Parameters that can be entered at the boot-loader prompt to influence the kernel or <i>live-config</i> .			• Ben Armstrong	27
20	• chroot : The <i>chroot</i> program, <i>chroot(8)</i> , enables us to run different instances of the GNU/Linux environment on a single system simultaneously without rebooting.			• Brendan Sleight	28
21	• Binary image : A file containing the live system, such as <i>binary.iso</i> or <i>binary.img</i> .			• Chris Lamb	29
22	• Target distribution : The distribution upon which your live system will be based. This can differ from the distribution of your host system.			• Daniel Baumann	30
23	• Squeeze/Wheezy/Sid (stable/testing/unstable) : Debian codenames for releases. At the time of writing, Squeeze is the current stable release and Wheezy is the current testing release. Sid will always be a synonym for the unstable release. Throughout the manual, we tend to use codenames for the releases, as that is what is supported by the tools themselves.			• Franklin Piat	31
24	The stable distribution contains the latest officially released distribution of Debian. The testing distribution is the staging area for the next stable release. A major advantage of using this distribution is that it has more recent versions of software relative to the stable release. The unstable distribution is where active development of Debian occurs. Generally, this distribution is run by developers and those who like to live on the edge.			• Jonas Stein	32
				• Kai Hendry	33
				• Marco Amadori	34
				• Mathieu Geli	35
				• Matthias Kirschner	36
				• Richard Nelson	37
				• Trent W. Buck	38
				1.4 Contributing to this document	39
				This manual is intended as a community project and all proposals for improvements and contributions are extremely welcome. The preferred way to submit a contribution is to send it to the mailing list. Please see the section Contact for more information.	40
				When submitting a contribution, please clearly identify its copyright holder and include the licensing statement. Note that to be accepted, the contribution must be licensed under the same license as the rest of the document, namely, GPL version 3 or later.	41
				The sources for this manual are maintained using the Git version	42

control system. You can check out the latest copy by execut- 55
ing:

```
43 $ git clone git://live.debian.net/git/live-manual.git
```

44 Prior to submission of your contribution, please preview your work.
To preview the *live-manual*, ensure the packages needed for build-
ing are installed by executing:

```
45 # apt-get install make po4a sisu-complete  
libnokogiri-ruby
```

46 You may build the *live-manual* from the top level directory of your
Git checkout by executing:

```
47 $ make build
```

48 Since it takes a while to build the manual in all supported lan-
guages, you may find it convenient when proofing to build for only
one language, e.g. by executing:

```
49 $ make build LANGUAGES=en
```

50 1.4.1 Applying patches

51 Anyone can directly commit to the repository. However, we ask you
to send bigger changes to the mailing list to discuss them first. To
push to the repository, you must follow this procedure:

- 52 • Fetch the public commit key:

```
53 $ mkdir -p ~/.ssh/identity.d  
$ wget http://live.debian.net/other/keys/git@live.debian.net \  
-O ~/.ssh/identity.d/git@live.debian.net  
$ wget http://live.debian.net/other/keys/git@live.debian.net.pub \  
\  
-O ~/.ssh/identity.d/git@live.debian.net.pub  
$ chmod 0600 ~/.ssh/identity.d/git@live.debian.net*
```

- 54 • Add the following section to your openssh-client config:

```
$ cat >> ~/.ssh/config << EOF  
Host live.debian.net  
    Hostname live.debian.net  
    User git  
    IdentityFile ~/.ssh/identity.d/git@live.debian.net  
EOF
```

- Check out a clone of the manual through ssh: 56

```
$ git clone git@live.debian.net:/live-manual.git 57  
$ cd live-manual && git checkout debian-next
```

- Note that you should commit any changes on the `debian-next` 58
branch, not on the `debian` branch.

- After editing the files in `manual/en/`, please call the ``commit'` tar- 59
get in the top level directory to sanitize the files and update the
translation files:

```
$ make commit 60
```

- After sanitizing, commit the changes. Write commit messages 61
that consist of full, useful sentences in English, starting with a
capital letter and ending with a full stop. Usually, these will start
with the form ``Fixing/Adding/Removing/Correcting/Translating'`,
e.g.

```
$ git commit -a -m "Adding a section on applying 62  
patches."
```

- Push the commit to the server: 63

```
$ git push 64
```

65 1.4.2 Translation

To submit a translation for a new language, follow these three 66
steps:

- Translate the `about_manual.ssi.pot`, `about_project.ssi.pot` and `index.html.in.pot` files to your language with your favourite editor (such as `poedit`). Send translated files to the mailing list. Once we have reviewed your submission, we will add the new language to the manual (providing the `po` files) and will enable it in the autobuild. 7867
- Once the new language is added, you can randomly start translating all `po` files in `manual/po/`. 80
- Don't forget you need `make commit` to ensure the translated manuals are updated from the `po` files, before `git commit -a` and `git push`. 81

2. About the Debian Live Project 82

2.1 Motivation 83

2.1.1 What is wrong with current live systems 84

When Debian Live was initiated, there were already several Debian based live systems available and they are doing a great job. From the Debian perspective most of them have one or more of the following disadvantages: 85

- They are not Debian projects and therefore lack support from within Debian. 86
- They mix different distributions, e.g. **testing** and **unstable**. 87
- They support i386 only. 88
- They modify the behaviour and/or appearance of packages by stripping them down to save space. 89
- They include packages from outside of the Debian archive. 90

- They ship custom kernels with additional patches that are not part of Debian. 80
- They are large and slow due to their sheer size and thus not suitable for rescue issues. 81
- They are not available in different flavours, e.g. CDs, DVDs, USB-stick and netboot images. 82

2.1.2 Why create our own live system? 83

Debian is the Universal Operating System: Debian has a live system to show around and to accurately represent the Debian system with the following main advantages: 84

- It would be a subproject of Debian. 85
- It reflects the (current) state of one distribution. 86
- It runs on as many architectures as possible. 87
- It consists of unchanged Debian packages only. 88
- It does not contain any packages that are not in the Debian archive. 89
- It uses an unaltered Debian kernel with no additional patches. 90

2.2 Philosophy 91

2.2.1 Only unchanged packages from Debian “main” 92

We will only use packages from the Debian repository in the “main” section. The non-free section is not part of Debian and therefore cannot be used for official live system images. 93

We will not change any packages. Whenever we need to change 93

something, we will do that in coordination with its package maintainer in Debian.

94 As an exception, our own packages such as *live-boot*, *live-build* or *live-config* may temporarily be used from our own repository for development reasons (e.g. to create development snapshots). They will be uploaded to Debian on a regular basis.

95 2.2.2 No package configuration of the live system

96 In this phase we will not ship or install sample or alternative configurations. All packages are used in their default configuration as they are after a regular installation of Debian.

97 Whenever we need a different default configuration, we will do that in coordination with its package maintainer in Debian.

98 A system for configuring packages is provided using debconf in `lb config` (use `--preseed FILE`) allowing custom configured packages to be installed in your custom produced Debian Live images, but for official live images only default configuration will be used. For more information, please see [<Customization overview>](#).

99 Exception: There are a few essential changes needed to bring a live system to life (e.g. configuring pam to allow empty passwords). These essential changes have to be kept as minimal as possible and should be merged within the Debian repository if possible.

100 2.3 Contact

101 • **Mailing list** : The primary contact for the project is the mailing list at [<http://lists.debian.org/debian-live/>](http://lists.debian.org/debian-live/). You can email the list directly by addressing your mail to [<debian-live@lists.debian.org.>](mailto:debian-live@lists.debian.org) The list archives are available at [<http://lists.debian.org/debian-live/>](http://lists.debian.org/debian-live/).

• **IRC** : A number of users and developers are present in the `#debian-live` channel on irc.debian.org (OFTC). When asking a question on IRC, please be patient for an answer. If no answer is forthcoming, please email the mailing list. 102

• **BTS** : The Debian Bug Tracking System (BTS) contains details of bugs reported by users and developers. Each bug is given a number, and is kept on file until it is marked as having been dealt with. For more information, please see [<Reporting bugs>](#). 103

• **Wiki** : The Debian Live wiki at <http://wiki.debian.org/DebianLive> is a place to gather information, discuss applied technologies, and document frameworks of Debian Live systems that go beyond the scope of this document. 104

User 105

3. Installation 106

3.1 Requirements 107

Building Debian Live images has very few system requirements: 108

- Super user (root) access 109
- An up-to-date version of *live-build* 110
- A POSIX-compliant shell, such as *bash* or *dash*. 111
- *debootstrap* or *cdebootstrap* 112
- Linux 2.6.x 113

Note that using Debian or a Debian-derived distribution is not required - *live-build* will run on almost any distribution with the above requirements. 114

115 3.2 Installing live-build

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

- 117 • From the Debian repository
- 118 • From source
- 119 • From snapshots

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

121 3.2.1 From the Debian repository

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

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

124 Or

```
125 # aptitude install live-build
```

126 3.2.2 From source

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

```
128 $ git clone git://live.debian.net/git/live-build.git
```

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

```
130 $ cd live-build
    $ dpkg-buildpackage -rfakeroot -b -uc -us
    $ cd ..
```

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

```
132 # dpkg -i live-build_2.0.8-1_all.deb
```

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

```
# make install
```

and uninstall it with:

```
# make uninstall
```

3.2.3 From `snapshots'

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

3.3 live-boot and live-config

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

3.3.1 From the Debian repository

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

3.3.2 From source

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

- 145 • Checkout the *live-boot* and *live-config* source

```
146 $ git clone git://live.debian.net/git/live-boot.git
146 $ git clone git://live.debian.net/git/live-config.git
```

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

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

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

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

```
151 $ cd live-boot
151 $ dpkg-buildpackage -b -uc -us
151 $ cd ../live-config
151 $ dpkg-buildpackage -b -uc -us
```

- 152 • Use all generated .deb files

153 As *live-boot* and *live-config* are installed by *live-build* system, installing the packages in the host system is not sufficient: you should treat the generated .deb files like any other custom packages. Please see [<Customizing package installation>](#) for more information. You should pay particular attention to [<Additional repositories>](#).

