

Manuale di Live Systems

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

Copyright © 2006-2014 Live Systems Project

Questo programma è software libero: è possibile ridistribuirlo e modificarlo secondo i termini della GNU General Public License come pubblicata dalla Free Software Foundation, sia la versione 3 della licenza o (a scelta) una versione successiva.

Questo programma è distribuito nella speranza che possa essere utile, ma SENZA ALCUNA GARANZIA, nemmeno la garanzia implicita di COMMERCIALIZZABILITÀ o IDONEITÀ PER UN PARTICOLARE SCOPO. Vedere la GNU General Public License per ulteriori dettagli.

Si dovrebbe aver ricevuto una copia della GNU General Public License con questo programma. In caso contrario, vedere <http://www.gnu.org/licenses/>.

Il testo completo della GNU General Public License può essere trovato nel file /usr/share/common-licenses/GPL-3.

Contents		Installazione	10
A proposito	2	3. Installazione	10
A proposito di questo manuale	3	3.1 Requisiti	10
1. A proposito di questo manuale	3	3.2 Installare live-build	10
1.1 Per gli impazienti	3	3.2.1 Dal repository Debian	10
1.2 Glossario	3	3.2.2 Da sorgenti	10
1.3 Autori	4	3.2.3 Da “istantanee”	11
1.4 Contribuire a questo documento	5	3.3 Installare live-boot e live-config	11
1.4.1 Applying changes	5	3.3.1 Dal repository Debian	11
1.4.2 Traduzione	5	3.3.2 Da sorgenti	11
		3.3.3 Da “istantanee”	12
About the Live Systems Project	7	Nozioni di base	13
2. About the Live Systems Project	7	4. Nozioni di base	13
2.1 Motivazioni	7	4.1 Cos'è un sistema live?	13
2.1.1 Cosa c'è di sbagliato con gli attuali sistemi live	7	4.2 Scaricare immagini precompilate	14
2.1.2 Perché creare il proprio sistema live?	7	4.3 Utilizzare il web builder per le immagini live	14
2.2 Filosofia	7	4.3.1 Utilizzo del web builder e raccomandazioni	14
2.2.1 Solamente pacchetti da Debian “main”, inalterati.	7	4.4 Primi passi: creare un'immagine ISO ibrida	14
2.2.2 Nessun pacchetto di configurazione per il sistema live	7	4.5 Utilizzare un'immagine ISO live ibrida	15
2.3 Contatti	8	4.5.1 Masterizzare un'immagine ISO su un supporto fisico	15
Utente	9	4.5.2 Copiare un'immagine ISO ibrida su una penna USB	15
		4.5.3 Usare lo spazio rimanente su una penna USB	16
		4.5.4 Avviare il supporto live	16
		4.6 Utilizzare una macchina virtuale per le prove	16
		4.6.1 Provare un'immagine ISO con QEMU	17
		4.6.2 Testing an ISO image with VirtualBox	17
		4.7 Creare e utilizzare un'immagine HDD	17

4.8 Creare un'immagine netboot	18	Personalizzazione dei contenuti	26
4.8.1 Server DHCP	19	7. Panoramica sulla personalizzazione	26
4.8.2 Server TFTP	19	7.1 Configurazione in fase di compilazione e di avvio . . .	26
4.8.3 Server NFS	20	7.2 Fasi della creazione	26
4.8.4 Come provare una netboot	20	7.3 Integrare la configurazione di lb con dei file	26
4.8.5 Qemu	20	7.4 Personalizzazione dei compiti	27
4.9 Webbooting	20	Personalizzare l'installazione dei pacchetti	28
4.9.1 Getting the webboot files	21	8. Personalizzare l'installazione dei pacchetti	28
4.9.2 Booting webboot images	21	8.1 Sorgenti dei pacchetti	28
Panoramica degli strumenti	22	8.1.1 Distribuzione, le aree di archivio e le modalità . . .	28
5. Panoramica degli strumenti	22	8.1.2 Mirror delle distribuzioni	29
5.1 Il pacchetto live-build	22	8.1.3 Mirror delle distribuzioni usati in fase di compilazione	29
5.1.1 Il comando lb config	22	8.1.4 Mirror delle distribuzioni usate durante l'esecuzione	29
5.1.2 Il comando lb build	23	8.1.5 Repository aggiuntivi	29
5.1.3 Il comando lb clean	23	8.2 Scegliere i pacchetti da installare	30
5.2 Il pacchetto live-boot	23	8.2.1 Elenchi di pacchetti	30
5.3 Il pacchetto live-config	23	8.2.2 Usare metapacchetti	30
Gestire una configurazione	24	8.2.3 Elenchi locali dei pacchetti	31
6. Gestire una configurazione	24	8.2.4 Elenchi locali di pacchetti binari	31
6.1 Gestire i cambiamenti di configurazione	24	8.2.5 Elenchi di pacchetti generati	31
6.1.1 Perché utilizzare gli script automatici? Cosa fanno?	24	8.2.6 Usare condizioni all'interno degli elenchi di pacchetti	32
6.1.2 Esempi d'uso di script automatici	24	8.2.7 Removing packages at install time	32
6.2 Clonare una configurazione pubblicata tramite Git.	25	8.2.8 Task per desktop e lingua	32
		8.2.9 Tipi e versioni del kernel	33
		8.2.10 Kernel personalizzati	33

8.3 Installare pacchetti modificati o di terze parti	34
8.3.1 Utilizzare packages.chroot per installare pacchetti personalizzati	34
8.3.2 Utilizzare un repository APT per installare pacchetti personalizzati	34
8.3.3 Pacchetti personalizzati e APT	35
8.4 Configurare APT in fase di compilazione	35
8.4.1 Scegliere apt o aptitude	35
8.4.2 Utilizzare un proxy con APT	35
8.4.3 Modificare APT per risparmiare spazio	35
8.4.4 Passare opzioni ad apt o aptitude	36
8.4.5 APT pinning	37
Personalizzazione dei contenuti	38
9. Personalizzazione dei contenuti	38
9.1 Include	38
9.1.1 Live/chroot include locali	38
9.1.2 Include locali binari	38
9.2 Hook	39
9.2.1 Live/chroot hook locali	39
9.2.2 Hook in fase di avvio	39
9.2.3 Hook binari locali	39
9.3 Preconfigurare le domande di Debconf	39
Personalizzare i comportamenti durante l'esecuzione	41
10. Personalizzare i comportamenti durante l'esecuzione	41
10.1 Personalizzare l'utente live	41
10.2 Personalizzare la localizzazione e la lingua	41
10.3 Persistenza	42
10.3.1 Il file persistence.conf	44

10.3.2 Utilizzare più di un'archiviazione persistente	44
10.4 Using persistence with encryption	45
Personalizzare l'immagine binaria	47
11. Personalizzare l'immagine binaria	47
11.1 Bootloader	47
11.2 Metadati ISO	47
Personalizzare l'Installatore Debian	48
12. Personalizzare l'Installatore Debian	48
12.1 Tipologie dell'Installatore Debian	48
12.2 Personalizzare il Debian Installer con la preconfigurazione	49
12.3 Personalizzare il contenuto dell'Installatore Debian	49
Progetto	50
Contribuire al progetto	51
13. Contribuire al progetto	51
13.1 Applicare le modifiche	51
Segnalare bug	53
14. Segnalare bug	53
14.1 Problemi noti	53
14.2 Ricompilare da zero	53
14.3 Usare pacchetti aggiornati	53

Contents

14.4 Raccogliere informazioni	53	Repository Git	61
14.5 Se possibile isolare il caso non andato a buon fine	54	17. Repository Git	61
14.6 Segnalare il bug del pacchetto giusto	54	17.1 Gestire repository multipli	61
14.6.1 Durante la compilazione mentre esegue il bootstrap	54	Esempi	63
14.6.2 Durante la compilazione mentre installa i pacchetti	55	Esempi	64
14.6.3 In fase di avvio	55	18. Esempi	64
14.6.4 In fase di esecuzione	55	18.1 Usare gli esempi	64
14.7 Fare la ricerca	55	18.2 Tutorial 1: un'immagine predefinita	64
14.8 Dove segnalare i bug	55	18.3 Tutorial 2: servizio browser web	65
Lo stile nello scrivere codice	57	18.4 Tutorial 3: un'immagine personalizzata	65
15. Lo stile nello scrivere codice	57	18.4.1 Prima revisione	65
15.1 Compatibilità	57	18.4.2 Seconda revisione	66
15.2 Rientri	57	18.5 Un client Kiosk VNC	67
15.3 Ritorno a capo	57	18.6 Un'immagine base per una chiavetta USB da 128MB	68
15.4 Variabili	57	18.7 Un desktop GNOME localizzato e l'installatore	69
15.5 Varie	58	Appendice	71
Procedure	59	Style guide	72
16. Procedure	59	19. Style guide	72
16.1 Rilasci importanti	59	19.1 Guidelines for authors	72
16.2 Rilasci minori	59	19.1.1 Linguistic features	72
16.2.1 Ultimo rilascio minore di un rilascio di Debian.	59	19.1.2 Procedures	73
16.2.2 Modello per l'annuncio di un rilascio minore.	59	19.2 Guidelines for translators	75
		19.2.1 Translation hints	75

SiSU Metadata, document information	77
--	-----------

1	Manuale di Live Systems
---	--------------------------------

A proposito

A proposito di questo manuale

1. A proposito di questo manuale

This manual serves as a single access point to all documentation related to the Live Systems Project and in particular applies to the software produced by the project for the Debian 8.0 “jessie” release. An up-to-date version can always be found at <http://live-systems.org/>

While *live-manual* 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 downloading prebuilt images and preparing images to be booted from media or the network, either using the web builder or running *live-build* directly on your system. **«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.

Alcuni dei comandi menzionati nel testo devono essere eseguiti con i privilegi di super-utente che possono essere ottenuti diventando utente root tramite `su` oppure usando `sudo`. Per distinguere i comandi che possono essere eseguiti come utente normale da quelli che necessitano dei privilegi di super-utente, i comandi sono preceduti rispettivamente da `$` o `#`. Questi simboli non fanno parte del comando.

1.1 Per gli impazienti

Sebbene crediamo che ogni cosa in questo manuale sia importante almeno per alcuni dei nostri utenti, ci rendiamo conto che c'è tanto materiale da trattare e che si potrebbe voler provare il software prima di entrare nei dettagli; pertanto suggeriamo di leggerlo nel

seguente ordine.

First, read this chapter, **«About this manual»**, from the beginning and ending with the **«Terms»** section. Next, skip to the three tutorials at the front of the **«Examples»** section designed to teach you image building and customization basics. Read **«Using the examples»** first, followed by **«Tutorial 1: A default 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 live systems.

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 live systems and motivated to read the rest of the manual, cover-to-cover.

1.2 Glossario

- **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 (via netboot images, see **«Building a netboot image»**), and over the Internet (via the boot parameter `fetch=URL`, see **«Webbooting»**).
- **Supporto Live** : diversamente dal sistema live, si riferisce a CD, DVD o penna USB dove viene scritto il binario prodotto da *live-build* e usato per l'avvio del sistema live. Più in generale, il termine si riferisce anche a qualsiasi posto in cui risiede il binario allo scopo di avviare il sistema, come il percorso per i file di avvio da rete.