154

3.3.3 From 'snapshots'

You can let *live-build* automatically use the latest snapshots of *live-boot* and *live-config* by configuring a third-party repository in your *live-build* configuration directory. Assuming you have already created a configuration tree in the current directory with *lb config*:

```
$ lb config --archives live.debian.net
```

155

156

4. The basics

157

This chapter contains a brief overview of the build process and instructions for using the three most commonly used image types. The most versatile image type, *iso-hybrid*, may be used on a virtual machine, optical media or USB portable storage device. In certain special cases, such as the use of persistence, *usb-hdd* may be more suitable for USB devices. The chapter finishes with instructions for building and using a *net* type image, which is a bit more involved due to the setup required on the server. This is a slightly advanced topic for anyone who is not familiar already with netbooting, but is included here because once the setup is done, it is a very convenient way to test and deploy images for booting on the local network without the hassle of dealing with image media.

158

Throughout the chapter, we will often refer to the default filenames produced by *live-build*. If you are downloading a prebuilt image instead, the actual filenames may vary.

159

4.1 What is a live system?

160

A live system usually means an operating system booted on a computer from a removable medium, such as a CD-ROM or USB stick, or from a network, ready to use without any installation on

161

the usual drive(s), with auto-configuration done at run time (see <Terms>).

162 With Debian Live, it's a Debian GNU/Linux operating system, built
for one of the supported architectures (currently amd64, i386, pow-
erpc and sparc). It is made from the following parts:

- 163 • **Linux kernel image** , usually named `vmlinux*`
- 164 • **Initial RAM disk image (initrd)** : a RAM disk set up for the Linux
boot, containing modules possibly needed to mount the System
image and some scripts to do it.
- 165 • **System image** : The operating system's filesystem image. Usu-
ally, a SquashFS compressed filesystem is used to minimize the
Debian Live image size. Note that it is read-only. So, during boot
the Debian Live system will use a RAM disk and `union' mecha-
nism to enable writing files within the running system. However,
all modifications will be lost upon shutdown unless optional per-
sistence is used (see <Persistence>).
- 166 • **Bootloader** : A small piece of code crafted to boot from the cho-
sen media, possibly presenting a prompt or menu to allow se-
lection of options/configuration. It loads the Linux kernel and its
initrd to run with an associated system filesystem. Different solu-
tions can be used, depending on the target media and format of
the filesystem containing the previously mentioned components:
isolinux to boot from a CD or DVD in ISO9660 format, syslinux
for HDD or USB drive booting from a VFAT partition, extlinux for
ext2/3/4 and btrfs partitions, pxelinux for PXE netboot, GRUB for
ext2/3/4 partitions, etc.

167 You can use *live-build* to build the system image from your spec-
ifications, set up a Linux kernel, its initrd, and a bootloader to run
them, all in one media-dependant format (ISO9660 image, disk im-
age, etc.).

4.2 First steps: building an ISO hybrid image

168

Regardless of the image type, you will need to perform the same
basic steps to build an image each time. As a first example, ex-
ecute the following sequence of *live-build* commands to create a
basic ISO hybrid image containing just the Debian standard sys-
tem without X.org. It is suitable for burning to CD or DVD media,
and also to copy onto a USB stick.

169

First, run the `lb config` command. This will create a “config/” hi-
erarchy in the current directory for use by other commands:

170

```
$ lb config
```

171

No parameters are passed to `lb config`, so defaults for all of its
various options will be used. See <The `lb config` command> for more
details.

172

Now that the “config/” hierarchy exists, build the image with the `lb
build` command:

173

```
# lb build
```

174

This process can take a while, depending on the speed of your net-
work connection. When it is complete, there should be a binary-
`hybrid.iso` image file, ready to use, in the current directory.

175

4.3 Using an ISO hybrid live image

176

After either building or downloading an ISO hybrid image, which
can be obtained at <<http://www.debian.org/CD/live/>>, the usual next step
is to prepare your media for booting, either CD-R(W) or DVD-R(W)
optical media or a USB stick.

177

4.3.1 Burning an ISO image to a physical medium

178

Burning an ISO image is easy. Just install `wodim` and use it from

179

the command-line to burn the image. For instance:

```
180 # apt-get install wodim
    $ wodim binary-hybrid.iso
```

181 4.3.2 Copying an ISO hybrid image to a USB stick

182 ISO images prepared with the `isohybrid` command, like the images produced by the default `iso-hybrid` binary image type, can be simply copied to a USB stick with the `dd` program or an equivalent. Plug in a USB stick with a size large enough for your image file and determine which device it is, which we hereafter refer to as `/${USBSTICK}`. This is the device file of your key, such as `/dev/sdb`, not a partition, such as `/dev/sdb1`! You can find the right device name by looking in `dmesg`'s output after plugging in the stick, or better yet, `ls -l /dev/disk/by-id`.

183 Once you are certain you have the correct device name, use the `dd` command to copy the image to the stick. **This will definitely overwrite any previous contents on your stick!**

```
184 $ dd if=binary-hybrid.iso of=${USBSTICK}
```

185 4.3.3 Booting the live media

186 The first time you boot your live media, whether CD, DVD, USB key, or PXE boot, some setup in your computer's BIOS may be needed first. Since BIOSes vary greatly in features and key bindings, we cannot get into the topic in depth here. Some BIOSes provide a key to bring up a menu of boot devices at boot time, which is the easiest way if it is available on your system. Otherwise, you need to enter the BIOS configuration menu and change the boot order to place the boot device for the live system before your normal boot device.

Once you've booted the media, you are presented with a boot menu. If you just press enter here, the system will boot using the default entry, `Live` and default options. For more information about boot options, see the "help" entry in the menu and also the `live-boot` and `live-config` man pages found within the live system. 187

Assuming you've selected `Live` and booted a default desktop live image, after the boot messages scroll by, you should be automatically logged into the user account and see a desktop, ready to use. If you've booted a console-only image, such as `standard` or `rescue` flavour prebuilt images, you should be automatically logged in on the console to the user account and see a shell prompt, ready to use. 188

4.4 Using a virtual machine for testing 189

It can be a great time-saver for the development of live images to run them in a virtual machine (VM). This is not without its caveats: 190

- Running a VM requires enough RAM for both the guest OS and the host and a CPU with hardware support for virtualization is recommended. 191
- There are some inherent limitations to running on a VM, e.g. poor video performance, limited choice of emulated hardware. 192
- When developing for specific hardware, there is no substitute for running on the hardware itself. 193
- Occasionally there are bugs that relate only to running in a VM. When in doubt, test your image directly on the hardware. 194

Provided you can work within these constraints, survey the available VM software and choose one that is suitable for your needs. 195

196 4.4.1 Testing an ISO image with QEMU

197 The most versatile VM in Debian is QEMU. If your processor has hardware support for virtualization, use the `qemu-kvm` package; the `qemu-kvm` package description briefly lists the requirements.

198 First, install `qemu-kvm` if your processor supports it. If not, install `qemu`, in which case the program name is `qemu` instead of `kvm` in the following examples. The `qemu-utils` package is also valuable for creating virtual disk images with `qemu-img`.

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

200 Booting an ISO image is simple:

```
201 $ kvm -cdrom binary-hybrid.iso
```

202 See the man pages for more details.

203 4.4.2 Testing an ISO image with virtualbox-ose

204 In order to test the ISO with `virtualbox-ose`:

```
205 # apt-get install virtualbox-ose virtualbox-ose-dkms
```

```
206 $ virtualbox
```

206 Create a new virtual machine, change the storage settings to use `binary-hybrid.iso` as the CD/DVD device, and start the machine.

207 Note: For live systems containing X.org that you want to test with `virtualbox-ose`, you may wish to include the VirtualBox X.org driver package, `virtualbox-ose-guest-x11`, in your *live-build* configuration. Otherwise, the resolution is limited to 800x600.

```
208 $ echo virtualbox-ose-guest-x11 >>
config/package-lists/my.list.chroot
```

209 4.5 Building a USB/HDD image

Building a USB/HDD image is similar to ISO hybrid in all respects 210 except you specify `-b usb-hdd` and the resulting filename is `binary.img` which cannot be burnt to optical media. It is suitable for booting from USB sticks, USB hard drives, and various other portable storage devices. Normally, an ISO hybrid image can be used for this purpose instead, but if you have a BIOS which does not handle hybrid images properly, or want to use the remaining space on the media for some purpose, such as a persistence partition, you need a USB/HDD image.

Note: if you created an ISO hybrid image with the previous exam- 211 ple, you will need to clean up your working directory with the `lb clean` command (see [The lb clean command](#)):

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

Run the `lb config` command as before, except this time specifying 213 the USB/HDD image type:

```
214 $ lb config -b usb-hdd
```

Now build the image with the `lb build` command: 215

```
216 # lb build
```

When the build finishes, a `binary.img` file should be present in the 217 current directory.

218 4.6 Using a USB/HDD image

The generated binary image contains a VFAT partition and the 219 `syslinux` bootloader, ready to be directly written on a USB stick. Since using a USB/HDD image is just like using an ISO hybrid image on USB, follow the instructions in [Using an ISO hybrid live image](#), except use the filename `binary.img` instead of `binary-hybrid.iso`.

220 4.6.1 Testing a USB/HDD image with Qemu

221 First, install QEMU as described above in [<Testing an ISO image with QEMU>](#). Then run `kvm` or `qemu`, depending on which version your host system needs, specifying `binary.img` as the first hard drive.

```
222 $ kvm -hda binary.img
```

223 4.6.2 Using the space left on a USB stick

224 To use the remaining free space after copying `binary.img` to a USB stick, use a partitioning tool such as `gparted` or `parted` to create a new partition on the stick. The first partition will be used by the Debian Live system.

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

226 After the partition is created, where `${PARTITION}` is the name of the partition, such as `/dev/sdb2`, you have to create a filesystem on it. One possible choice would be `ext4`.

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

228 Note: If you want to use the extra space with Windows, apparently that OS cannot normally access any partitions but the first. Some solutions to this problem have been discussed on our [<mailing list>](#), but it seems there are no easy answers.

229 **Remember: Every time you install a new `binary.img` on the stick, all data on the stick will be lost because the partition table is overwritten by the contents of the image, so back up your extra partition first to restore again after updating the live image.**

230 4.7 Building a netboot image

231 The following sequence of commands will create a basic netboot

image containing the Debian standard system without X.org. It is suitable for booting over the network.

Note: if you performed any previous examples, you will need to clean up your working directory with the `lb clean` command:

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

Run the `lb config` command as follows to configure your image for netbooting:

```
$ lb config -b net --net-root-path "/srv/debian-live"
--net-root-server "192.168.0.1"
```

In contrast with the ISO and USB/HDD images, netbooting does not, itself, serve the filesystem image to the client, so the files must be served via NFS. The `--net-root-path` and `--net-root-server` options specify the location and server, respectively, of the NFS server where the filesystem image will be located at boot time. Make sure these are set to suitable values for your network and server.

Now build the image with the `lb build` command:

```
# lb build
```

In a network boot, the client runs a small piece of software which usually resides on the EPROM of the Ethernet card. This program sends a DHCP request to get an IP address and information about what to do next. Typically, the next step is getting a higher level bootloader via the TFTP protocol. That could be `pxelinux`, `GRUB`, or even boot directly to an operating system like Linux.

For example, if you unpack the generated `binary-net.tar.gz` archive in the `/srv/debian-live` directory, you'll find the filesystem image in `live/filesystem.squashfs` and the kernel, `initrd` and `pxelinux` bootloader in `tftpboot/debian-live/i386`.

We must now configure three services on the server to enable netboot: the DHCP server, the TFTP server and the NFS server.

242 4.7.1 DHCP server

243 We must configure our network's DHCP server to be sure to give an IP address to the netbooting client system, and to advertise the location of the PXE bootloader.

244 Here is an example for inspiration, written for the ISC DHCP server `isc-dhcp-server` in the `/etc/dhcp/dhcpd.conf` configuration file:

```
245 # /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;
        next-server servername;
        filename "pxelinux.0";
    }
```

246 4.7.2 TFTP server

247 This serves the kernel and initial ramdisk to the system at run time.

248 You should install the `tftpd-hpa` package. It can serve all files contained inside a root directory, usually `/srv/tftp`. To let it serve

files inside `/srv/debian-live/tftpboot`, run as root the following command:

```
# dpkg-reconfigure -plow tftpd-hpa 249
```

and fill in the new tftp server directory when being asked about it. 250

4.7.3 NFS server 251

252 Once the guest computer has downloaded and booted a Linux kernel and loaded its `initrd`, it will try to mount the Live filesystem image through a NFS server.

You need to install the `nfs-kernel-server` package. 253

254 Then, make the filesystem image available through NFS by adding a line like the following to `/etc/exports`:

```
/srv/debian-live 255
*(ro,async,no_root_squash,no_subtree_check)
```

256 and tell the NFS server about this new export with the following command:

```
# exportfs -rv 257
```

258 Setting up these three services can be a little tricky. You might need some patience to get all of them working together. For more information, see the `syslinux` wiki at <http://syslinux.zytor.com/wiki/index.php/PXELINUX> or the Debian Installer Manual's TFTP Net Booting section at <http://d-i.alioth.debian.org/manual/en.i386/ch04s05.html>. They might help, as their processes are very similar.

4.7.4 Netboot testing HowTo 259

260 Netboot image creation is made easy with *live-build* magic, but testing the images on physical machines can be really time consuming.

261 To make our life easier, we can use virtualization. There are two solutions.

262 4.7.5 Qemu

- 263 • Install `qemu`, `bridge-utils`, `sudo`.

264 Edit `/etc/qemu-ifup`:

```
265 #!/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
```

266 Get, or build a `grub-floppy-netboot` (in the svn).

267 Launch `qemu` with `"-net nic,vlan=0 -net tap,vlan=0,ifname=tun0"`

268 4.7.6 VMWare Player

- 269 • Install VMWare Player ("free as in beer" edition)
- 270 • Create a `PXETester` directory, and create a text file called `pxe.vwx` inside
- 271 • Paste this text inside:

```
272 #!/usr/bin/vmware
config.version = "8"
virtualHW.version = "4"
memsize = "512"
MemAllowAutoScaleDown = "FALSE"

ide0:0.present = "FALSE"
```

```
ide1:0.present = "FALSE"
floppy0.present = "FALSE"
sound.present = "FALSE"
tools.remindInstall = "FALSE"
```

```
ethernet0.present = "TRUE"
ethernet0.addressType = "generated"
```

```
displayName = "Test Boot PXE"
guestOS = "other"
```

```
ethernet0.generatedAddress = "00:0c:29:8d:71:3b"
uuid.location = "56 4d 83 72 5c c4 de 3f-ae 9e 07 91 1d 8d
71 3b"
uuid.bios = "56 4d 83 72 5c c4 de 3f-ae 9e 07 91 1d 8d 71
3b"
ethernet0.generatedAddressOffset = "0"
```

- 273 • You can play with this configuration file (e.g. change memory limit to 256)
- 274 • Double click on this file (or run VMWare player and select this file).
- 275 • When running just press space if that strange question comes up...

276 5. Overview of tools

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

278 5.1 live-build

279 *live-build* is a collection of scripts to build Debian Live systems.

These scripts are also referred to as “commands”.

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

281 Many concepts are similar to those in the debhelper Debian package tools written by Joey Hess:

282 • The scripts have a central location for configuring their operation. In debhelper, this is the `debian/` subdirectory of a package tree. For example, `dh_install` will look, amongst others, for a file called `debian/install` to determine which files should exist in a particular binary package. In much the same way, *live-build* stores its configuration entirely under a `config/` subdirectory.

283 • The scripts are independent - that is to say, it is always safe to run each command.

284 Unlike debhelper, *live-build* contains a tool to generate a skeleton configuration directory, `lb config`. This could be considered to be similar to tools such as `dh-make`. For more information about `lb config`, please see [<The lb config command>](#).

285 The remainder of this section discusses the three most important commands:

286 • **lb config** : Responsible for initializing a Live system configuration directory. See [<The lb config command>](#) for more information.

287 • **lb build** : Responsible for starting a Live system build. See [<The lb build command>](#) for more information.

288 • **lb clean** : Responsible for removing parts of a Live system build. See [<The lb clean command>](#) for more information.

289

5.1.1 The `lb config` command

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

Issuing `lb config` without any arguments creates a `config/` subdirectory which it populates with some default settings, and a skeleton `auto/` subdirectory tree.

```
$ lb config
P: Considering defaults defined in
/etc/live/build.conf
P: Creating config tree
```

Using `lb config` without any arguments would be suitable for users who need a very basic image, or who intend to later provide a more complete configuration via `auto/config` (see [<Managing a configuration>](#) for details).

Normally, you will want to specify some options. For example, to include the ``gnome'` package list in your configuration:

```
$ lb config -p gnome
```

It is possible to specify many options, such as:

```
$ lb config --binary-images net --hostname live-machine
--username live-user ...
```

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

5.1.2 The `lb build` command

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

301 5.1.3 The `lb clean` command

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

303 5.2 The `live-boot` package

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

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

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

307 5.3 The `live-config` package

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

309 6. Managing a configuration

This chapter explains how to manage a live configuration from initial 310
 creation, through successive revisions and successive releases of
 both the *live-build* software and the live image itself.

6.1 Use `auto` to manage configuration changes 311

Live configurations rarely are perfect on the first try. You'll likely 312
 need to make a series of revisions until you are satisfied. How-
 ever, inconsistencies can creep into your configuration from one
 revision to the next if you aren't careful. The main problem is, once
 a variable is given a default value, that value will not be recomputed
 from other variables that may change in later revisions.

For example, when the distribution is first set, many 'dependent' 313
 variables are given default values that suit that distribution. How-
 ever, if you later decide to change the distribution, those dependent
 variables continue to retain old values that are no longer appropri-
 ate.

A second, related problem is that if you run `lb config` and then 314
 upgrade to a new version of *live-build* that has changed one of the
 variable names, you will discover this only by manual review of the
 variables in your `config/*` files, which you will then need to use to
 set the appropriate option again.

All of this would be a terrible nuisance if it weren't for `auto/*` scripts, 315
 simple wrappers to the `lb config`, `lb build` and `lb clean` com-
 mands that are designed to help you manage your configuration.
 Simply create an `auto/config` script containing `lb config` com-
 mand with all desired options, and an `auto/clean` that removes the
 files containing configuration variable values, and each time you `lb
 config` and `lb clean`, these files will be executed. This will ensure

that your configuration is kept internally consistent from one revision to the next and from one *live-build* release to the next (though you will still have to take care and read the documentation when you upgrade *live-build* and make adjustments as needed).

6.2 Example auto scripts

Use auto script examples such as the following as the starting point for your new *live-build* configuration. Take note that when you call the `lb` command that the auto script wraps, you must specify `noauto` as its parameter to ensure that the auto script isn't called again, recursively. Also, don't forget to ensure the scripts are executable (e.g. `chmod 755 auto/*`).

```
auto/config
```

```
#!/bin/sh
lb config noauto \
  --package-lists "standard" \
  "${@}"
```

```
auto/clean
```

```
#!/bin/sh
lb clean noauto "${@}"
rm -f config/binary config/bootstrap \
  config/chroot config/common config/source
rm -f binary.log
```

```
auto/build
```

```
#!/bin/sh
lb build noauto "${@}" 2>&1 |tee binary.log
```

We now ship example auto scripts with *live-build* based on the examples above. You may copy those as your starting point.

```
$ cp /usr/share/live/build/examples/auto/* auto/
```

Edit `auto/config`, changing or adding any options as you see fit. In the example above, `--package-lists standard` is set to the

default value. Change this to an appropriate value for your image (or delete it if you want to use the default) and add any additional options in continuation lines that follow.

7. Customization overview

This chapter gives an overview of the various ways in which you may customize a Debian Live system.

7.1 Build time vs. boot time configuration

Live system configuration options are divided into build-time options which are options that are applied at build time and boot-time options which are applied at boot time. Boot-time options are further divided into those occurring early in the boot, applied by the *live-boot* package, and those that happen later in the boot, applied by *live-config*. Any boot-time option may be modified by the user by specifying it at the boot prompt. The image may also be built with default boot parameters so users can normally just boot directly to the live system without specifying any options when all of the defaults are suitable. In particular, the argument to `lb --bootappend-live` consists of any default kernel command line options for the Live system, such as persistence, keyboard layouts, or timezone. See [Customizing locale and language](#), for example.

Build-time configuration options are described in the `lb config` man page. Boot-time options are described in the man pages for *live-boot* and *live-config*. Although the *live-boot* and *live-config* packages are installed within the live system you are building, it is recommended that you also install them on your build system for easy reference when you are working on your configuration. It is safe to do so, as none of the scripts contained within them are executed unless the system is configured as a live system.

332 7.2 Stages of the build

333 The build process is divided into stages, with various customizations applied in sequence in each. The first stage to run is the **bootstrap** stage. This is the initial phase of populating the chroot directory with packages to make a barebones Debian system. This is followed by the **chroot** stage, which completes the construction of chroot directory, populating it with all of the packages listed in the configuration, along with any other materials. Most customization of content occurs in this stage. The final stage of preparing the live image is the **binary** stage, which builds a bootable image, using the contents of the chroot directory to construct the root filesystem for the Live system, and including the installer and any other additional material on the target media outside of the Live system's filesystem. After the live image is built, if enabled, the source tarball is built in the **source** stage.