15	• Live Systems Project : The project which maintains, among others, the <i>live-boot</i> , <i>live-build</i> , <i>live-config</i> , <i>live-tools</i> and <i>live-manual</i> packages.	28	• stable/testing/unstable : The stable distribution, currently codenamed wheezy , contains the latest officially released distribution of Debian. The testing distribution, temporarily codenamed jessie , 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, permanently codenamed sid , is where active development of Debian occurs. Generally, this distribution is run by developers and those who like to live on the edge. Throughout the manual, we tend to use codenames for the releases, such as jessie or sid , as that is what is supported by the tools themselves.	
16	• Sistema host : l'ambiente utilizzato per creare il sistema live.			
17	• Sistema di destinazione : l'ambiente usato per eseguire il sistema live.			
18	• live-boot : una raccolta di script usati per avviare sistemi live.			
19	• live-build : A collection of scripts used to build customized live systems.			
20	• live-config : una raccolta di script usati per configurare un sistema live durante il processo di avvio.			
21	• live-tools : una raccolta di script aggiuntivi usati per eseguire utili compiti in un sistema live avviato.			
22	• live-manual : questo documento è inserito nel pacchetto chiamato <i>live-manual</i> .			
23	• Debian Installer (d-i) : il sistema d'installazione ufficiale per la distribuzione Debian.			
24	• Boot parameters : parametri che possono essere immessi nel prompt del boot loader per modificare il comportamento del kernel o di <i>live-config</i> .			
25	• chroot : il programma <i>chroot</i> , <i>chroot(8)</i> , rende possibile eseguire diverse istanze dell'ambiente GNU/Linux su un singolo sistema simultaneamente senza riavviare.			
26	• Binary image : A file containing the live system, such as <i>live-image-i386.hybrid.iso</i> or <i>live-image-i386.img</i> .			
27	• Distribuzione di destinazione : la distribuzione su cui sarà basato il sistema live. Può differire dalla distribuzione presente sul proprio computer.			
			1.3 Autori	29
			Lista degli autori (in ordine alfabetico):	30
			• Ben Armstrong	31
			• Brendan Sleight	32
			• Carlos Zuferri	33
			• Chris Lamb	34
			• Daniel Baumann	35
			• Franklin Piat	36
			• Jonas Stein	37
			• Kai Hendry	38
			• Marco Amadori	39
			• Mathieu Geli	40
			• Matthias Kirschner	41
			• Richard Nelson	42
			• Trent W. Buck	43

1.4 Contribuire a questo documento

Questo manuale è pensato come un progetto comunitario e ogni suggerimento e contributo è benvenuto. Si veda [«Contribuire al progetto»](#) per informazioni dettagliate su come prelevare la chiave SSH ed eseguire buoni commit.

1.4.1 Applying changes

Per apportare modifiche alla versione inglese del manuale è necessario modificare i file giusti in `manual/en/`, ma prima di sottoporre il proprio contributo si prega di visionare l'anteprima del proprio lavoro. Per ottenere l'anteprima di *live-manual*, assicurarsi di avere installato i pacchetti necessari per la sua compilazione eseguendo:

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

Si può compilare il *live-manual* dalla directory superiore del checkout di Git eseguendo:

```
$ make build
```

Since it takes a while to build the manual in all supported languages, authors may find it convenient to use one of the fast proofing shortcuts when reviewing the new documentation they have added to the English manual. Using `PROOF=1` builds *live-manual* in html format, but without the segmented html files, and using `PROOF=2` builds *live-manual* in pdf format, but only the A4 and letter portraits. That is why using either of the `PROOF=` possibilities can save up a considerable amount of time, e.g:

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

When proofing one of the translations it is possible to build only one language by executing, e.g:

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

È inoltre possibile compilare in base al tipo di documento, esempio:

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

O entrambi:

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

Dopo aver revisionato il proprio lavoro e assicurato che tutto funzioni, non usare `make commit` a meno che nel commit non si stiano aggiornando delle traduzioni, in tal caso non mescolare nello stesso le modifiche al manuale inglese e le traduzioni ma eseguire un commit per ognuna. Per maggiori dettagli vedere la sezione [«Traduzione»](#).

1.4.2 Traduzione

In order to translate *live-manual*, follow these steps depending on

whether you are starting a translation from scratch or continue working on an already existing one:

- Start a new translation from scratch
 - Translate the **about_manual.ssi.pot**, **about_project.ssi.pot** and **index.html.in.pot** files in `manual/pot/` to your language with your favourite editor (such as *poedit*) and send the translated .po files to the mailing list to check their integrity. *live-manual*'s integrity check not only ensures that the .po files are 100% translated but it also detects possible errors.
 - Once checked, to enable a new language in the autobuild it is enough to add the initial translated files to `manual/-po/${LANGUAGE}/` and run `make commit`. And then, edit `manual/_sisu/home/index.html` adding the name of the language and its name in English between brackets.
- Continue with an already started translation
 - If your target language has already been added, you can randomly continue translating the remaining .po files in `manual/po/${LANGUAGE}/` using your favourite editor (such as *poedit*).
 - Do not forget that you need to run `make commit` to ensure that the translated manuals are updated from the .po files and then you can review your changes launching `make build` before `git add .`, `git commit -m "Translating..."` and `git push`. Remember that since `make build` can take a considerable amount of time, you can proofread languages individually as explained in [«Applying changes»](#)

Dopo aver eseguito `make commit` si vedrà del testo scorrere. Questi sono messaggi informativi sullo stato del processo e alcuni suggerimenti su cosa si può fare per migliorare *live-manual*. A

meno che non si ottenga un errore fatale si può procedere e inviare il proprio contributo.

live-manual comes with two utilities that can greatly help translators to find untranslated and changed strings. The first one is “make translate”. It launches a script that tells you in detail how many untranslated strings there are in each .po file. The second one, the “make fixfuzzy” target, only acts upon changed strings but it helps you to find and fix them one by one.

È da considerare che nonostante queste utilità possono davvero risultare utili per tradurre dalla riga di comando, si raccomanda l'uso di uno strumento specifico come *poedit*. È inoltre una buona idea leggere la documentazione sulla localizzazione in Debian (l10n) e, specifiche per *live-manual*, le [«Linee guida per i traduttori»](#).

Nota: si può usare `make clean` per pulire l'albero del repository git locale prima del push. Grazie al file `.gitignore` questo passo non è obbligatorio ma è una buona abitudine che evita di fare involontariamente il commit di certi file.

72	About the Live Systems Project		
73	2. About the Live Systems Project		
74	2.1 Motivazioni		
75	2.1.1 Cosa c'è di sbagliato con gli attuali sistemi live		
76	When Live Systems Project 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:		
77	• Non sono progetti Debian, per cui non sono supportati da Debian.		87
78	• Mischiano differenti distribuzioni come ad esempio: testing e unstable .		88
79	• Supportano solamente i386.		89
80	• Modificano l'aspetto e il comportamento dei pacchetti snellendoli per risparmiare spazio.		90
81	• Includono pacchetti esterni all'archivio Debian.		91
82	• Forniscono un kernel con patch aggiuntive che non appartengono a Debian.		92
83	• Sono grandi e lenti a causa delle loro dimensioni e non adatti per operazioni di salvataggio.		
84	• Non sono disponibili in diversi formati come CD, DVD, penne USB e immagini netboot.		
85	2.1.2 Perché creare il proprio sistema live?		
86	Debian è il Sistema Operativo Universale, ha un sistema live per mostrare e rappresentare accuratamente il sistema con i seguenti vantaggi:		
	• È un sottoprogetto di Debian.		87
	• Riflette lo stato (attuale) di una distribuzione.		88
	• Gira su più architetture possibili.		89
	• È costituito solo da pacchetti Debian non modificati.		90
	• Non contiene nessun pacchetto che non sia presente nell'archivio di Debian.		91
	• Usa un kernel Debian inalterato senza patch aggiuntive.		92
	2.2 Filosofia		93
	2.2.1 Solamente pacchetti da Debian "main", inalterati.		94
	Verranno usati solo pacchetti dal repository Debian della sezione "main". La sezione non-free non è parte di Debian perciò non possono essere affatto usati per le immagini ufficiali del sistema live.		95
	Non verrà cambiato nessun pacchetto. Nel caso in cui sarà necessario cambiare qualcosa sarà fatto in coordinazione con il maintainer del pacchetto Debian.		96
	In via eccezionale i nostri pacchetti come <i>live-boot</i> , <i>live-build</i> o <i>live-config</i> possono temporaneamente essere usati dal nostro repository per ragioni di sviluppo (ad esempio per creare istantanee). Verranno caricati regolarmente in Debian.		97
	2.2.2 Nessun pacchetto di configurazione per il sistema live		98
	In questa fase non saranno disponibili né esempi di installazione né configurazioni alternative. Tutti i pacchetti vengono usati con la		99

loro configurazione predefinita così come accade con una regolare installazione di Debian.

Nel caso in cui serva una configurazione predefinita differente, sarà fatto in coordinazione con il maintainer del pacchetto in Debian.

A system for configuring packages is provided using debconf allowing custom configured packages to be installed in your custom produced live system images, but for the **prebuilt live images** we choose to leave packages in their default configuration, unless absolutely necessary in order to work in the live environment. Wherever possible, we prefer to adapt packages within the Debian archive to work better in a live system versus making changes to the live toolchain or **prebuilt image configurations**. For more information, please see **Customization overview**.

2.3 Contatti

- **Mailing list** : il principale contatto del progetto è la mailing list <https://lists.debian.org/debian-live/>, si possono inviare email alla lista direttamente a debian-live@lists.debian.org. Gli archivi sono disponibili presso <https://lists.debian.org/debian-live/>.
- **IRC** : molti utenti e sviluppatori sono presenti sul canale #debian-live su irc.debian.org (OFTC). Quando si pone una domanda su IRC, si prega di essere pazienti nell'ottenere una risposta; se non si riceve risposta scrivere alla mailing list.
- **BTS** : il **Segnalare bug**.

Utente

107 Installazione

108 3. Installazione

109 3.1 Requisiti

110 Building live system images has very few system requirements:

- 111 • Accesso come utente root
- 112 • Una versione aggiornata di *live-build*
- 113 • Una shell POSIX-compliant, come *bash* o *dash*
- 114 • *debootstrap* o *cdebootstrap*
- 115 • Linux 2.6 o successivi

116 Si noti che usare Debian o una distribuzione derivata Debian non è richiesto - *live-build* funzionerà sostanzialmente su qualsiasi distribuzione che soddisfi i requisiti di cui sopra.

117 3.2 Installare live-build

118 Si può installare *live-build* in diversi modi:

- 119 • Dal repository Debian
- 120 • Da sorgenti
- 121 • Da istantanee

122 Se si sta usando Debian, il metodo raccomandato è di installare *live-build* attraverso il repository Debian.

123 3.2.1 Dal repository Debian

124 Installare *live-build* come qualsiasi altro pacchetto:

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

3.2.2 Da sorgenti

live-build è sviluppato usando il sistema di controllo versione Git. Sui sistemi basati su Debian è fornito dal pacchetto *git*. Per scaricare il codice aggiornato, eseguire:

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

È possibile costruirsi ed installarsi il proprio pacchetto Debian eseguendo:

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

Si installino ora i file *.deb* appena generati ai quali si è interessati, ad esempio:

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

Si può anche installare *live-build* direttamente sul proprio sistema eseguendo:

```
# make install
```

e disinstallarlo con:

```
# make uninstall
```

3.2.3 Da “istantanee”

Se non si desidera generare o installare *live-build* da sorgenti, è possibile usare le istantanee. Sono costruite automaticamente dall'ultima versione presente su Git e disponibili su <http://live-systems.org/debian/>.

3.3 Installare live-boot e live-config

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

3.3.1 Dal repository Debian

Sia *live-boot* che *live-config* sono disponibili dai repository Debian come per l'installazione di *live-build*.

3.3.2 Da sorgenti

Per utilizzare i sorgenti più recenti da Git si può seguire il procedimento seguente. Assicurarsi di conoscere i termini menzionati nel [Glossario](#).

- Scaricare i sorgenti di *live-boot* e *live-config*

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

Consultare la pagine man di *live-boot* e *live-config* per i dettagli sulla personalizzazione se questa è il motivo per compilare questi pacchetti dai sorgenti.

- Costruire un .deb di *live-boot* e *live-config*

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

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

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

- Usare il .deb di *live-boot* generato

As *live-boot* and *live-config* are installed by *live-build* system, installing the packages in the host system is not sufficient: you should treat the generated .deb files like any other custom packages. Since your purpose for building from source is likely to test new things over the short term before the official release,

follow <Installing modified or third-party packages> to temporarily include the relevant files in your configuration. In particular, notice that both packages are divided into a generic part, a documentation part and one or more back-ends. Include the generic part, only one back-end matching your configuration, and optionally the documentation. Assuming you are building a live image in the current directory and have generated all .deb files for a single version of both packages in the directory above, these bash commands would copy all of the relevant packages including default back-ends:

154

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

155

3.3.3 Da “istantanee”

156

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

157 Nozioni di base

158 4. Nozioni di base

159 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 medium or USB portable storage device. In certain special cases, as explained later, the `hdd` type may be more suitable. The chapter includes detailed instructions for building and using a `netboot` type image, which is a bit more involved due to the setup required on the server. This is an slightly advanced topic for anyone who is not already familiar with netbooting, but it 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.

160 The section finishes with a quick introduction to `<webbooting>` which is, perhaps, the easiest way of using different images for different purposes, switching from one to the other as needed using the internet as a means.

161 In tutto il capitolo faremo spesso riferimento ai nomi dei file predefiniti creati da *live-build*. Se invece si `<scarica un'immagine precompilata>`, i nomi possono variare.

162 4.1 Cos'è un sistema live?

163 Per sistema live generalmente si intende un sistema operativo che può essere avviato da un supporto rimovibile, come un CD-ROM o una chiavetta USB oppure da una rete, pronto per l'uso senza alcuna installazione su hard disk con una configurazione automatica fatta durante l'esecuzione (vedere `<Glossario>`).

164 With live systems, it's an operating system, built for one of the

supported architectures (currently amd64 and i386). It is made from the following parts:

- **Immagine del kernel Linux** , comunemente chiamata `vmlinuz*` 165
- **Initial RAM disk image (initrd)** : un disco RAM creato per il boot di Linux, contenente i moduli potenzialmente necessari per montare l'immagine di sistema e alcuni script per farlo. 166
- **System image** : The operating system's filesystem image. Usually, a SquashFS compressed filesystem is used to minimize the live system image size. Note that it is read-only. So, during boot the live system will use a RAM disk and 'union' mechanism to enable writing files within the running system. However, all modifications will be lost upon shutdown unless optional persistence is used (see `<Persistence>`). 167
- **Bootloader** : una piccola porzione di codice predisposto per l'avvio dal supporto scelto, che presenta un prompt o un menu per la selezione di opzioni/configurazioni. Carica il kernel Linux ed il suo `initrd` da eseguire con un filesystem associato. Possono essere usate diverse soluzioni, in base al supporto di destinazione ed al formato del filesystem contenenti le componenti precedentemente citate: `isolinux` per il boot da CD o DVD nel formato ISO9660, `syslinux` per supporti HDD o USB che si avviano da una partizione VFAT, `extlinux` per le partizioni `ext/2/3/4` e `btrfs`, `pxelinux` per il netboot PXE, GRUB per partizioni `ext2/3/4`, ecc. 168

È possibile usare *live-build* per creare l'immagine di sistema secondo le proprie specifiche, scegliere un kernel Linux, il suo `initrd` ed un bootloader per avviarli, tutto in un unico formato che dipende dal mezzo (immagini ISO9660, immagine disco, ecc.) 169

4.2 Scaricare immagini precompilate

171 Nonostante l'obiettivo di questo manuale è di sviluppare e creare
le proprie immagini live si potrebbe semplicemente voler provare
una di quelle precompilate, sia come introduzione al loro uso o per
costruirne una propria. Queste immagini sono create utilizzando
il nostro `<repository git live-images>` e i rilasci ufficiali di stable
pubblicati all'indirizzo `<https://www.debian.org/CD/live/>`. In aggiunta,
all'indirizzo `<http://live-systems.org/cdimage/release/>` sono disponibili le
versioni vecchie e future e le immagini non ufficiali contenenti
firmware e driver non-free.

4.3 Utilizzare il web builder per le immagini live

172 Come servizio alla comunità, all'indirizzo `<http://live-build.debian.net/>`
abbiamo un servizio web per la creazione di immagini live, il sito è
mantenuto col massimo sforzo possibile. Sebbene ci impegnamo
nel tenerlo aggiornato e funzionante e notificiamo eventuali
interruzioni, non possiamo garantire al 100% la sua disponibilità
o una compilazione veloce dell'immagine; occasionalmente il
servizio potrebbe avere problemi che richiedono tempo per essere
risolti. Se si hanno problemi o domande in proposito `<contattarci>`
fornendo il link della propria compilazione.

4.3.1 Utilizzo del web builder e raccomandazioni

174 Attualmente l'interfaccia web non previene l'uso di combinazioni
non valide di opzioni, in particolare dove la modifica di un'opzione
(ovvero usando *live-build* direttamente) cambia i valori predefiniti
di altre opzioni elencate nel form web, il web builder non modifica
questi default. Se si cambia `--architectures` da `i386` a `amd64` è
necessario modificare l'opzione `--linux-flavours` corrispondente
da `486` a `amd64`. Per ulteriori dettagli sulla versione installata sul web

170 builder consultare la pagina di manuale di `lb_config`; il numero
della versione è scritto al fondo della pagina.

Le tempistiche fornite dal web builder sono solo una stima
approssimata e possono non riflettere quanto realmente ci vorrà
per compilare, né vengono aggiornate una volta apparse. Vi
preghiamo di essere pazienti e di non aggiornare la pagina dopo
aver commissionato la compilazione in quanto invierà la richiesta
per una nuova operazione con gli stessi parametri. Se, dopo aver
aspettato a sufficienza e verificato che l'email non sia finita nello
spam, non ricevete alcuna notifica allora `<contattateci>`.

Il web builder ha un limite di tipologie di immagini creabili,
mantenendosi semplice e efficiente da utilizzare e mantenere.
Le personalizzazioni non sono fornite dall'interfaccia web, il resto
di questo manuale sipegia come creare le proprie immagini tramite
live-build.

4.4 Primi passi: creare un'immagine ISO ibrida

Indipendentemente dal tipo di immagine, per crearne una è
necessario eseguire ogni volta la stessa procedura. Come
primo esempio si creerà una directory di lavoro, vi si entrerà e
si eseguirà la seguente sequenza di comandi di *live-build* per
creare un'immagine ISO ibrida di base contenente un sistema live
predefinito senza X.org. È adatta per essere masterizzata su CD
o DVD e anche per essere copiata su una penna USB.

Il nome della directory di lavoro è arbitrario ma guardando gli
esempi usati da *live-manual* è una buona idea utilizzare un nome
che aiuta a identificare in qualsiasi directory l'immagine su cui si
sta lavorando, in particolar modo se si stanno sperimentando tipi di
immagine differenti. In questo caso stiamo creando un sistema
predefinito quindi chiamiamolo ad esempio `live-default`.

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

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

```
$ lb config
```

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

Ora che si ha una gerarchia “config/” si può generare l’immagine con il comando `lb build`:

```
# lb build
```

This process can take a while, depending on the speed of your computer and your network connection. When it is complete, there should be a `live-image-i386.hybrid.iso` image file, ready to use, in the current directory.

Note: If you are building on an amd64 system the name of the resulting image will be `live-image-amd64.hybrid.iso`. Keep in mind this naming convention throughout the manual.

4.5 Utilizzare un’immagine ISO live ibrida

Dopo aver costruito o scaricato un’immagine ISO ibrida, ottenibile all’indirizzo <<https://www.debian.org/CD/live/>>, il passo successivo è preparare il supporto per l’avvio, che sia esso un CD-R(W), un DVD-R(W) o una penna USB.

4.5.1 Masterizzare un’immagine ISO su un supporto fisico

Masterizzare un’immagine ISO è semplice, basta installare *xorriso* e utilizzarlo da riga di comando; ad esempio:

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

4.5.2 Copiare un’immagine ISO ibrida su una penna USB

ISO images prepared with *xorriso*, can be simply copied to a USB stick with the `cp` 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`.

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

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

Note: The `sync` command is useful to ensure that all the data, which is stored in memory by the kernel while copying the image, is written to the USB stick.

4.5.3 Usare lo spazio rimanente su una penna USB

After copying the `live-image-i386.hybrid.iso` to a USB stick, the first partition on the device will be filled up by the live system. To use the remaining free space, use a partitioning tool such as *gparted* or *parted* to create a new partition on the stick.

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

Dopo aver creato la partizione, dove `${PARTITION}` è il nome della partizione, ad esempio `/dev/sdb2`, si deve creare su di essa un filesystem. Una scelta possibile potrebbe essere `ext4`.

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

Nota: se si desidera utilizzare lo spazio extra con Windows pare che questo sistema operativo non possa accedere a nessuna partizione eccetto la prima. Alcune soluzioni a questo problema sono state discusse sulla nostra [mailing list](#), ma non sembrano esserci risposte semplici.

Remember: Every time you install a new live-image-i386.hybrid.iso 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.

4.5.4 Avviare il supporto live

La prima volta che si avvia il supporto live, CD, DVD, penna USB o PXE, può essere necessario impostare il BIOS del computer,

ma giacché questi variano parecchio in opzioni e scorciatoie, non siamo in grado di descriverli. Alcuni BIOS offrono un menu per selezionare il device in fase di boot, in caso sia disponibile nel vostro sistema è il modo più semplice. Altrimenti è necessario accedere alla sua configurazione e modificare l'ordine di avvio per posizionare la periferica di boot del sistema live prima di quella usuale.

Avviando il supporto si otterrà un menu, premendo il tasto `enter` il sistema partirà utilizzando la voce `Live` e le opzioni predefinite. Per ulteriori informazioni sulle opzioni di boot, si veda la voce “help” nel menu e le pagine di manuale di *live-boot* e *live-config* all'interno del sistema.

Supponendo di aver selezionato `Live` e avviato l'immagine desktop predefinita, dopo i messaggi di avvio si dovrebbe automaticamente accedere all'account `user` e avere il desktop pronto all'uso. Se invece si è avviata un'immagine per la sola console, come le [immagini precompilate](#) standard o `rescue`, si accederà alla console dell'account `user` ed avere un prompt di shell pronto da usare.

4.6 Utilizzare una macchina virtuale per le prove

Per lo sviluppo delle immagini live, può essere un notevole risparmio di tempo eseguirle in una macchina virtuale (VM). Non senza qualche raccomandazione:

- Eseguire una VM richiede un quantitativo sufficiente di RAM sia per il sistema ospitato che per quello ospitante; è consigliato un processore che gestisca la virtualizzazione a livello hardware.
- Ci sono alcune limitazioni inerenti, quali uno scarso rendimento video e una scelta limitata di hardware emulato.
- Quando si sviluppa per un hardware specifico non vi è alcun

sostituto migliore del proprio hardware.

226

- Occasionalmente possono esserci dei bug relativi al solo utilizzo di una VM. Nel dubbio si provi l'immagine direttamente sul proprio hardware.

A condizione che si possa lavorare entro questi vincoli, cercare il software disponibile per la virtualizzazione e scegliere quello adatto alle proprie necessità.