334 Within each of these stages, there is a particular sequence in which commands are applied. These are arranged in such a way as to ensure customizations can be layered in a reasonable fashion. For example, within the **chroot** stage, preseeds are applied before any packages are installed, packages are installed before any locally included files or patches are applied, and hooks are run later, after all of the materials are in place.

335 7.3 Supplement lb config with files

336 Although `lb config` does create a skeletal configuration in the `config/` directory, to accomplish your goals, you may need to provide additional files in subdirectories of `config/`. Depending on where the files are stored in the configuration, they may be copied into the live system's filesystem or into the binary image filesystem, or may provide build-time configurations of the system that would be cumbersome to pass as command-line options. You may include things

such as custom lists of packages, custom artwork, or hook scripts to run either at build time or at boot time, boosting the already considerable flexibility of `debian-live` with code of your own.

7.4 Customization tasks 337

338 The following chapters are organized by the kinds of customization task users typically perform: [<Customizing package installation>](#), [<Customizing contents>](#) and [<Customizing locale and language>](#) cover just a few of the things you might want to do.

8. Customizing package installation 339

340 Perhaps the most basic customization of a Debian live system is the selection of packages to be included in the image. This chapter guides you through the various build-time options to customize `live-build`'s installation of packages. The broadest choices influencing which packages are available to install in the image are the distribution and archive areas. To ensure decent download speeds, you should choose a nearby distribution mirror. You can also add your own repositories for backports, experimental or custom packages, or include packages directly as files. You can define your own lists of packages to include, use `live-build`'s predefined lists, use `taskset` tasks, or a combination of all three. Finally, a number of options give some control over `apt`, or if you prefer, `aptitude`, at build time when packages are installed. You may find these handy if you use a proxy, want to disable installation of recommended packages to save space, or need to control which versions of packages are installed via APT pinning, to name a few possibilities.

8.1 Package sources

8.1.1 Distribution, archive areas and mode

343 The distribution you choose has the broadest impact on which packages are available to include in your live image. Specify the codename, which defaults to wheezy for the **Wheezy** version of *live-build*. Any current distribution carried in the Debian archive may be specified by its codename here. (See <Terms> for more details.) The `--distribution` option not only influences the source of packages within the archive, but also instructs `live-build` to behave as needed to build each supported distribution. For example, to build against the `*unstable*` release, **Sid**, specify:

```
344 $ lb config --distribution sid
```

345 Within the distribution archive, archive areas are major divisions of the archive. In Debian, these are `main`, `contrib` and `non-free`. Only `main` contains software that is part of the Debian distribution, hence that is the default. One or more values may be specified, e.g.

```
346 $ lb config --archive-areas "main contrib"
```

347 Experimental support is available for some Debian derivatives through a `--mode` option. By default, this option is set to `debian`, even if you are building on a non-Debian system. If you specify `--mode ubuntu` or `--mode emdebian`, the distribution names and archive areas for the specified derivative are supported instead of the ones for Debian. The mode also modifies *live-build* behaviour to suit the derivatives.

348 **Note:** The projects for whom these modes were added are primarily responsible for supporting users of these options. The Debian live project, in turn, provides development support on a best-effort basis only, based on feedback from the derivative projects as we do not develop or support these derivatives ourselves.

341 8.1.2 Distribution mirrors

342 The Debian archive is replicated across a large network of mirrors around the world so that people in each region can choose a nearby mirror for best download speed. Each of the `--parent-mirror-*` options governs which distribution mirror is used at various stages of the build. Recall from <Stages of the build> that the `*bootstrap*` stage is when the chroot is initially populated by `debootstrap` with a minimal system, and the `*chroot*` stage is when the chroot used to construct the live system's filesystem is built. Thus, the corresponding mirror switches are used for those stages, and later, in the `*binary*` stage, the `--parent-mirror-binary` and `--parent-mirror-binary-security` values are used, superceding any mirrors used in an earlier stage.

8.1.3 Distribution mirrors used at build time

To set the distribution mirrors used at build time to point at a local mirror, it is sufficient to set `--parent-mirror-bootstrap`, `--parent-mirror-chroot-security` and `--parent-mirror-chroot-backports` as follows.

```
353 $ lb config --parent-mirror-bootstrap http://localhost/debian/ \
\
--parent-mirror-chroot-security
http://localhost/debian-security/ \
--parent-mirror-chroot-backports
http://localhost/debian-backports/
```

The chroot mirror, specified by `--parent-mirror-chroot`, defaults to the `--parent-mirror-bootstrap` value.

8.1.4 Distribution mirrors used at run time

The `--parent-mirror-binary*` options govern the distribution mir-

rors placed in the binary image. These may be used to install additional packages while running the live system. The defaults employ `cdn.debian.net`, a service that chooses a geographically close mirror based on the user's IP number. This is a suitable choice when you cannot predict which mirror will be best for all of your users. Or you may specify your own values as shown in the example below. An image built from this configuration would only be suitable for users on a network where "mirror" is reachable.

```
357 $ lb config --parent-mirror-binary http://mirror/debian/ \
    --parent-mirror-binary-security
    http://mirror/debian-security/
```

358 8.1.5 Additional repositories

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

360 For example, `config/archives/live.list.chroot` allows you to install packages from the debian live snapshot repository at live system build time.

```
361 deb http://live.debian.net/ sid-snapshots main contrib
    non-free
```

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

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

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

Note: some preconfigured package repositories are available for easy selection through the `--archives` option, e.g. for enabling live snapshots, a simple command is enough to enable it:

```
$ lb config --archives live.debian.net
```

367 8.2 Choosing packages to install

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

369 8.2.1 Package lists

370 Package lists are a powerful way of expressing which packages should be installed. The list syntax supports included files and conditional sections which makes it easy to build lists from other lists and adapt them for use in multiple configurations. You can use predefined package lists, providing in a modular fashion package selections from each of the major desktop environments and some special purpose lists, as well as standard lists the others are based upon. You can also provide your own package lists, or use a combination of both.

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

372 8.2.2 Predefined package lists

373 The simplest way to use lists is to specify one or more predefined
lists with the `--package-lists` option. For example:

```
374 $ lb config --package-lists "gnome rescue"
```

375 The default location for the list files on your system is `/usr/share/-
live/build/package-lists/`. To determine the packages in a
given list, read the corresponding file, paying attention to included
files and conditionals as described in the following sections.

376 8.2.3 Local package lists

377 You may supplement the predefined lists using local package lists
stored in `config/package-lists/`.

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

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

380 8.2.4 Local binary package lists

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

382 8.2.5 Extending a provided package list using includes

The package lists that are included with *live-build* make extensive
use of includes. Refer to these in the `/usr/share/live/build/-
package-lists/` directory, as they serve as good examples of how
to write your own lists.

For example, to make a list that includes the predefined `gnome` list
plus `iceweasel`, create `config/package-lists/my.list.chroot`
with the following contents:

```
#include <gnome> 385  
iceweasel
```

386 8.2.6 Using conditionals inside package lists

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

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

```
#if ARCHITECTURE amd64 389  
ia32-libs  
#endif
```

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

```
#if ARCHITECTURE i386 amd64 391  
memtest86+  
#endif
```

You may also test against variables that may contain more than one 392

value, e.g. to install `vrms` if either `contrib` or `non-free` is specified via `--archive-areas`:

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

394 A conditional may surround an `#include` directive:

```
395     #if ARCHITECTURE amd64
        #include <gnome-full>
    #endif
```

396 The nesting of conditionals is not supported.

397 8.2.7 Tasks

398 The Debian Installer offers the user choices of a number of pre-selected lists of packages, each one focused on a particular kind of system, or task a system may be used for, such as “Graphical desktop environment”, “Mail server” or “Laptop”. These lists are called “tasks” and are supported by APT through the “Task:” field. You can specify one or more tasks in *live-build* by putting them in a list in `config/task-lists/`, as in the example below.

```
399     $ lb config
        $ echo "mail-server file-server" >>
config/task-lists/my.list.chroot
```

400 The primary tasks available in the Debian Installer can be listed with `tasksel --list-tasks` in the live system. The contents of any task, including ones not included in this list, may be examined with `tasksel --task-packages`.

401 8.2.8 Desktop and language tasks

402 Desktop and language tasks are special cases that need some ex-

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

When developing a desktop live image, the image typically boots 403 directly to a working desktop, the choices of both desktop and default language having been made at build time, not at run time as in the case of the Debian Installer. That's not to say that a live image couldn't be built to support multiple desktops or multiple languages and offer the user a choice, but that is not *live-build*'s default behaviour.

Because there is no provision made automatically for language 404 tasks, which include such things as language-specific fonts and input-method packages, if you want them, you need to specify them in your configuration. For example, a GNOME desktop image containing support for Japanese might include these tasks:

```
405     $ lb config
        $ echo "gnome-desktop desktop standard laptop" >>
config/task-lists/my.list.chroot
        $ echo "japanese japanese-desktop japanese-gnome-desktop"
>> config/task-lists/my.list.chroot
```

Since desktop tasks are “internal” tasks, for every desktop flavour 406 task included in the image, the corresponding value, if it differs from the default, “gnome”, must be preseeded in the “`tasksel/desktop`” `debconf` variable or else `tasksel` will not recognize and install it. Thus:

```
407     $ lb config
```

```
$ echo 'tasksel tasksel/desktop multiselect kde' >>
config/preseed/my.preseed.chroot
```

408 This parameter can take multiple values, e.g. “lxde xfce” instead of “kde”.

409 8.3 Installing modified or third-party packages

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

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

412 There are two ways of installing modified custom packages:

- 413 • `packages.chroot`
- 414 • Using a custom APT repository

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

416 8.3.1 Using `packages.chroot` to install custom packages

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

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

Packages **must** be named in the prescribed way. One simple way to do this is to use `dpkg-name`. 418

Using `packages.chroot` for installation of custom packages has disadvantages: 419

- It is not possible to use secure APT. 420
- You must install all appropriate packages in the `config/-packages.chroot/` directory. 421
- It does not lend itself to storing Debian Live configurations in revision control. 422

423 8.3.2 Using an APT repository to install custom packages

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

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

426 8.3.3 Custom packages and APT

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

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

429 8.4 Configuring APT at build time

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

431 8.4.1 Choosing apt or aptitude

432 You can elect to use either `apt` or `aptitude` when installing packages at build time. Which utility is used is governed by the `--apt` argument to `lb config`. Choose the method implementing the preferred behaviour for package installation, the notable difference being how missing packages are handled.