4.6.1 Provare un'immagine ISO con QEMU

Il programma più versatile in Debian è QEMU. Se il processore gestisce la virtualizzazione hardware utilizzare il pacchetto *qemu-kvm*; la descrizione elenca brevemente i requisiti.

Per prima cosa installare *qemu-kvm* o altrimenti *qemu*, nel qual caso il nome del programma nei successivi sarà *qemu* invece di *kvm*. Il pacchetto *qemu-utils* è inoltre utile per creare immagini di dischi virtuali con *qemu-img*.

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

Avviare un'immagine ISO è semplice:

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

Per maggiori dettagli si vedano le pagine di manuale.

4.6.2 Testing an ISO image with VirtualBox

Per provare la ISO con *virtualbox*:

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

Create a new virtual machine, change the storage settings to use *live-image-i386.hybrid.iso* as the CD/DVD device, and start the machine.

Nota: per sistemi live contenenti X.org che si vogliono provare con *virtualbox*, si può voler includere il pacchetto dei driver per X.org di VirtualBox, *virtualbox-guest-dkms* e *virtualbox-guest-x11*, nella configurazione di *live-build*. In caso contrario la risoluzione è limitata a 800x600.

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

Per far funzionare il pacchetto *dkms* vanno anche installati gli header per il kernel utilizzato nell'immagine. Aniché indicare manualmente il pacchetto *linux-headers* adeguato nell'elenco dei pacchetti creato prima, la selezione può essere fatta automaticamente da *live-build*.

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

4.7 Creare e utilizzare un'immagine HDD

Building an HDD image is similar to an ISO hybrid one in all respects except you specify *-b hdd* and the resulting filename

is `live-image-i386.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, you need an HDD image.

Nota: se si è creata un'immagine ISO ibrida con gli esempi precedenti, occorre pulire la directory di lavoro con il comando `lb clean` (vedere <Il comando `lb clean`>):

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

Eseguire il comando `lb config` come prima, questa volta specificando però il tipo di immagine HDD:

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

Creare ora l'immagine con il comando `lb build`:

```
# lb build
```

When the build finishes, a `live-image-i386.img` file should be present in the current directory.

The generated binary image contains a VFAT partition and the syslinux bootloader, ready to be directly written on a USB device. Once again, using an HDD image is just like using an ISO hybrid one on USB. Follow the instructions in <Using an ISO hybrid live

image>, except use the filename `live-image-i386.img` instead of `live-image-i386.hybrid.iso`.

Likewise, to test an HDD image with Qemu, 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 `live-image-i386.img` as the first hard drive.

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

4.8 Creare un'immagine netboot

La seguente sequenza di comandi creerà un'immagine netboot di base contenente un sistema live predefinito senza X.org. È adatta per il boot tramite rete.

Nota: se qualcuno tra gli esempi precedenti è stato seguito, bisogna pulire la directory di lavoro con il comando `lb clean`:

```
# lb clean
```

In this specific case, a `lb clean --binary` would not be enough to clean up the necessary stages. The cause for this is that in netboot setups, a different initramfs configuration needs to be used which *live-build* performs automatically when building netboot images. Since the initramfs creation belongs to the chroot stage, switching to netboot in an existing build directory means to rebuild the chroot stage too. Therefore, `lb clean` (which will remove the chroot stage, too) needs to be used.

Per configurare l'immagine per l'avvio da rete, eseguire il comando

lb config come segue:

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

Diversamente dalle immagini ISO e HDD, il boot via rete non fornisce un'immagine del filesystem al client, perciò i file devono essere forniti via NFS. Con lb config si possono scegliere filesystem di rete differenti. Le opzioni --net-root-path e --net-root-server specificano, rispettivamente, il percorso e il server del server NFS dove l'immagine del filesystem sarà situata all'avvio. Accertarsi che questi siano impostati su valori adeguati alla propria rete.

Creare ora l'immagine con il comando lb build:

```
# lb build
```

In un avvio tramite rete, il client esegue una piccola parte di software che normalmente risiede sulla EPROM della scheda Ethernet. Questo programma invia una richiesta DHCP per ottenere un indirizzo IP e le informazioni su cosa fare in seguito. In genere il passo successivo è ottenere un bootloader di di livello superiore attraverso il protocollo TFTP. Questi potrebbe essere pxelinux, GRUB, o anche avviare direttamente un sistema operativo come Linux.

For example, if you unpack the generated live-image-i386.netboot.tar 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 tftpbboot/.

We must now configure three services on the server to enable

netbooting: the DHCP server, the TFTP server and the NFS server.

4.8.1 Server DHCP

Si deve configurare il server DHCP della rete per essere sicuri di fornire un indirizzo IP al sistema client che si avvia tramite rete, e notificare la posizione del bootloader PXE.

Ecco un esempio, scritto per un server DHCP ISC isc-dhcp-server nel file di configurazione /etc/dhcp/dhcpd.conf:

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

ddns-update-style none;

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

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

log-facility local7;

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

4.8.2 Server TFTP

Fornisce al sistema il kernel e il ramdisk iniziale in fase di esecuzione.

Si installi il pacchetto *tftpd-hpa*, che mette a disposizione tutti i file contenuti in una directory root, di solito */srv/tftp*. Affinché si possa disporre dei file contenuti in */srv/debian-live/tftpboot*, eseguire il seguente comando come utente root:

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

e inserire la nuova directory del server tftp quando richiesto.

4.8.3 Server NFS

Una volta che il computer ospite ha scaricato e avviato un kernel Linux e caricato il suo initrd, cercherà di montare l'immagine del filesystem Live tramite un server NFS.

Bisogna installare il pacchetto *nfs-kernel-server*.

Quindi, rendere disponibile l'immagine del filesystem via NFS aggiungendo una riga come la seguente in */etc/exports*:

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

e comunicare il nuovo export al server NFS con il seguente comando:

```
# exportfs -rv
```

Configurare questi tre servizi può essere un po' problematico, serve un attimo di pazienza per farli funzionare assieme. Per ulteriori informazioni vedere il wiki syslinux <<http://www.syslinux.org/wiki/>

[index.php/PXELINUX](http://d-i.alioth.debian.org/manual/en.i386/ch04s05.html)> o il manuale del Debian Installer alla sezione per l'avvio TFTP da rete <<http://d-i.alioth.debian.org/manual/en.i386/ch04s05.html>>. Ciò può essere d'aiuto, considerato che il procedimento è molto simile.

4.8.4 Come provare una netboot

La creazione di immagini netboot è resa semplice da *live-build*, ma provare le immagini su una macchina reale può essere davvero dispendioso in termini di tempo.

Per semplificarsi il vita si può usare la virtualizzazione.

4.8.5 Qemu

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

Modificare */etc/qemu-ifup*:

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

Procurarsi o compilare grub-floppy-netboot.

Lanciare qemu con "-net nic,vlan=0 -net tap,vlan=0,ifname=tun0"

4.9 Webbooting

Webbooting is a convenient way of retrieving and booting live

systems using the internet as a means. The requirements for webbooting are very few. On the one hand, you need a medium with a bootloader, an initial ramdisk and a kernel. On the other hand, a web server to store the squashfs files which contain the filesystem.

4.9.1 Getting the webboot files

As usual, you can build the images yourself or use the prebuilt files, which are available on the project's homepage at <http://live-systems.org/>. Using prebuilt images would be handy for doing initial testing until one can fine tune their own needs. If you have built a live image you will find the files needed for webbooting in the build directory under `binary/live/`. The files are called `vmlinuz`, `initrd.img` and `filesystem.squashfs`.

It is also possible to extract those files from an already existing iso image. In order to achieve that, loopback mount the image as follows:

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

The files are to be found under the `live/` directory. In this specific case, it would be `/mnt/live/`. This method has the disadvantage that you need to be root to be able to mount the image. However, it has the advantage that it is easily scriptable and thus, easily automatized.

But undoubtedly, the easiest way of extracting the files from an iso image and uploading it to the web server at the same time, is using the midnight commander or *mc*. If you have the *genisoimage* package installed, the two-pane file manager allows you to browse the contents of an iso file in one pane and upload the files via ftp in

the other pane. Even though this method requires manual work, it does not require root privileges.

4.9.2 Booting webboot images

While some users will prefer virtualization to test webbooting, we refer to real hardware here to match the following possible use case which should only be considered as an example.

In order to boot a webboot image it is enough to have the components mentioned above, i.e. `vmlinuz` and `initrd.img` in a usb stick inside a directory named `live/` and install `syslinux` as bootloader. Then boot from the usb stick and type `fetch=URL/-PATH/TO/FILE` at the boot options. *live-boot* will retrieve the squashfs file and store it into ram. This way, it is possible to use the downloaded compressed filesystem as a regular live system. For example:

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

Use case: You have a web server in which you have stored two squashfs files, one which contains a full desktop, like for example `gnome`, and a rescue one. If you need a graphical environment for one machine, you can plug your usb stick in and webboot the `gnome` image. If you need the rescue tools included in the second type of image, perhaps for another machine, you can webboot the rescue one.

Panoramica degli strumenti

5. Panoramica degli strumenti

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

5.1 Il pacchetto live-build

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

L’idea dietro *live-build* è di essere un’infrastruttura che utilizza una directory di configurazione per automatizzare totalmente e personalizzare tutti gli aspetti della creazione di un’immagine live.

Molti concetti sono simili a quelli applicati per creare pacchetti Debian con *debhelper*:

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

- Gli script sono indipendenti, vale a dire che è sempre sicuro eseguire ogni comando.

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

- **lb config** : Responsible for initializing a Live system configuration directory. See <The **lb config** command> for more information.

- **lb build** : responsabile di iniziare la creazione di un sistema live. Si veda <Il comando **lb**> per maggiori informazioni.

- **lb clean** : responsabile della rimozione di parti della creazione di un sistema live. Si veda <Il comando **lb clean**> per maggiori informazioni.

5.1.1 Il comando lb config

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

Issuing `lb config` without any arguments creates the `config/` subdirectory which is populated with some default settings in configuration files, and two skeleton trees named `auto/` and `local/`.

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

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

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

315

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

316

È possibile specificare molte opzioni, come:

317

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

318

Una lista completa delle opzioni è disponibile nel manuale di `lb_config`.

319

5.1.2 Il comando `lb build`

320

Il comando `lb build` legge la configurazione dalla directory `config/` ed esegue a un livello inferiore i comandi necessari a costruire il sistema live.

321

5.1.3 Il comando `lb clean`

322

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

323

5.2 Il pacchetto `live-boot`

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

324

All'avvio cercherà supporti in sola lettura che contengano una directory `/live/` dove sia presente un filesystem root (spesso un'immagine compressa come *squashfs*). Se trovata, creerà un ambiente scrivibile usando *aufs*, per avviarsi da sistemi simili a Debian.

325

Si possono trovare maggiori informazioni sui ramfs iniziali nel capitolo su *initramfs* del Debian Linux Kernel Handbook all'indirizzo <http://kernel-handbook.alioth.debian.org/>.

326

5.3 Il pacchetto `live-config`

327

live-config è costituito da script eseguiti all'avvio dopo *live-boot* per configurare automaticamente il sistema live. Gestisce attività quali impostare l'*hostname*, localizzazione e fuso orario, creare l'utente live, inibire compiti automatizzati tramite cron ed eseguire il login automatico dell'utente live.

328

Gestire una configurazione

6. Gestire una configurazione

Questo capitolo spiega come gestire una configurazione per una live sin dalla creazione iniziale, attraverso le successive revisioni e rilasci sia del software *live-build* che della stessa immagine.

6.1 Gestire i cambiamenti di configurazione

Le configurazioni live sono di rado perfette al primo tentativo. Può andar bene passare le opzioni di `lb config` a riga di comando per eseguire una compilazione ma è tipico rivedere queste opzioni e compilare finché non si è soddisfatti. Per gestire le modifiche c'è bisogno di script automatici che assicurano che la propria configurazione sia coerente.

6.1.1 Perché utilizzare gli script automatici? Cosa fanno?

The `lb config` command stores the options you pass to it in `config/*` files along with many other options set to default values. If you run `lb config` again, it will not reset any option that was defaulted based on your initial options. So, for example, if you run `lb config` again with a new value for `--binary-images`, any dependent options that were defaulted for the old image type may no longer work with the new ones. Nor are these files intended to be read or edited. They store values for over a hundred options, so nobody, let alone yourself, will be able to see in these which options you actually specified. And finally, if you run `lb config`, then upgrade *live-build* and it happens to rename an option, `config/*` would still contain variables named after the old option that are no longer valid.

Per queste ragioni gli script nella directory `auto/*` faciliteranno il

lavoro; sono semplici wrapper ai comandi `lb config`, `lb build` e `lb clean` designati per aiutare a gestire la configurazione. Gli script in `auto/config` memorizzano i comandi di `lb config` con le opzioni desiderate, quelli in `auto/clean` rimuovono i file contenenti i valori delle variabili di configurazione, mentre gli script in `auto/build` tengono un `build.log` di ogni compilazione. Ognuno di questi script viene eseguito automaticamente ogni qualvolta si esegue il comando `lb` corrispondente; utilizzandoli la vostra configurazione sarà più semplice da leggere e verrà mantenuta coerente da una revisione all'altra. Inoltre sarà molto più facile identificare e sistemare le opzioni che necessitano di modifiche quando si aggiorna *live-build* dopo aver letto la documentazione aggiornata.

6.1.2 Esempi d'uso di script automatici

Per comodità *live-build* è fornito di esempi di script automatici da copiare e modificare. Inizializzare una nuova configurazione predefinita quindi copiare gli esempi in essa:

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

Modificare `auto/config` aggiungendo qualsiasi opzione vi serva, esempio:

```
#!/bin/sh
lb config noauto \
  --architectures i386 \
  --linux-flavours 686-pae \
  --binary-images hdd \
```

```
--mirror-bootstrap http://ftp.ch.debian.org/debian/ \
--mirror-binary http://ftp.ch.debian.org/debian/ \
"${@}"
```

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

Ogni volta che verrà usato `lb config`, `auto/config` ripristinerà la configurazione in base a queste opzioni; quando si vogliono apportare modifiche basterà modificare le opzioni in questo file invece di passarle a `lb config`. Utilizzando `lb clean`, `auto/clean` pulirà i file in `config/*` insieme a qualsiasi altro creato dalla compilazione. Infine, quando si usa `lb build`, verrà scritto da `auto/build` un file di log della compilazione in `build.log`.

Nota: il parametro speciale `noauto` viene qui usato per impedire un'ulteriore chiamata di `auto/config`, impedendo quindi infinite chiamate ricorsive; assicurarsi di non rimuoverlo facendo modifiche. Quando si dividono comandi lunghi di `lb config` su più righe per agevolarne la leggibilità, non dimenticare il backslash (alla fine di ogni riga che continua sulla successiva, come mostrato poc'anzi nell'esempio di script.

Modificare `auto/config` e qualsiasi altro file presente in `config` necessario alle proprie esigenze. Ad esempio, le immagini non-free precompilate non ufficiali sono create semplicemente aggiungendo `--archive-areas "main contrib non-free"`.

È possibile definire una scorciatoia nella configurazione di Git aggiungendo quanto segue al file `/${HOME}/.gitconfig`:

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

This enables you to use `lso`: anywhere you need to specify the address of a `live-systems.org` git repository. If you also drop the optional `.git` suffix, starting a new image using this configuration is as easy as:

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

Clonando l'intero repository `live-images` si ottengono configurazioni usate per svariate immagini. Se dopo aver terminato la prima si vuole creare un'immagine differente, basterà cambiare directory e opzionalmente fare di nuovo le modifiche necessarie alle proprie esigenze.

In ogni caso ricordarsi che ogni volta si dovrà creare l'immagine come utente `root`: `lb build`

6.2 Clonare una configurazione pubblicata tramite Git.

Use the `lb config --config` option to clone a Git repository that contains a live system configuration. If you would like to base your configuration on one maintained by the Live Systems Project, look at <http://live-systems.org/gitweb/> for the repository named `live-images` in the category `Packages`. This repository contains the configurations for the live systems **prebuilt images**.

Ad esempio, per creare un'immagine d'emergenza usare il repository `live-images` come segue:

Personalizzazione dei contenuti

7. Panoramica sulla personalizzazione

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

7.1 Configurazione in fase di compilazione e di avvio

La configurazione del sistema live è divisa in opzioni applicate in fase di compilazione e al momento dell'avvio. Le opzioni di compilazione sono ulteriormente divise in quelle che si verificano prima dell'avvio, applicate dal pacchetto *live-boot*, e quelle dopo l'avvio, applicate da *live-config*. Qualsiasi opzione in fase di avvio può essere modificata dall'utente specificandola al prompt di avvio. L'immagine può inoltre essere costruita con i parametri di avvio predefiniti in modo che quando tutti i valori predefiniti sono adatti gli utenti possano avviare direttamente il sistema senza specificare alcuna opzione. In particolare, l'argomento di `lb --bootappend-live` è costituito da tutte le opzioni da riga di comando del kernel predefinite in un sistema live, come persistenza dei dati, layout di tastiera o fuso orario. Per gli esempi si veda [Personalizzare localizzazione e lingua](#).

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.

7.2 Fasi della creazione

Il processo di creazione è diviso in due fasi, con varie personalizzazioni applicate in sequenza a ciascuna di esse. La prima consiste nell'**avvio**, questa è la fase iniziale di popolamento della directory di chroot con i pacchetti atti a creare un sistema Debian di base. Viene quindi seguita dalla fase **chroot** che completa la costruzione della directory chroot e la popola con tutti i pacchetti elencati nella configurazione insieme a qualsiasi altro materiale; la maggior parte della personalizzazione dei contenuti avviene in questa tappa. La parte finale della preparazione dell'immagine è la fase **binaria** che genera un'immagine avviabile utilizzando i contenuti della directory chroot per costruire il file system principale per il sistema live, includere l'installatore e ogni altro materiale aggiuntivo sul supporto di destinazione al di fuori del file system del sistema live. Una volta che l'immagine è pronta viene creato, se abilitato, l'archivio dei sorgenti nella fase **sorgenti**.

All'interno di ciascuna di queste fasi c'è una sequenza particolare in cui vengono applicati i comandi, sono organizzati in modo da assicurare che le personalizzazioni siano ragionevolmente stratificate. Ad esempio, nella fase **chroot** i preseed vengono applicati prima che qualsiasi pacchetto sia installato, i pacchetti vengono installati prima che qualsiasi file incluso localmente venga copiato e gli hook eseguiti dopo che tutto il materiale è a posto.

7.3 Integrare la configurazione di lb con dei file

Although `lb config` creates 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.

366

7.4 Personalizzazione dei compiti

367

I capitoli seguenti sono costituiti dai tipi di compito personalizzato che gli utenti eseguono solitamente: `<personalizzare l'installazione dei pacchetti>`, `<personalizzare i contenuti>` e `<personalizzare localizzazione e lingua>` coprono solo alcune delle cose che si potrebbero desiderare.

Personalizzare l'installazione dei pacchetti

8. Personalizzare l'installazione dei pacchetti

Perhaps the most basic customization of a 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 lists of packages, including metapackages which will install many related packages at once, such as packages for a particular desktop or language. 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 Sorgenti dei pacchetti

8.1.1 Distribuzione, le aree di archivio e le modalità

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 **jessie** for the **jessie** version of *live-build*. Any current distribution carried in the 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:

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

All'interno dell'archivio dei pacchetti, le aree sono le principali divisioni dello stesso. In Debian queste sono `main`, `contrib` e `non-free`; soltanto `main` contiene il software che è parte di Debian, perciò questa è la predefinita. Possono essere specificati uno o più valori:

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

Attraverso l'opzione `--mode` è disponibile un supporto sperimentale per alcune derivate di Debian; per impostazione predefinita, questa opzione è impostata su `debian` solo se si sta costruendo su un sistema Debian o sconosciuto. Invocando `lb config` su una delle derivate supportate, verrà creata un'immagine di quella derivata in modo predefinito. Se `lb config` viene ad esempio eseguito in modalità `ubuntu`, saranno gestiti i nomi della distribuzione e le aree di archivio per la derivata specificata e non quelli di Debian. La modalità cambia anche il comportamento di *live-build* per adattarlo alle derivate.

Note: The projects for whom these modes were added are primarily responsible for supporting users of these options. The Live Systems 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.

8.1.2 Mirror delle distribuzioni

L'archivio Debian è replicato attraverso una vasta rete di mirror in tutto il mondo cosicché chiunque in ogni nazione può selezionare il mirror più vicino per una migliore velocità di scaricamento. Ciascuna delle opzioni `--mirror-*` determina quale mirror della distribuzione è usato nei vari stadi della compilazione. Ricordando dalle **«Fasi della creazione»** che la fase di **avvio** è quando il chroot è inizialmente popolato da *debootstrap* con un sistema minimale e quella di **chroot** è quando viene creato il chroot usato per costruire il file system del sistema live. Perciò per queste fasi vengono usati i corrispondenti cambi di mirror, e in seguito, nella fase **binaria** vengono usati i valori di `--mirror-binary` e `--mirror-binary-security` sostituendo qualsiasi altro mirror usato nelle fasi iniziali.

8.1.3 Mirror delle distribuzioni usati in fase di compilazione

Per impostare i mirror delle distribuzioni usati in fase di compilazione ad uno locale, è sufficiente impostare `--mirror-bootstrap`, `--mirror-chroot-security` e `--mirror-chroot-backports` come segue.

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

Il mirror chroot, specificato da `--mirror-chroot`, è impostato al valore di `--mirror-bootstrap` in modo predefinito.

8.1.4 Mirror delle distribuzioni usate durante l'esecuzione

Le opzioni `--mirror-binary*` determinano i mirror delle distribuzioni inseriti nell'immagine binaria. Questi possono essere usati per installare pacchetti aggiuntivi mentre il sistema live è in funzione. Le impostazioni predefinite impiegano `http.debian.net`, un servizio che sceglie un mirror geograficamente vicino basandosi sul numero IP dell'utente. Questo è una scelta conveniente quando non si può pronosticare quale sarà il mirror migliore per tutti gli utenti. Oppure si può specificare il proprio valore come mostrato nell'esempio qui sotto. Un'immagine compilata con questa configurazione sarebbe adatta solamente ad utenti di una rete dove sia raggiungibile il "mirror".

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

8.1.5 Repository aggizionali

Si possono aggiungere altri repository, ampliando così la scelta dei pacchetti al di là di quelli disponibili nella distribuzione di destinazione. Questi possono essere, per esempio, pacchetti di backport, sperimentali o personalizzati. Per configurare repository aggiuntivi, creare i file `config/archives/vostro-repository.-list.chroot`, o `config/archives/vostro-repository.list.-binary`. Come per le opzioni `--mirror-*`, queste controlleranno i repository usati nella fase **chroot** quando si compila l'immagine, e nella fase **binaria**, ad esempio per usarli quando il sistema live è avviato.

Per esempio, `config/archives/live.list.chroot` permette di installare pacchetti dal repository snapshot `debian-live` al momento della creazione del sistema live.