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

435 8.4.2 Using a proxy with APT

436 One commonly required APT configuration is to deal with building an image behind a proxy. You may specify your APT proxy

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

```
437 $ lb config --apt-http-proxy http://proxy/
```

438 8.4.3 Tweaking APT to save space

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

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

```
441 $ lb config --apt-indices false
```

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

443 If you find the installation of recommended packages bloats your image too much, you may disable that default option of APT with:

```
444 $ lb config --apt-recommends false
```

445 The tradeoff here is that if you don't install recommended packages for a given package, that is, "packages that would be found together with this one in all but unusual installations" (Debian Policy Manual, section 7.2), some packages that you actually need may be omitted. Therefore, we suggest you review the difference turning off `recommends` makes to your packages list (see the `binary.packages` file generated by `lb build`) and re-include in your list any missing packages that you still want installed. Alternatively,

if you find you only want a small number of recommended packages left out, leave recommends enabled and set a negative APT pin priority on selected packages to prevent them from being installed, as explained in [APT pinning](#).

446 8.4.4 Passing options to apt or aptitude

447 If there is not an `lb config` option to alter APT's behaviour in the way you need, use `--apt-options` or `--aptitude-options` to pass any options through to your configured APT tool. See the man pages for `apt` and `aptitude` for details.

448 8.4.5 APT pinning

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

450 Let's say you are building a **Wheezy** live system but need all the live packages that end up in the binary image to be installed from **Sid** at build time. You need to add **Sid** to your APT sources and pin it so that only the packages you want are installed from it at build time and all others are taken from the target system distribution, **Wheezy**. The following will accomplish this:

```
451 $ echo "deb http://mirror/debian sid main" >
config/archives/sid.list.chroot
$ cat >> config/chroot_apt/preferences <<END
Package: live-boot live-boot-initramfs-tools live-config
live-config-sysvinit
Pin: release n=sid
Pin-Priority: 600
```

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

Note: Wildcards can be used in package names (e.g. **Package: live-***) with Apt version 0.8.14 or higher. This means that it works with **Wheezy** using: 452

```
$ lb config --distribution wheezy 453
```

Negative pin priorities will prevent a package from being installed, as in the case where you do not want a package that is recommended by another package. Suppose you are building an LXDE image using `--package-lists lxde` option, but don't want the user prompted to store wifi passwords in the keyring. This list includes `gdm`, which depends on `gksu`, which in turn recommends `gnome-keyring`. So you want to omit the recommended `gnome-keyring` package. This can be done by adding the following stanza to `config/chroot_apt/preferences`: 454

```
Package: gnome-keyring 455
Pin: version *
Pin-Priority: -1
```

9. Customizing contents 456

This chapter discusses fine-tuning customization of the live system contents beyond merely choosing which packages to include. Includes allow you to add or replace arbitrary files in your Debian Live image, hooks allow you to execute arbitrary commands at different stages of the build and at boot time, and preseeding allows you to configure packages when they are installed by supplying answers to `debconf` questions. 457

9.1 Includes

459 While ideally a Debian live system would include files entirely provided by unmodified Debian packages, it is sometimes convenient to provide or modify some content by means of files. Using includes, it is possible to add (or replace) arbitrary files in your Debian Live image. *live-build* provides three mechanisms for using them:

- 460 • Chroot local includes: These allow you to add or replace files to the chroot/Live filesystem. Please see [<Live/chroot local includes>](#) for more information.
- 461 • Binary local includes: These allow you to add or replace files in the binary image. Please see [<Binary local includes>](#) for more information.
- 462 • Binary includes: These allow you to add or replace Debian specific files in the binary image, such as the templates and tools directories. Please see [<Binary includes>](#) for more information.

463 Please see [<Terms>](#) for more information about the distinction between the “Live” and “binary” images.

9.1.1 Live/chroot local includes

465 Chroot local includes can be used to add or replace files in the chroot/Live filesystem so that they may be used in the Live system. A typical use is to populate the skeleton user directory (`/etc/skel`) used by the Live system to create the live user's home directory. Another is to supply configuration files that can be simply added or replaced in the image without processing; see [<Live/chroot local hooks>](#) if processing is needed.

466 To include files, simply add them to your `config/includes.chroot` directory. This directory corresponds to the root directory (`/`) of the

458 live system. For example, to add a file `/var/www/index.html` in the live system, use:

```
$ mkdir -p config/includes.chroot/var/www 467
$ cp /path/to/my/index.html
config/includes.chroot/var/www
```

Your configuration will then have the following layout: 468

```
-- config 469
[...
|-- includes.chroot
|   |-- var
|       |-- www
|           |-- index.html
[...
|-- templates
```

Chroot local includes are installed after package installation so that files installed by packages are overwritten. 470

9.1.2 Binary local includes

472 To include material such as documentation or videos on the media filesystem so that it is accessible immediately upon insertion of the media without booting the Live system, you can use binary local includes. This works in a similar fashion to chroot local includes. For example, suppose the files `~/video_demo.*` are demo videos of the live system described by and linked to by an HTML index page. Simply copy the material to `config/includes.binary/` as follows:

```
$ cp ~/video_demo.* config/includes.binary/ 473
```

474 These files will now appear in the root directory of the live media.

9.1.3 Binary includes

476 *live-build* has some standard files (like documentation) that gets
included in the default configuration on every live media. This can
be disabled with:

```
477 $ lb config --includes none
```

478 Otherwise, the material will be installed by *live-build* in `/includes/`
by default on the media filesystem, or else you can specify an al-
ternate path with `--includes`.

9.2 Hooks

480 Hooks allow commands to be performed in the chroot and binary
stages of the build in order to customize the image.

9.2.1 Live/chroot local hooks

482 To run commands in the chroot stage, create a hook script with a
`.chroot` suffix containing the commands in the `config/hooks/` di-
rectory. The hook will run in the chroot after the rest of your chroot
configuration has been applied, so remember to ensure your con-
figuration includes all packages and files your hook needs in order
to run. See the example chroot hook scripts for various common
chroot customization tasks provided in `/usr/share/live/build/-`
`examples/hooks` which you can copy or symlink to use them in your
own configuration.

9.2.2 Boot-time hooks

484 To execute commands at boot time, you can supply *live-config*
hooks as explained in the “Customization” section of its man page.
Examine *live-config*'s own hooks provided in `/lib/live/config/`,

475 noting the sequence numbers. Then provide your own hook pre-
fixed with an appropriate sequence number, either as a chroot lo-
cal include in `config/includes.chroot/lib/live/config/`, or as
a custom package as discussed in [Installing modified or third-party
packages](#).

9.2.3 Binary local hooks

485 To run commands in the binary stage, create a hook script with a
486 `.binary` suffix containing the commands in the `config/hooks/` di-
rectory. The hook will run after all other binary commands are run,
but before `binary_checksums`, the very last binary command. The
commands in your hook do not run in the chroot, so take care to
not modify any files outside of the build tree, or you may damage
your build system! See the example binary hook scripts for vari-
ous common binary customization tasks provided in `/usr/share/-`
`live/build/examples/hooks` which you can copy or symlink to use
them in your own configuration.

9.3 Preseeding Debconf questions

487 Files in the `config/preseed/` directory suffixed with `.preseed` fol-
488 lowed by the stage (`.chroot` or `.binary`) are considered to be de-
bconf preseed files and are installed by *live-build* using `debconf-`
`set-selections` during the corresponding stage.

489 For more information about debconf, please see `debconf(7)` in the
debconf package.

10. Customizing run time behaviours

490 All configuration that is done during run time is done by *live-config*.
491 Here are some of the most common options of *live-config* that users

are interested in. A full list of all possibilities can be found in the manpage of *live-config*.

492 10.1 Customizing the live user

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

494 You can specify additional groups that the live user will belong to by preseeding the `passwd/user-default-groups` debconf value. For example, to add the live user to the `fuse` group, add the following preseed under `config/preseed/` for the chroot stage:

```
495 $ lb config
    $ echo user-setup passwd/user-default-groups string audio
    cdrom \
        dip floppy video plugdev netdev powerdev scanner bluetooth
    fuse \
        >> config/preseed/my.preseed.chroot
```

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

497 To change the default username you can simply specify it in your config:

```
498 $ lb config --bootappend-live "username=live-user"
```

499 One possible way of changing the default password is by means of a hook as described in [«Boot-time hooks»](#). In order to do that you can use the “passwd” hook from `/usr/share/doc/live-config/-examples/hooks`, prefix it accordingly (e.g. `200-passwd`) and add it to `config/includes.chroot/lib/live/config/`

10.2 Customizing locale and language

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

- the locale generation 502
- setting the keyboard layout for the console 503
- setting the keyboard layout for X 504

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

```
$ lb config --bootappend-live "locales=de_CH.UTF-8" 506
```

This parameter can also be used at the kernel command line. You can specify a locale by a full `language_country.encoding` word.