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

Se si aggiunge la stessa riga in `config/archives/live.list.binary`, il repository verrà aggiunto alla directory `/etc/apt/sources.list.d/` del sistema live.

Se questi file esistono saranno prelevati automaticamente.

Bisogna inoltre inserire la chiave GPG usata per firmare il repository nei file `config/archives/vostro-repository.key-{binary,chroot}`.

Se si necessita di personalizzare il pinning di APT, le sezioni di APT preferences possono essere inserite nei file `config/archives/mio-repository.pref.{binary,chroot}` e verranno automaticamente aggiunte nella directory `/etc/apt/preferences.d/` del sistema live.

8.2 Scegliere i pacchetti da installare

Ci sono diversi modi per scegliere quali pacchetti *live-build* installerà nell'immagine, coprendo una gamma di esigenze diverse. Si possono richiamare i singoli pacchetti da un elenco, usare i metapacchetti o selezionarli tramite il file `control`. E infine inserire i file dei pacchetti nell'albero `config/`, che ben si adatta a provare pacchetti nuovi o sperimentali prima che siano disponibili in un repository.

8.2.1 Elenchi di pacchetti

Gli elenchi di pacchetti sono un potente mezzo per esprimere quali pacchetti devono essere installati. La sintassi gestisce sezioni condizionali rendendo semplice la creazione di elenchi e adattarli per l'uso in molteplici configurazioni. I nomi dei pacchetti possono inoltre essere inseriti nell'elenco utilizzando script shell in fase di compilazione.

Nota: quando si specifica un pacchetto che non esiste, il comportamento di *live-build* è determinato dalla scelta delle utilità di APT. Per ulteriori dettagli si veda [«Scegliere apt o aptitude»](#).

8.2.2 Usare metapacchetti

Il metodo più semplice per popolare una lista di pacchetti è utilizzare un metapacchetto task mantenuto dalla distribuzione. Ad esempio:

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

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

Tutti i metapacchetti task iniziano per `task-`, un modo per

determinare quali siano disponibili (sebbene possa contenere alcuni falsi positivi che corrispondono al nome ma non sono metapacchetti) è di controllare il nome del pacchetto con:

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

In aggiunta a questi si trovano altri metapacchetti per vari scopi. Alcuni sono dei sottoinsiemi dei pacchetti task generici, come `gnome-core`, mentre altri sono parti individuali di un Debian Pure Blend, come il metapacchetto `education-*`. Per elencarli tutti installare il pacchetto `debtags` e usare il tag `role::metapackage` come segue:

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

8.2.3 Elenchi locali dei pacchetti

Se si richiede l'elenco di metapacchetti, pacchetti individuali o una combinazione di entrambi tutte le liste dei pacchetti locali vengono salvate in `config/package-lists/`. Giacché è possibile usare più di una lista, ciò si presta bene a progetti modulari. Si può ad esempio decidere di dedicare un elenco ad un particolare desktop, un altro ad un insieme di pacchetti correlati utilizzabili con desktop differenti. Questo permette di sperimentare diverse combinazioni di insiemi di pacchetti con il minimo sforzo condividendo gli elenchi tra progetti live differenti.

Per essere processati, gli elenchi dei pacchetti che si trovano in questa directory devono avere un suffisso `.list` e un suffisso `.chroot` o `.binary` aggiuntivo per indicare per quale fase sia

l'elenco.

Nota: se non si specifica il suffisso l'elenco sarà usato per entrambe le fasi. Normalmente è preferibile specificare `.list.chroot` in modo che i pacchetti vengono installati solo nel filesystem live evitando di avere una copia extra del `.deb` sul dispositivo.

8.2.4 Elenchi locali di pacchetti binari

Per creare un elenco di binari inserire un file con suffisso `.list.binary` in `config/package-lists/`; questi pacchetti non sono installati nel filesystem ma inclusi sul dispositivo live sotto `pool/`. Solitamente questo elenco si usa con una delle varianti non-live dell'installatore; come detto sopra, se si vuole che questo sia identico all'elenco della fase `chroot`, usare semplicemente il suffisso `.list`.

8.2.5 Elenchi di pacchetti generati

Talvolta succede che il modo migliore per ottenere un elenco è di generarlo con uno script. Ogni riga che inizia con un punto esclamativo indica un comando da eseguire nel `chroot` quando viene creata l'immagine. Ad esempio si potrebbe includere la riga `! grep-aptavail -n -sPackage -FPriority standard |sort` in una lista di pacchetti per produrne una contenente i pacchetti con `Priority: standard` disponibili.

Infatti selezionare i pacchetti con il comando `grep-aptavail` (presente nel pacchetto `dctrl-tools`) è talmente utile che `live-build` fornisce uno script `Packages` per comodità; accetta due argomenti: `field` e `pattern`. Per cui si può creare un elenco con il seguente contenuto:

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

8.2.6 Usare condizioni all'interno degli elenchi di pacchetti

Ognuna delle variabili di configurazione di *live-build* situate in `config/*` (senza il prefisso `LB_`) possono essere utilizzate per istruzioni condizionali nell'elenco dei pacchetti. In genere questo significa qualsiasi opzione di `lb config` in maiuscolo e con trattini cambiati in trattini bassi; ma in pratica è la sola ad influenzare la selezione dei pacchetti che abbia senso, come `DISTRIBUTION`, `ARCHITECTURES` o `ARCHIVE_AREAS`.

Per esempio, per installare `ia32-libs` se è specificata `--architectures amd64`:

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

Si può provare per ognuna di una serie di valori, ad esempio per installare `memtest86+` specificando sia `--architectures i386` sia `--architectures amd64`:

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

È possibile provare altre variabili che contengano più di un valore,

ad esempio per installare `vrms` specificando sia da `contrib` sia da `non-free` tramite `--archive-areas`:

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

Le condizioni nidificate non sono supportate.

8.2.7 Removing packages at install time

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

8.2.8 Task per desktop e lingua

I task per i desktop e la lingua sono un caso particolare che necessita di ulteriori pianificazioni e configurazioni e in questo senso le immagini live sono diverse da quelle dell'Installatore Debian. Nell'Installatore Debian, se il supporto è stato preparato per un particolare ambiente desktop, il corrispondente task verrà automaticamente installato. Perciò ci sono task `gnome-desktop`, `kde-desktop`, `lxde-desktop` e `xfce-desktop` interni, nessuno dei quali è offerto nel menu di `taskse1`. Allo stesso modo, non c'è

nessuna voce nel menu per i task delle lingue, ma la scelta della lingua dell'utente durante l'installazione influenza la selezione dei corrispondenti task della lingua.

Sviluppando un'immagine live per desktop, questa si avvia direttamente su un'area di lavoro, le scelte del desktop e della lingua predefinita sono state fatte al momento della compilazione e non al volo come nel caso dell'installatore Debian. Questo non per dire che un'immagine live non possa essere creata con un supporto per desktop o lingue multipli per offrire all'utente una scelta, ma che non è il comportamento predefinito nella creazione di una live.

Poiché automaticamente non viene fatta alcuna preparazione sui task della lingua, i quali includono cose come caratteri specifici per la lingua e pacchetti per i metodi di input, se li si vogliono, vanno specificati nella configurazione. Per esempio, un'immagine del desktop GNOME contenente il supporto per il tedesco può includere questi metapacchetti task:

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

8.2.9 Tipi e versioni del kernel

A seconda dell'architettura, nell'immagine verranno inclusi uno o più tipi di kernel in modo predefinito. È possibile scegliere tipi differenti tramite l'opzione `--linux-flavours`, ognuno ha come suffisso `linux-image` che costituisce il nome del metapacchetto che a sua volta dipende dall'esatto pacchetto del kernel da inserire nell'immagine.

Perciò un'immagine con architettura `amd64` includerà il metapacchetto `linux-image-amd64` in modo predefinito, mentre un'immagine con architettura `i386` includerà i metapacchetti `linux-image-486` e `linux-image-686-pae`. Al momento della scrittura del manuale questi pacchetti dipendono rispettivamente da `linux-image-3.2.0-4-amd64`, `linux-image-3.2.0-4-486` e `linux-image-3.2.0-4-686-pae`.

Quando nel proprio archivio è disponibile più di una versione del kernel, si può specificare un suffisso di pacchetto differente con l'opzione `--linux-packages`. Ad esempio, supponendo di creare un'immagine per architettura `amd64` e aggiungere l'archivio `experimental` per fare dei test e poter installare il kernel `linux-image-3.7-trunk-amd64`, si configurerà tale immagine come segue:

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

8.2.10 Kernel personalizzati

Si può compilare e includere i propri kernel personalizzati a patto che siano integrati nel sistema di gestione dei pacchetti di Debian. Il sistema *live-build* non supporta i kernel né crea pacchetti `.deb`.

La maniera corretta e raccomandata per collocare i propri pacchetti è di seguire le istruzioni nel `kernel-handbook`. Ricordarsi di modificare i suffissi per ABI e tipologia in modo appropriato quindi includere una compilazione completa del pacchetto `linux` e del corrispondente `linux-latest` nel repository.

Se si opta per creare i pacchetti del kernel senza i metapacchetti

corrispondenti, bisogna specificare un suffisso `--linux-packages` appropriato come discusso in [«Tipi e versioni del kernel»](#). Come spiegato in [«Installare pacchetti modificati o di terze parti»](#), è meglio includere i propri pacchetti del kernel nel proprio repository, sebbene funzionino anche le alternative discusse in tale sezione.

Fornire suggerimenti sul come personalizzare il proprio kernel va oltre lo scopo di questa documentazione, tuttavia è necessario assicurarsi che la configurazione soddisfi almeno i seguenti requisiti minimi:

- Utilizzare un ramdisk iniziale;
- includere il modulo del filesystem union (solitamente aufs);
- includere qualsiasi altro modulo del filesystem necessario alla configurazione (solitamente squashfs).

8.3 Installare pacchetti modificati o di terze parti

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

Questa sezione non tratta la compilazione e il mantenimento di pacchetti modificati. Può comunque essere interessante leggere “How to fork privately” di Joachim Breitner: <http://www.joachim-breitner.de/blog/archives/282-How-to-fork-privately.html> La creazione di pacchetti su misura è esposta nella “Guida per il nuovo Maintainer” all’indirizzo <https://www.debian.org/doc/maint-guide/> e altrove.

Ci sono due modi per installare pacchetti personalizzati:

- `packages.chroot` 452
- utilizzare repository APT personalizzati 453

Usando `packages.chroot` è più semplice da ottenere e utile per una personalizzazione “una tantum” ma ha una serie di svantaggi, mentre un repository APT personalizzato è più laborioso da configurare. 454

8.3.1 Utilizzare `packages.chroot` per installare pacchetti personalizzati 455

Per installare un pacchetto personalizzato copiarlo nella directory `config/packages.chroot/`; i pacchetti al suo interno verranno installati automaticamente durante la creazione del sistema live, non è necessario specificarli altrove. 456

I pacchetti **devono** essere nominati nel modo prescritto, un metodo semplice per farlo è usare `dpkg-name`. 457

L'utilizzo di `packages.chroot` per l'installazione di pacchetti personalizzati presenta degli svantaggi: 458

- non è possibile usare secure APT, 459
- è necessario installare i pacchetti adeguati nella directory `config/packages.chroot/`, 460
- It does not lend itself to storing live system configurations in revision control. 461

8.3.2 Utilizzare un repository APT per installare pacchetti personalizzati 462

A differenza di `packages.chroot`, quando si usa un repository APT personalizzato è necessario assicurarsi di specificare 463

altrove i pacchetti. Per i dettagli si veda [«Scegliere i pacchetti da installare»](#). 470

464 Sebbene creare un repository APT possa sembrare uno sforzo inutile, l'infrastruttura può facilmente essere riutilizzata in un secondo momento per offrire aggiornamenti dei pacchetti modificati.

465 8.3.3 Pacchetti personalizzati e APT

466 *live-build* utilizza APT per installare tutti i pacchetti nel sistema live in modo da ereditare i comportamenti di questo programma. Un esempio rilevante è che (considerando una configurazione predefinita) dato un pacchetto disponibile in due repository differenti con numeri di versione diversi, APT sceglie di installare quello con il numero di versione più alto.

467 A causa di questo si può voler incrementare il numero della versione nei file `debian/changelog` dei pacchetti personalizzati per accertare che la propria versione avrà la precedenza sui repository Debian ufficiali. È anche ottenibile modificando le preferenze del APT pinning del sistema live, si veda [«APT pinning»](#) per maggiori informazioni.

468 8.4 Configurare APT in fase di compilazione

469 APT è configurabile tramite una serie di opzioni applicate solo in fase di costruzione (la configurazione di APT utilizzata nel sistema live in esecuzione può essere configurata nel solito modo, ovvero includendo le impostazioni appropriate attraverso `config/includes.chroot/`). Per un elenco completo, cercare nel manuale di `lb_config` le opzioni che iniziano con `apt`.

8.4.1 Scegliere apt o aptitude

471 Per installare pacchetti in fase di compilazione si può optare sia per *apt* sia per *aptitude*, l'argomento `--apt` di `lb config` determina quale usare. Sceglie il metodo implementando il comportamento preferito per l'installazione dei pacchetti, la notevole differenza è come vengono gestiti quelli mancanti.

- 472 • `apt`: se viene specificato un pacchetto mancante, l'installazione avrà esito negativo; questo è l'impostazione predefinita.
- 473 • `aptitude`: se viene specificato un pacchetto mancante, l'installazione avrà successo.

474 8.4.2 Utilizzare un proxy con APT

475 Una configurazione di APT spesso richiede di amministrare la creazione di un'immagine dietro un proxy, lo si può specificare con le opzioni `--apt-ftp-proxy` o `--apt-http-proxy` secondo necessità:

```
476 $ lb config --apt-http-proxy http://proxy/
```

477 8.4.3 Modificare APT per risparmiare spazio

478 Si può aver bisogno di risparmiare dello spazio sul supporto dell'immagine, in tal caso una o entrambe delle seguenti opzioni possono essere d'interesse.

479 È possibile non includere gli indici di APT con:

480

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

481 Questo non influenzerà le voci in `/etc/apt/sources.list`,
determina solo se `#{var/lib/apt}#` contiene o meno i file degli indici.
Il compromesso è che APT necessita di quegli indici per operar
enel sistema live, perciò prima di eseguire `apt-cache search` o
`apt-get install`, per esempio, l'utente deve usare prima `apt-get`
`update` per crearli.

482 In caso si trovi che l'installazione dei pacchetti raccomandati
appesantisca troppo l'immagine, a patto si è preparati ad
affrontare le conseguenze discusse prima, si può disabilitare
l'opzione predefinita di APT con:

483

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

484 La conseguenza più importante di disattivare i raccomandati è
che `live-boot` e `live-config` raccomandano a loro volta alcuni
pacchetti che forniscono funzionalità importanti utilizzate da molte
configurazioni, come `user-setup` che `live-config` raccomanda
ed è usato per creare l'utente live. Salvo eccezioni ci sarà bisogno
di riaggiungere all'elenco almeno alcuni di questi o l'immagine
non funzionerà come ci si aspetta. Controllare i raccomandati per
ognuno dei pacchetti `live-*` inclusi nella compilazione, se non si è
certi di poterli omettere aggiungerli nuovamente agli elenchi.

485 La conseguenza generica è che se non si installano i raccomandati
per un certo pacchetto, ovvero "pacchetti che si trovano assieme
a questo eccetto in installazioni non usuali" (Debian Policy
Manual, paragrafo 7.2), saranno omessi alcuni di quelli realmente
necessari. Si suggerisce pertanto di verificare la differenza ottenuta
nel proprio elenco di pacchetti disabilitando i raccomandati
(vedere il file `binary.packages` generato da `lb build`) e includere

nuovamente in esso quelli omessi che si desiderano installare.
In alternativa, se si desidera tenere un modesto numero di
raccomandati, li si lasci abilitati e si assegni ad APT un pin di
priorità negativo sui pacchetti selezionati affinché non vengano
installati, come spiegato in [«APT pinning»](#).

8.4.4 Passare opzioni ad apt o aptitude

486

Se non esiste un'opzione di `lb config` per modificare il
comportamento di APT come si desidera, utilizzare `--apt-`
`options` o `--aptitude-options` per passare qualsiasi argomento
tramite lo strumento APT scelto. Per i dettagli consultare le
pagine di manuale di `apt` e `aptitude`. Notare che entrambe le
opzioni hanno valori predefiniti che servirà mantenere in aggiunta
a qualsiasi altra fornita. Per cui supponendo di aver incluso
qualcosa da `snapshot.debian.org` per fare dei test e volendo
specificare `Acquire::Check-Valid-Until=false` per soddisfare
APT con il vecchio file `Release`, si procederà come nell'esempio
riportato di seguito, appendendo la nuova opzione al valore
predefinito `--yes`:

488

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

Per apprendere a pieno queste opzioni e sapere quando usarle
consultare i manuali. Questo è solo un esempio e non va
interpretato come il modo per configurare la propria immagine,
non sarebbe appropriato per il rilascio finale.

489

Per configurazioni di APT più complesse che comportano l'uso di
opzioni in `apt.conf` si può voler creare invece il file `config/apt/-`
`apt.conf`. Vedere anche le altre opzioni `apt-*` per alcune comode
scorciatoie di operazioni di uso frequente.

490

8.4.5 APT pinning

Si prega di leggere prima il manuale di `apt_preferences(5)`. Il pinning può essere configurato sia in fase di costruzione sia di esecuzione; per la prima creare `config/archives/*.pref`, `config/archives/*.pref.chroot`, e `config/apt/preferences` mentre per l'ultima creare `config/includes.chroot/etc/apt/preferences`.

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

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

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

Un valore negativo della priorità evita che un pacchetto venga installato, come nel caso in cui non se ne voglia uno raccomandato da un altro. Supponiamo di costruire un'immagine di LXDE utilizzando l'opzione `task-lxde-desktop` in `config/package-lists/desktop.list.chroot` ma non si desidera che all'utente venga richiesto di

salvare la password del wifi nel portachiavi. Questo metapacchetto dipende da `lxde-core` che raccomanda `gksu` e che a sua volta raccomanda `gnome-keyring`, in questo caso si vorrà omettere il pacchetto `gnome-keyring` aggiungendo a `#{config/apt/preferences}` la seguente istruzione:

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

496

Personalizzazione dei contenuti

9. Personalizzazione dei contenuti

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 live system 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.

9.1 Include

While ideally a live system would include files entirely provided by unmodified 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 live system image. *live-build* provides two mechanisms for using them:

- Include locali del chroot: permettono di aggiungere o sostituire file al file system chroot/Live. Vedere [«Live/chroot include locali»](#) per maggiori informazioni.
- Include locali binari: permettono di aggiungere o sostituire file nell'immagine binaria. Vedere [«Include locali binari»](#) per maggiori informazioni

Si consulti il [«Glossario»](#) per ulteriori informazioni sulla distinzione tra immagini “Live” e “binarie”.

9.1.1 Live/chroot include locali

Gli include locali del chroot possono essere usati per aggiungere o sostituire file nel filesystem chroot/Live in modo che possano

essere utilizzati nel sistema live. Un utilizzo tipico è popolare la directory scheletro dell'utente (/etc/skel) che il sistema impiega per creare la home dell'utente. Un altro è quello di fornire file di configurazione che possono essere semplicemente aggiunti o sostituiti nell'immagine senza elaborazione; si veda [«Live/chroot hook locali»](#) se è necessaria l'elaborazione.

Per includere i file si aggiungano semplicemente alla directory config/includes.chroot. Questa corrisponde alla directory root / del sistema live. Per esempio, per aggiungere un file /var/www/index.html nel sistema live, si usi:

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

La configurazione avrà quindi il seguente schema:

```
-- config
[...]
```

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

Gli include locali del chroot vengono installati dopo l'installazione dei pacchetti in modo che tali file vengano in seguito sovrascritti.

9.1.2 Include locali binari

Si possono utilizzare include locali binari per inserire sul filesystem del supporto materiale come documentazione o video affinché sia

immediatamente accessibile dopo l'inserimento dello stesso senza avviare il sistema live. Ciò funziona in modo simile agli include locali del chroot; supponendo che i file `~/video_demo.*` siano video dimostrativi del sistema descritti da e collegati a una pagina HTML indice, basta copiare il materiale in `config/includes.binary/` come segue:

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

Questi file appariranno nella directory principale del supporto live.

9.2 Hook

Gli hook permettono di eseguire comandi nel chroot e nelle fasi binarie della creazione al fine di personalizzare l'immagine.

9.2.1 Live/chroot hook locali

To run commands in the chroot stage, create a hook script with a `.hook.chroot` suffix containing the commands in the `config/hooks/` directory. The hook will run in the chroot after the rest of your chroot configuration has been applied, so remember to ensure your configuration 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/doc/live-build/examples/hooks` which you can copy or symlink to use them in your own configuration.

9.2.2 Hook in fase di avvio

Per eseguire comandi all'avvio, è possibile fornire degli hook a *live-config* come spiegato nella sezione "Customization" del suo manuale. Controllare gli hook di *live-config* in `/lib/live/config/` e notare i numeri sequenziali; fornire quindi i propri hook con una sequenza numerica appropriata, sia come include locali del chroot in `config/includes.chroot/lib/live/config/`, sia come pacchetto personalizzato come discusso in [Installare pacchetti modificati o di terze parti](#).

9.2.3 Hook binari locali

To run commands in the binary stage, create a hook script with a `.hook.binary` suffix containing the commands in the `config/hooks/` directory. 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 various common binary customization tasks provided in `/usr/share/doc/live-build/examples/hooks` which you can copy or symlink to use them in your own configuration.

9.3 Preconfigurare le domande di Debconf

I file nella directory `config/preseed/` con suffisso `.cfg` seguiti dalla fase (`.chroot` o `.binary`) sono considerati file di preconfigurazione di debconf e sono installati da *live-build* usando `debconf-set-selections` durante la fase corrispondente.

Per ulteriori informazioni su debconf, vedere `debconf(7)` nel

pacchetto *debconf*.

Personalizzare i comportamenti durante l'esecuzione

10. Personalizzare i comportamenti durante l'esecuzione

Tutte le configurazioni durante l'esecuzione sono eseguite da *live-config*. Vengono qui presentate alcune delle opzioni di *live-config* più comuni alle quali gli utenti sono interessati; una lista completa può essere trovata nel suo manuale.

10.1 Personalizzare l'utente live

Un'importante considerazione è che l'utente live viene creato all'avvio da *live-boot* e non da *live-build* durante la compilazione. Questo non solo influenza dove viene introdotto il materiale relativo all'utente nella creazione, come discusso in [«Live/chroot include locali»](#), ma anche ogni gruppo e permesso associato all'utente live.

È possibile specificare gruppi aggiuntivi ai quali l'utente live apparterrà utilizzando una delle possibilità di configurazione di *live-config*. Ad esempio, per aggiungere l'utente al gruppo *fuse*, è possibile sia inserire in `config/includes.chroot/etc/live/-config/user-setup.conf` quanto segue:

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

o utilizzare `live-config.user-default-groups=audio,cdrom,dip,floppy,video,plugdev,netdev,powerdev,scanner,bluetooth,fuse` come parametro di boot.