Both the console and X keyboard configuration depend on the `keyboard-layouts` parameter of the `--bootappend-live` option. Valid options for X keyboard layouts can be found in `/usr/share/X11/xkb/rules/base.xml` (rather limited to two-letters country codes). To find the value (the two characters) corresponding to a language try searching for the english name of the nation where the language is spoken, e.g:

```
$ grep -i sweden -C3 /usr/share/X11/xkb/rules/base.xml 509
|grep name
<name>se</name>
```

To get the locale files for German and Swiss German keyboard layout in X use:

```
$ lb config --bootappend-live "locales=de_CH.UTF-8 511
keyboard-layouts=ch"
```

A list of the valid values of the keyboards for the console can be 512

figured with the following command:

```
513 $ for i in $(find /usr/share/keymaps/ -iname "*kmap.gz");
    \
        do basename $i |head -c -9; echo; done |sort |less
```

514 Alternatively, you can use the `console-setup` package, a tool to let you configure console layout using X (XKB) definitions; you can then set your keyboard layout more precisely with `keyboard-layouts`, `keyboard-variant`, `keyboard-options` and `keyboard-model` variables; *live-boot* will use also these parameters for X configuration. For example, to set up a French system with a French-Dvorak layout (called Bepo) on a TypeMatrix keyboard, both in console and X11, use:

```
515 $ lb config --bootappend-live \
    "locales=fr_FR.UTF-8 keyboard-layouts=fr
    keyboard-variant=bepo keyboard-model=tm2030usb"
```

516 10.3 Persistence

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

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

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

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

521 10.3.1 Full persistence

522 By 'full persistence' it is meant that instead of using a tmpfs for storing modifications to the read-only media (with the copy-on-write, COW, system) a writable partition is used. In order to use this feature a partition with a clean writable supported filesystem on it labeled "live-rw" must be attached on the system at boot time and the system must be started with the boot parameter 'persistent'. This partition could be an ext2 partition on the hard disk or on a usb key created with, e.g.:

```
# mkfs.ext2 -L live-rw /dev/sdb1
```

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

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

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

527 But since live system users cannot always use a hard drive partition, and considering that most USB keys have poor write speeds, 'full' persistence could be also used with just image files, so you could create a file representing a partition and put this image file even on a NTFS partition of a foreign OS, with something like:

```
$ dd if=/dev/null of=live-rw bs=1G seek=1 # for a 1GB sized
image file
$ /sbin/mkfs.ext2 -F live-rw
```

529 Then copy the `live-rw` file to a writable partition and reboot with
the boot parameter ``persistent'`.

530 **10.3.2 Home automounting**

531 If during the boot a partition (filesystem) image file or a partition
labeled `home-rw` is discovered, this filesystem will be directly
mounted as `/home`, thus permitting persistence of files that
belong to e.g. the default user. It can be combined with full
persistence.

532 **10.3.3 Snapshots**

533 Snapshots are collections of files and directories which are not
mounted while running but which are copied from a persistent de-
vice to the system (tmpfs) at boot and which are resynced at re-
boot/shutdown of the system. The content of a snapshot could re-
side on a partition or an image file (like the above mentioned types)
labeled `live-sn`, but it defaults to a simple cpio archive named
`live-sn.cpio.gz`. As above, at boot time, the block devices con-
nected to the system are traversed to see if a partition or a file
named like that could be found. A power interruption during run
time could lead to data loss, hence a tool invoked `live-snapshot`
`--refresh` could be called to sync important changes. This type of
persistence, since it does not write continuously to the persistent
media, is the most flash-based device friendly and the fastest of all
the persistence systems.

534 A `/home` version of snapshot exists too and its label is `home-sn.*`;
it works the same as the main snapshot but it is only applied to
`/home`.

535 Snapshots cannot currently handle file deletion but full persistence
and home automounting can.

10.3.4 Persistent SubText

If a user would need multiple persistent storage of the same type
for different locations or testing, such as `live-rw-nonwork` and
`live-rw-work`, the boot parameter `persistent-subtext` used in
conjunction with the boot parameter `persistent` will allow for
multiple but unique persistent media. An example would be if
a user wanted to use a persistent partition labeled `live-sn-`
`subText` they would use the boot parameters of: `persistent`
`persistent-subtext=subText`.

10.3.5 Partial remastering

The run-time modification of the tmpfs could be collected using `live-`
`snapshot` in a squashfs and added to the cd by remastering the iso
in the case of `cd-r` or adding a session to multisession `cd/dvd(rw)`;
`live-boot` mounts all `/live` filesystem in order or with the module boot
parameter.

11. Customizing the binary image

11.1 Bootloader

`live-build` uses `syslinux` as bootloader by default, which is by de-
fault configured to pause indefinitely at its splash screen. To adjust
this, you can pass `--syslinux-timeout TIMEOUT` to `lb config`.
The value is specified in units of seconds. A timeout of 0 (zero)
disables the timeout completely. For more information please see
`syslinux(1)`.

11.2 ISO metadata

When creating an ISO9660 binary image, you can use the following

options to add various textual metadata for your image. This can help you easily identify the version or configuration of an image without booting it.

- 545 • `LB_ISO_APPLICATION/--iso-application` NAME: This should describe the application that will be on the image. The maximum length for this field is 128 characters.
- 546 • `LB_ISO_PREPARER/--iso-preparer` NAME: This should describe the preparer of the image, usually with some contact details. The default for this option is the *live-build* version you are using, which may help with debugging later. The maximum length for this field is 128 characters.
- 547 • `LB_ISO_PUBLISHER/--iso-publisher` NAME: This should describe the publisher of the image, usually with some contact details. The maximum length for this field is 128 characters.
- 548 • `LB_ISO_VOLUME/--iso-volume` NAME: This should specify the volume ID of the image. This is used as a user-visible label on some platforms such as Windows and Apple Mac OS. The maximum length for this field is 32 characters.

549 12. Customizing Debian Installer

550 Debian Live system images can be integrated with Debian Installer. There are a number of different types of installation, varying in what is included and how the installer operates.

551 Please note the careful use of capital letters when referring to the “Debian Installer” in this section - when used like this we refer explicitly to the official installer for the Debian system, not anything else. It is often seen abbreviated to “d-i”.

12.1 Types of Debian Installer

The three main types of installer are:

553
554 **“Regular” Debian Installer** : This is a normal Debian Live image with a separate kernel and `initrd` which (when selected from the appropriate bootloader) launches into a standard Debian Installer instance, just as if you had downloaded a CD image of Debian and booted it. Images containing a live system and such an otherwise independent installer are often referred to as “combined images”.

555 On such images, Debian is installed by fetching and installing `.deb` packages using `debootstrap` or `cdebootstrap`, from the local media or some network-based network, resulting in a standard Debian system being installed to the hard disk.

556 This whole process can be preseeded and customized in a number of ways; see the relevant pages in the Debian Installer manual for more information. Once you have a working preseed file, *live-build* can automatically put it in the image and enable it for you.

557 **“Live” Debian Installer** : This is a Debian Live image with a separate kernel and `initrd` which (when selected from the appropriate bootloader) launches into an instance of the Debian Installer.

558 Installation will proceed in an identical fashion to the “Regular” installation described above, but at the actual package installation stage, instead of using `debootstrap` to fetch and install packages, the live filesystem image is copied to the target. This is achieved with a special `udeb` called *live-installer*.

559 After this stage, the Debian Installer continues as normal, installing and configuring items such as bootloaders and local users, etc.

Note: to support both normal and live installer entries in the boot-

loader of the same live media, you must disable *live-installer* by preseeding `live-installer/enable=false`.

561 **“Desktop” Debian Installer** : Regardless of the type of Debian Installer included, `d-i` can be launched from the Desktop by clicking on an icon. This is user friendlier in some situations. In order to make use of this, the *debian-installer-launcher* package needs to be included.

562 Note that by default, *live-build* does not include Debian Installer images in the images, it needs to be specifically enabled with `lb config`. Also, please note that for the “Desktop” installer to work, the kernel of the live system must match the kernel `d-i` uses for the specified architecture. For example:

```
563 $ lb config --architecture i386 --linux-flavours 486 \
    --debian-installer live
    $ echo debian-installer-launcher >>
    config/package-lists/my.list.chroot
```

564 12.2 Customizing Debian Installer by preseeding

565 As described in the Debian Installer Manual, Appendix B at <http://www.debian.org/releases/stable/i386/apb.html>, “Preseeding provides a way to set answers to questions asked during the installation process, without having to manually enter the answers while the installation is running. This makes it possible to fully automate most types of installation and even offers some features not available during normal installations.” This kind of customization is best accomplished with *live-build* by placing the configuration in a `preseed.cfg` file included in `config/binary_debian-installer/`. For example, to preseed setting the locale to `en_US`:

```
566 $ echo "d-i debian-installer/locale string en_US" \
    >> config/binary_debian-installer/preseed.cfg
```

12.3 Customizing Debian Installer content

568 For experimental or debugging purposes, you might want to include locally built `d-i` component `udeb` packages. Place these in `config/packages.binary/` to include them in the image. Additional or replacement files and directories may be included in the installer `initrd` as well, in a similar fashion to `<Live/chroot local includes>`, by placing the material in `config/binary_debian-installer-includes/`.

Project

13. Reporting bugs

571 Debian Live is far from being perfect, but we want to make it as close as possible to perfect - with your help. Do not hesitate to report a bug: it is better to fill a report twice than never. However, this chapter includes recommendations how to file good bug reports.

For the impatient:

- 573 • Always check first the image status updates on our homepage at <http://live.debian.net/> for known issues.
- 574 • Always try to reproduce the bug with the **most recent versions** of *live-build*, *live-boot*, and *live-config* before submitting a bug report.
- 575 • Try to give **as specific information as possible** about the bug. This includes (at least) the version of *live-build*, *live-boot*, and *live-config* used and the distribution of the live system you are building.

13.1 Known issues

577 Because Debian **testing** and Debian **unstable** distributions are a moving target, when you specify either as the target system distribution, a successful build may not always be possible.

578 If this causes too much difficulty for you, do not build a system based on **testing** or **unstable**, but rather, use **stable**. *live-build* does always default to the **stable** release.

579 Currently known issues are listed under the section 'status' on our homepage at <http://live.debian.net/>.

580 It is out of the scope of this manual to train you to correctly identify and fix problems in packages of the development distributions, however, there are two things you can always try: If a build fails when the target distribution is **testing**, try **unstable**. If **unstable** does not work either, revert to **testing** and pin the newer version of the failing package from **unstable** (see [APT pinning](#) for details).

581 13.2 Rebuild from scratch

582 To ensure that a particular bug is not caused by an uncleanly built system, please always rebuild the whole live system from scratch to see if the bug is reproducible.

583 13.3 Use up-to-date packages

584 Using outdated packages can cause significant problems when trying to reproduce (and ultimately fix) your problem. Make sure your build system is up-to-date and any packages included in your image are up-to-date as well.

585/6

13.4 Collect information

Please provide enough information with your report. At least include the exact version of *live-build* version where the bug is encountered and steps to reproduce it. Please use common sense and include other relevant information if you think that it might help in solving the problem.

To make the most out of your bug report, we require at least the following information:

- Architecture of the host system 588
- Version of *live-build* on the host system 589
- Version of *live-boot* on the live system 590
- Version of *live-config* on the live system 591
- Version of *debootstrap* and/or *cdebootstrap* on the host system 592
- Architecture of the live system 593
- Distribution of the live system 594
- Version of the kernel on the live system 595

You can generate a log of the build process by using the `tee` command. We recommend doing this automatically with an `auto/build` script; (see [Managing a configuration](#) for details).

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

At boot time, *live-boot* stores a log in `/var/log/live.log` (or `/var/log/live-boot.log`).

Additionally, to rule out other errors, it is always a good idea to tar up your `config/` directory and upload it somewhere (do **not** send it as an attachment to the mailing list), so that we can try to reproduce the errors you encountered. If this is difficult (e.g. due to size) you can use the output of `lb config --dump` which produces a summary

of your config tree (i.e. lists files in subdirectories of `config/` but does not include them).

600 Remember to send in any logs that were produced with English locale settings, e.g. run your *live-build* commands with a leading `LC_ALL=C` or `LC_ALL=en_US`.

601 **13.5 Isolate the failing case if possible**

602 If possible, isolate the failing case to the smallest possible change that breaks. It is not always easy to do this, so if you can't manage it for your report, don't worry. However, if you plan your development cycle well, using small enough change sets per iteration, you may be able to isolate the problem by constructing a simpler 'base' configuration that closely matches your actual configuration plus just the broken change set added to it. If you have a hard time sorting out which of your changes broke, it may be that you are including too much in each change set and should develop in smaller increments.

603 **13.6 Use the correct package to report the bug against**

604 Where does the bug appear?

605 **13.6.1 At build time whilst bootstrapping**

606 *live-build* first bootstraps a basic Debian system with `debootstrap` or `cdebootstrap`. Depending on the bootstrapping tool used and the Debian distribution it is bootstrapping, it may fail. If a bug appears here, check if the error is related to a specific Debian package (most likely), or if it is related to bootstrapping tool itself.

607 In both cases, this is not a bug in Debian Live, but rather in Debian

itself which we can not fix this directly. Please report such a bug against the bootstrapping tool or the failing package.

608 **13.6.2 At build time whilst installing packages**

609 *live-build* installs additional packages from the Debian archive and depending on the Debian distribution used and the daily archive state, it can fail. If a bug appears here, check if the error is also reproducible on a normal system.

610 If this is the case, this is not a bug in Debian Live, but rather in Debian - please report it against the failing package. Running `debootstrap` separately from the Live system build or running `lb bootstrap --debug` will give you more information.

611 Also, if you are using a local mirror and/or any of sort of proxy and you are experiencing a problem, please always reproduce it first by bootstrapping from an official mirror.

612 **13.6.3 At boot time**

613 If your image does not boot, please report it to the mailing list together with the information requested in [Collect information](#). Do not forget to mention, how/when the image failed, in Qemu, Virtualbox, VMWare or real hardware. If you are using a virtualization technology of any kind, please always run it on real hardware before reporting a bug. Providing a screenshot of the failure is also very helpful.

614 **13.6.4 At run time**

615 If a package was successfully installed, but fails while actually running the Live system, this is probably a bug in Debian Live. However,

616 13.7 Do the research

617 Before filing the bug, please search the web for the particular error message or symptom you are getting. As it is highly unlikely that you are the only person experiencing a particular problem, there is always a chance that it has been discussed elsewhere, and a possible solution, patch, or workaround has been proposed.

618 You should pay particular attention to the Debian Live mailing list, as well as the homepage, as these are likely to contain the most up-to-date information. If such information exists, always include the references to it in your bug report.

619 In addition, you should check the current bug lists for *live-build*, *live-boot*, and *live-config* to see whether something similar has been reported already.

620 13.8 Where to report bugs

621 The Debian Live project keeps track of all bugs in the Debian Bug Tracking System (BTS). For information on how to use the system, please see <http://bugs.debian.org/>. You can also submit the bugs by using the `reportbug` command from the package with the same name.

622 In general, you should report build time errors against the *live-build* package, boot time errors against *live-boot*, and run time errors against *live-config*. If you are unsure of which package is appropriate or need more help before submitting a bug report, please send a message to the mailing list and we will help you to figure it out.

623 Please note that bugs found in distributions derived from Debian (such as Ubuntu and others) should **not** be reported to the Debian BTS unless they can be also reproduced on a Debian system using official Debian packages.

14. Coding Style 624

This chapter documents the coding style used in *live-boot* and others. 625

14.1 Compatibility 626

- Don't use syntax or semantics that are unique to the Bash shell. For example, the use of array constructs. 627
- Only use the POSIX subset - for example, use `$(foo)` over ``foo``. 628
- You can check your scripts with ``sh -n`` and ``checkbashisms``. 629

14.2 Indenting 630

- Always use tabs over spaces. 631

14.3 Wrapping 632

- Generally, lines are 80 chars at maximum. 633
- Use the “Linux style” of line breaks: 634

Bad: 635

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

Good: 637

```
if foo
then
    bar
fi
```

- The same holds for functions: 639

Bad:

```
641     foo () {
           bar
        }
```

Good:

```
643     foo ()
        {
           bar
        }
```

14.4 Variables

- 645 • Variables are always in capital letters.
- 646 • Variables that used in `lb config` always start with `LB_` prefix.
- 647 • Internal temporary variables in *live-build* should start with the `<=underscore>LB_` prefix.
- 648 • Local variables start with *live-build* `<=underscore><=underscore>LB_` prefix.
- 649 • Variables in connection to a boot parameter in *live-config* start with `LIVE_`.
- 650 • All other variables in *live-config* start with `_` prefix.
- 651 • Use braces around variables; e.g. write `${FOO}` instead of `$FOO`.
- 652 • Always protect variables with quotes to respect potential whitespaces: write `"${FOO}"` not `${FOO}`.
- 653 • For consistency reasons, always use quotes when assigning values to variables:

Bad:

```
654     FOO=bar
```

Good:

640

```
F00="bar"
```

657

- If multiple variables are used, quote the full expression:

Bad:

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

Good:

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

14.5 Miscellaneous

- 664 • Use `|` (without the surround quotes) as a separator in calls to `sed`, e.g. `"sed -e `s|`"` (without `""`).
- 665 • Don't use the `test` command for comparisons or tests, use `"[" "` (without `""`); e.g. `"if [-x /bin/foo]; ..."` and not `"if test -x /bin/foo; ..."`.
- 666 • Use `case` wherever possible over `test`, as it's easier to read and faster in execution.

15. Procedures

This chapter documents the procedures within the Debian Live project for various tasks that need cooperation with other teams in Debian.

15.1 Udeb Uploads

670 Before committing releases of a udeb in d-i svn, one has to
call:

```
671 $ ../../scripts/l10n/output-l10n-changes . -d
```

15.2 Major Releases

673 Releasing a new stable major version of Debian includes a lot of
different teams working together to make it happen. At some point,
the Live team comes in and builds live system images. The require-
ments to do this are:

- 674 • A mirror containing the released versions for the debian, debian-
security and debian-volatile archive which the debian-live build
can access.
- 675 • The names of the image need to be known (e.g. debian-live-
VERSION-ARCH-FLAVOUR.iso).
- 676 • The packagelists need to have been updated.
- 677 • The data from debian-cd needs to be synced (udeb exclude lists).
- 678 • The includes from debian-cd needs to be synced (README.*,
doc/*, etc.).
- 679 • Images are built and mirrored on cdimage.debian.org.

15.3 Point Releases

- 681 • Again, we need updated mirror of debian, debian-security and
debian-volatile.
- 682 • Images are built and mirrored on cdimage.debian.org.
- 683 • Send announcement mail.

669 15.3.1 Point release announcement template

684

An announcement mail for point releases can be generated using
the template below and the following command:

```
$ sed \  
-e 's|%major%|5.0' \  
-e 's|%minor%|5.0.2' \  
-e 's|%codename%' \  
-e 's|%release_mail%|2009/msg00007.html'
```

686

Please check the mail carefully before sending and pass it to others
for proof-reading.

687

```
Debian Live images for Debian GNU/Linux %major% updated
```

688

```
The Debian Live project is pleased to announce the  
availability of  
updated Live images for its stable distribution Debian  
GNU/Linux %major%  
(codename "%codename%").
```

```
The images are available for download at:
```

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

```
This update incorporates the changes made in the %minor%  
point release,  
which adds corrections for security problems to the stable  
release  
along with a few adjustments for serious problems. A full  
list of the  
changes may be viewed at:
```

```
<http://lists.debian.org/debian-announce/%release\_mail%>
```

```
It also includes the following Live-specific changes:
```

- * [INSERT LIVE-SPECIFIC CHANGE HERE]
- * [INSERT LIVE-SPECIFIC CHANGE HERE]

* [LARGER ISSUES MAY DESERVE THEIR OWN SECTION]

URLs

Download location of updated images:

[<http://cdimage.debian.org/cdimage/release/current-live/>](http://cdimage.debian.org/cdimage/release/current-live/)

Debian Live project homepage:

[<http://live.debian.net/>](http://live.debian.net/)

The current stable distribution:

[<http://ftp.debian.org/debian/dists/stable>](http://ftp.debian.org/debian/dists/stable)

stable distribution information (release notes, errata etc.):

[<http://www.debian.org/releases/stable/>](http://www.debian.org/releases/stable/)

Security announcements and information:

[<http://www.debian.org/security/>](http://www.debian.org/security/)

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 GNU/Linux.

About Debian Live

Debian Live is an official sub-project of Debian which produces Debian systems that do not require a classical installer. Images are available for CD/DVD discs, USB sticks and PXE netbooting as well as a

bare

filesystem images for booting directly from the internet.

Contact Information

For further information, please visit the Debian Live web pages at

[<http://live.debian.net/>](http://live.debian.net/) or alternatively send mail to [<debian-live@lists.debian.org>](mailto:debian-live@lists.debian.org).

Examples

689

16. Examples

690

This chapter covers example builds for specific use cases with Debian Live. If you are new to building your own Debian Live images, we recommend you first look at the three tutorials in sequence, as each one teaches new techniques that will help you use and understand the remaining examples.

691

16.1 Using the examples

692

To use these examples you need a system to build them on that meets the requirements listed in [<Requirements>](#) and has *live-build* installed as described in [<Installing live-build>](#).

693

Note that, for the sake of brevity, in these examples we do not specify a local mirror to use for the build. You can speed up downloads considerably if you use a local mirror. You may specify the options when you use `lb config`, as described in [<Distribution mirrors used at build time>](#), or for more convenience, set the default for your build system in `/etc/live/build.conf`. Simply create this file and in it, set the corresponding `LB_PARENT_MIRROR_*` variables

694

to your preferred mirror. All other mirrors used in the build will be defaulted from these values. For example:

```
695     LB_PARENT_MIRROR_BOOTSTRAP="http://mirror/debian"
        LB_PARENT_MIRROR_CHROOT_SECURITY="http:
//mirror/debian-security"
        LB_PARENT_MIRROR_CHROOT_BACKPORTS="http:
//mirror/debian-updates"
```

696 16.2 Tutorial 1: A standard image

697 **Use case:** Create a simple first image, learning the basics of *live-build*.

698 In this tutorial, we will build a default ISO hybrid Debian Live image containing only base packages (no Xorg) and some Debian Live support packages, as a first exercise in using *live-build*.

699 You can't get much simpler than this:

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

701 Examine the contents of the `config/` directory if you wish. You will see stored here a skeletal configuration, ready to customize or, in this case, use immediately to build a default image.

702 Now, as superuser, build the image, saving a log as you build with `tee`.

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

704 Assuming all goes well, after a while, the current directory will contain `binary-hybrid.iso`. This ISO hybrid image can be booted directly in a virtual machine as described in [<Testing an ISO image with Qemu>](#) and [<Testing an ISO image with virtualbox-ose>](#), or else imaged onto optical media or a USB flash device as described in [<Burning an ISO image to a physical medium>](#) and [<Copying an ISO hybrid image to a USB stick>](#), respectively.

16.3 Tutorial 2: A web browser utility

Use case: Create a web browser utility image, learning how to apply customizations.

In this tutorial, we will create an image suitable for use as a web browser utility, serving as an introduction to customizing Debian Live images.

```
708     $ mkdir tutorial2
        $ cd tutorial2
        $ lb config -p lxde
        $ echo iceweasel >>
config/package-lists/my.list.chroot
```

Our choice of LXDE for this example reflects our desire to provide a minimal desktop environment, since the focus of the image is the single use we have in mind, the web browser. We could go even further and provide a default configuration for the web browser in `config/includes.chroot/etc/iceweasel/profile/`, or additional support packages for viewing various kinds of web content, but we leave this as an exercise for the reader.

Build the image, again as superuser, keeping a log as in [<Tutorial 1>](#):

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

Again, verify the image is OK and test, as in [<Tutorial 1>](#).

16.4 Tutorial 3: A personalized image

Use case: Create a project to build a personalized image, containing your favourite software to take with you on a USB stick wherever you go, and evolving in successive revisions as your needs and preferences change.

Since we will be changing our personalized image over a number

of revisions, and we want to track those changes, trying things experimentally and possibly reverting them if things don't work out, we will keep our configuration in the popular `git` version control system. We will also use the best practice of autoconfiguration via auto scripts as described in [«Managing a configuration»](#).

16.4.1 First revision

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

Edit `auto/config` to read as follows:

```
#!/bin/sh

lb config noauto \
  --architecture i386 \
  --linux-flavours 686-pae \
  --package-lists lxde \
  "${@}"
```

Now populate your local package list:

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

First, `--architecture i386` ensures that on our amd64 build system, we build a 32-bit version suitable for use on most machines. Second, we use `--linux-flavours 686-pae` because we don't anticipate using this image on much older systems. Third, we've chosen the `lxde` package list to give us a minimal desktop. And finally, we have added two initial favourite packages: `iceweasel` and `xchat`.

Now, build the image:

```
# lb build
```

Note that unlike in the first two tutorials, we no longer have to type `2>&1 | tee binary.log` as that is now included in `auto/build`.

Once you've tested the image (as in [«Tutorial 1»](#)) and are satisfied it works, it's time to initialize our `git` repository, adding only the auto scripts we just created, and then make the first commit:

```
$ git init
$ git add auto
$ git commit -a -m "Initial import."
```

16.4.2 Second revision

In this revision, we're going to clean up from the first build, add the `vlc` package to our configuration, rebuild, test and commit.

The `lb clean` command will clean up all generated files from the previous build except for the cache, which saves having to re-download packages. This ensures that the subsequent `lb build` will re-run all stages to regenerate the files from our new configuration.

```
# lb clean
```

Now append the `vlc` package to our local package list in `config/package-lists/my.list.chroot`:

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

Build again:

```
# lb build
```

Test, and when you're satisfied, commit the next revision:

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

Of course, more complicated changes to the configuration are possible, perhaps adding files in subdirectories of `config/`. When you commit new revisions, just take care not to hand edit or commit the top-level files in `config` containing `LB_*` variables, as these are

build products, too, and are always cleaned up by `lb clean` and re-created with `lb config` via their respective auto scripts.

739 We've come to the end of our tutorial series. While many more kinds of customization are possible, even just using the few features explored in these simple examples, an almost infinite variety of different images can be created. The remaining examples in this section cover several other use cases drawn from the collected experiences of users of Debian Live.

740 16.5 A VNC Kiosk Client

741 **Use case:** Create an image with *live-build* to boot directly to a VNC server.

742 Make a build directory and create a skeletal configuration in it built around the standard-x11 list, including `gdm3`, `metacity` and `xvnc4viewer`, disabling recommends to make a minimal system:

```
743 $ mkdir vnc_kiosk_client
    $ cd vnc_kiosk_client
    $ lb config -a i386 -k 686-pae -p standard-x11 \
      --apt-recommends false
    $ echo "gdm3 metacity xvnc4viewer" >>
config/package-lists/my.list.chroot
```

744 Create the directory `/etc/skel` and put a custom `.xsession` in it for the default user that will launch `metacity` and start `xvncviewer`, connecting to port 5901 on a server at `192.168.1.2`:

```
745 $ mkdir -p config/includes.chroot/etc/skel
    $ cat > config/includes.chroot/etc/skel/.xsession
<<END
    #!/bin/sh

    /usr/bin/metacity &
```

```
/usr/bin/xvncviewer 192.168.1.2:1
```

```
exit
END
```

Build the image:

```
# lb build
```

Enjoy.

16.6 A base image for a 128M USB key

750 **Use case:** Create a standard image with some components removed in order to fit on a 128M USB key with space left over to use as you see fit.

751 When optimizing an image to fit a certain media size, you need to understand the tradeoffs you are making between size and functionality. In this example, we trim only so much as to make room for additional material within a 128M media size, but without doing anything to destroy integrity of the packages contained within, such as the purging of locale data via the `localepurge` package, or other such “intrusive” optimizations. Of particular note, you should not use `--bootstrap-flavour minimal` unless you really know what you're doing, as omitting priority important packages will most likely produce a broken live system.

```
752 $ lb config -k 486 -p minimal --apt-indices false \
    --memtest none --apt-recommends false --includes
none
```

Now, build the image in the usual way:

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

754 On the author's system at time of writing, the above configuration 755 produced a 78Mbyte image. This compares favourably with the 166Mbyte image produced by the default configuration in [Tutorial 1](#).

756 The biggest space-saver here, compared to building a standard image on an i386 architecture system, is to select only the 486 kernel flavour instead of the default `-k "486 686-pae"`. Leaving off APT's indices with `--apt-indices false` also saves a fair amount of space, the tradeoff being that you need to `apt-get update` before using `apt` in the live system. Choosing the minimal package list leaves out the large `locales` package and associated utilities. Dropping recommended packages with `--apt-recommends false` saves some additional space, at the expense of omitting some packages you might otherwise expect to be there, such as `firmware-linux-free` which may be needed to support certain hardware. The remaining options shave off additional small amounts of space. It's up to you to decide if the functionality that is sacrificed with each optimization is worth the loss in functionality.

757 16.7 A localized KDE desktop and installer

758 **Use case:** Create a KDE desktop image, localized for Brazilian Portuguese and including an installer.

759 We want to make an iso-hybrid image for i386 architecture using our preferred desktop, in this case KDE, containing all of the same packages that would be installed by the standard Debian installer for KDE.

760 Our initial problem is the discovery of the names of the appropriate language tasks. Currently, *live-build* cannot help with this. While we might get lucky and find this by trial-and-error, there is a tool, `grep-dctrl`, which can be used to dig it out of the task descriptions in `tasksel-data`, so to prepare, make sure you have both of those things:

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

762 Now we can search for the appropriate tasks, first with:

```
$ grep-dctrl -FTest-lang pt_BR 763
/usr/share/tasksel/descs/debian-tasks.desc -sTask
Task: brazilian-portuguese
```

By this command, we discover the task is called, plainly enough, `brazilian-portuguese`. Now to find the related tasks:

```
$ grep-dctrl -FEnhances brazilian-portuguese 765
/usr/share/tasksel/descs/debian-tasks.desc -sTask
Task: brazilian-portuguese-desktop
Task: brazilian-portuguese-kde-desktop
```

At boot time we will generate the `pt_BR.UTF-8` locale and select the `pt-latin1` keyboard layout. We will also need to preseed our desktop choice, "kde" so that `tasksel` will install the correct desktop task, as it differs from the default (see [Desktop and languages tasks](#)). Now let's put the pieces together:

```
$ mkdir live-pt_BR-kde 767
$ cd live-pt_BR-kde
$ lb config \
  -a i386 \
  -k 486 \
  --bootappend-live "locales=pt_BR.UTF-8
keyboard-layouts=pt-latin1" \
  --debian-installer live
$ echo kde-desktop brazilian-portuguese
brazilian-portuguese-desktop \
  brazilian-portuguese-kde-desktop >>
config/task-lists/my.list.chroot
$ echo debian-installer-launcher >>
config/package-lists/my.list.chroot
$ echo tasksel tasksel/desktop multiselect kde >>
config/preseed/my.preseed.chroot
```

Note that we have included the `debian-installer-launcher` package to launch the installer from the live desktop, and have also specified the 486 flavour kernel, as it is currently necessary to make the installer and live system kernels match for the launcher to work properly.

Metadata

SiSU Metadata, document information

Document Manifest @:

<http://live.debian.net/manual/en/live-manual/manifest.html>

Title: Debian Live Manual

Creator: Debian Live Project <debian-live@lists.debian.org>

Rights: Copyright (C) 2006-2011 Debian Live 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: Debian Live Project <debian-live@lists.debian.org>

Date: 2011-11-13

Version Information

Sourcefile: live-manual.ssm.sst

Filetype: SiSU text 2.0

Source Digest: SHA256(live-manual.ssm.sst)=b81e95585839755ced2d05c5-8332288719f8fcaa07d502c3f86dab32cb270b43

Skin Digest: SHA256(skin_debian-live.rb)=be92275c5ee3367edeed653901c34601-c545c50acecc23ab65594d8e2f4df9af

Generated

Document (dal) last generated: 2011-11-21 13:14:08 +0000

Generated by: SiSU 3.1.3 of 2011w44/6 (2011-11-05)

Ruby version: ruby 1.8.7 (2011-06-30 patchlevel 352) [x86_64-linux]