È inoltre possibile modificare facilmente il nome utente "user" e la

password "live" predefiniti.

Per cambiare il nome utente specificare quanto segue nella configurazione:

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

Un modo per cambiare la password è tramite un hook come descritto in [«Hook in fase di avvio»](#). Si può usare l'hook "passwd" da `/usr/share/doc/live-config/examples/hooks`, antepoendolo di conseguenza (ad esempio, 2000-passwd) e aggiungerlo al file `config/includes.chroot/lib/live/config/`

10.2 Personalizzare la localizzazione e la lingua

Quando il sistema live si avvia, la lingua è inserita in due fasi:

- generazione della localizzazione

impostare la configurazione della tastiera

Quando si crea un sistema live la localizzazione predefinita è `locales=en_US.UTF-8`. Per definire quale generare, si usi il parametro `locales` nell'opzione `--bootappend-live` di `lb config`:

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

Possono essere specificate più lingue separate da una virgola, `locales=en_US.UTF-8,de_CH.UTF-8`.

Questo parametro, così come quelli della tastiera indicati più avanti, può essere usato anche dalla riga di comando del

kernel specificando una lingua con `language_country` (nel qual caso verrà usata la codifica predefinita) o l'intera stringa `language_country.encoding`. In `/usr/share/i18n/SUPPORTED` è possibile trovare un elenco delle lingue supportate e la codifica per ognuna di esse.

Sia la configurazione della tastiera in console sia di X sono eseguite da `live-config` con il pacchetto `console-setup`. Per fare ciò usare i parametri `keyboard-layouts`, `keyboard-variants`, `keyboard-options` e `keyboard-model` tramite l'opzione `--bootappend-live`. Le opzioni valide si trovano in `/usr/share/X11/xkb/rules/base.lst`. Per ottenere i layout e le varianti di una data lingua, provare a cercare il loro nome inglese o il paese in cui è usata, esempio:

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

Notare che ogni variante mostra nella descrizione il layout alla quale viene applicata.

Spesso c'è bisogno di configurare solo il layout. Ad esempio per ottenere i file di localizzazione per il layout di tastiera tedesco e svizzero-tedesco in X:

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

Tuttavia per casi molto particolari si vorrà includere altri parametri. Ad esempio per configurare un sistema in francese con un layout Dvorak (chiamato Bepo) su una tastiera USB TypeMatrix EZ-Reach 2030:

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

Per ogni opzione `keyboard-*` si possono specificare più valori separati da una virgola, con l'eccezione di `keyboard-model` che ne accetta uno solo. Consultare la pagina di manuale di `keyboard(5)` per dettagli ed esempi delle variabili `XKBMODEL`, `XKBLAYOUT`, `XKBVARIANT` e `XKBOPTIONS`. Se vengono forniti più valori per `keyboard-variants`, questi verranno combinati uno ad uno con quelli di `keyboard-layouts` (vedere l'opzione `-variant` in `setxkbmap(1)`). Sono permessi valori vuoti, ad esempio per definire due layout, US QWERTY come predefinito e US Dvorak, usare:

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

10.3 Persistenza

Uno dei paradigmi di un cd live è un sistema preinstallato eseguito da un supporto in sola lettura, come un cdrom, dove le modifiche non sopravvivono ai riavvii dell'hardware della macchina ospitante.

Un sistema live è una generalizzazione di questo paradigma e di conseguenza oltre ai CD gestisce altri supporti; ma comunque, nel suo comportamento predefinito, deve essere considerato in sola lettura e tutte i cambiamenti fatti durante l'esecuzione del sistema verranno persi allo spegnimento.

Persistenza è il nome comune per differenti tipi di soluzioni per salvare alcune o tutte queste modifiche con i riavvii. Per capire come funziona potrebbe essere utile sapere che sebbene il sistema venga avviato ed eseguito da un dispositivo in sola lettura, le modifiche a file e directory vengono scritte su uno scrivibile, tipicamente un ram disk (tmpfs) e i dati sui ram disk non sopravvivono ai riavvii.

I dati immagazzinati su questo ramdisk andrebbero salvati un supporto scrivibile persistente come un supporto di memorizzazione locale, una condivisione di rete o anche una sessione di un CD/DVD riscrivibile multisessione. Tutti questi supporti sono gestiti in modi differenti e tutti tranne l'ultimo richiedono un parametro d'avvio speciale da specificare all'avvio: persistence.

Se il parametro di boot persistence è impostato (e non lo è nopersistence), i supporti di memorizzazione locali (hard disk, dispositivi USB) saranno rilevati come volumi persistenti durante l'avvio. È possibile selezionare quali tipi utilizzare specificando certi parametri di avvio descritti nella manpage di *live-boot(7)*. Un volume persistente è uno dei seguenti:

- una partizione, identificata dal suo nome GPT (GUID Partition Table).
- un filesystem, identificato dalla sua label.
- un file immagine situato nella directory radice di un qualsiasi filesystem leggibile (anche una partizione NTFS di un sistema estraneo), identificato dal nome del file.

La label del volume per le stratificazioni deve essere persistence ma verrà ignorata a meno che non sia presente nella directory radice un file chiamato persistence.conf che viene usato per personalizzare la persistenza del volume, in altre parole, specificare le directory che si vogliono salvare dopo un riavvio. Per maggiori dettagli vedere <Il file persistence.conf>.

Ecco alcuni esempi per preparare un volume da utilizzare per la persistenza. Può ad esempio essere una partizione ext4 su un hard disk o una penna USB creata con:

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

Vedere anche <Usare lo spazio rimanente su una penna USB>.

Se si possiede già una partizione sul dispositivo basta solo cambiare l'etichetta con una delle seguenti:

```
# tune2fs -L persistence /dev/sdb1 # per filesystem ext2,3,4
```

Un esempio di come creare un file immagine ext4 da utilizzare per la persistenza:

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

Una volta che il file immagine è stato creato, ad esempio per rendere /usr persistente salvando solo le modifiche fatte a quella directory e non tutto il contenuto di /usr, si può usare l'opzione

“union”. Se l'immagine è situata nella propria home copiarla nella radice del filesystem sul disco e montarla in /mnt come segue:

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

Creare quindi il file `persistence.conf` aggiungendovi il contenuto e smontare il file immagine.

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

Ora riavviare il dispositivo live con il parametro d'avvio “persistence”.

10.3.1 Il file `persistence.conf`

Un volume con la label `persistence` deve essere configurato mediante il file `persistence.conf` per creare directory persistenti arbitrarie. Tale file, situato nella directory radice del filesystem del volume, controlla quali rendere persistenti e in che modo.

Nella manpage di `persistence.conf(5)` è descritto dettagliatamente come è configurato il mount degli strati personalizzati, ma un semplice esempio dovrebbe essere sufficiente per la maggior parte degli usi. Supponendo di voler creare la directory `home` e quella della cache di APT in modo persistente in un filesystem `ext4` sulla partizione `/dev/sdb1`:

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

Quindi riavviare. Durante il primo avvio il contenuto di `/home` e `/var/cache/apt` saranno copiati nel volume persistente e da allora tutte le modifiche a queste directory risiederanno in modo persistente sul volume. C'è da considerare che tutti i path elencati nel file `persistence.conf` non possono contenere spazi o i caratteri speciali `.` e `..`, inoltre né `/lib`, `/lib/live` (o una delle sue sottodirectory) né `/` può essere resa persistente tramite i mount personalizzati. Come workaround a questa limitazione è possibile aggiungere `/ union` al file `persistence.conf` file per ottenere la persistenza completa.

10.3.2 Utilizzare più di un'archiviazione persistente

Ci sono tre metodi differenti di utilizzare persistenze multiple per differenti casi d'uso. Ad esempio l'utilizzo di svariati volumi contemporaneamente o selezionandone uno solo per scopi molto specifici.

Possono essere utilizzati svariati volumi di stratificazione personalizzati (con i rispettivi file `persistence.conf`) allo stesso tempo ma se questi creano la stessa directory persistente, ne verrà usata solo una. Se due directory montate sono “nidificate” (una è la sottodirectory dell'altra), la superiore sarà montata per prima, per cui nessuna operazione di mount verrà sovrastata dall'altra. I mount nidificati personalizzati sono problematici se sono elencati nello stesso file `persistence.conf`. Se si ha davvero la necessità (in genere non si dovrebbe averla), consultare la manpage di `persistence.conf(5)` per sapere come gestire questo

caso.

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

Se un utente avesse bisogno di spazi di archiviazione multipli dello stesso tipo per posizioni differenti o per test, come privato e lavoro, il parametro d'avvio persistence-label usato in congiunzione con persistent permetterà supporti persistenti multipli ma univoci. Un esempio potrebbe essere un utente che vuole usare una partizione etichettata come privato per dati personali come i preferiti del browser o di altro tipo, questi userà i parametri d'avvio persistence persistence-label=privato. E per archiviare dati inerenti il lavoro, come documenti, ricerche e altro, verranno usati i parametri d'avvio persistence persistence-label=lavoro.

È importante ricordare che ognuno di questi volumi, privato e lavoro, necessitano anche di un file persistence.conf nella propria radice. Il manuale di *live-boot* contiene altre informazioni su come utilizzare queste etichette con nomi usati in versioni precedenti.

10.4 Using persistence with encryption

Using the persistence feature means that some sensible data might get exposed to risk. Especially if the persistent data is stored on a portable device such as a usb stick or an external hard drive. That is when encryption comes in handy. Even if the entire procedure

might seem complicated because of the number of steps to be taken, it is really easy to handle encrypted partitions with *live-boot*. In order to use **luks**, which is the supported encryption type, you need to install *cryptsetup* both on the machine you are creating the encrypted partition with and also in the live system you are going to use the encrypted persistent partition with.

To install *cryptsetup* on your machine:

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

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

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

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

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

Let's go into the details for all of those who are not familiar with encryption. In the following example we are going to use a partition on a usb stick which corresponds to /dev/sdc2. Please be warned

that you need to determine which partition is the one you are going to use in your specific case.

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

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

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

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

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

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

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

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

To continue with our setup, we need to mount the device, for example in `/mnt`.

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

And create the `persistence.conf` file in the root of the partition. This is, as explained before, strictly necessary. See [‹The persistence.conf file›](#).

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

Then unmount the mount point:

```
# umount /mnt
```

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

```
# cryptsetup luksClose live
```

Let's summarize the process. So far, we have created an encryption capable live system, which can be copied to a usb stick as explained in [‹Copying an ISO hybrid image to a USB stick›](#). We have also created an encrypted partition, which can be located in the same usb stick to carry it around and we have configured the encrypted partition to be used as persistence store. So now, we only need to boot the live system. At boot time, *live-boot* will prompt us for the passphrase and will mount the encrypted partition to be used for persistence.

615 Personalizzare l'immagine binaria

616 11. Personalizzare l'immagine binaria

617 11.1 Bootloader

618 *live-build* usa *syslinux* e alcuni dei suoi derivati (a seconda del tipo di immagine) come bootloader predefiniti. Si possono facilmente personalizzare per soddisfare le proprie esigenze.

619 Per utilizzare un tema completo, copiare `/usr/share/live/-build/bootloaders` in `config/bootloaders` e modificare i file. Se non si vogliono modificare tutte le configurazioni dei bootloader supportati è sufficiente fornire la copia locale di uno di essi, ad esempio **isolinux** in `config/bootloaders/isolinux` può bastare, dipende dalle esigenze.

620 When modifying one of the default themes, if you want to use a personalized background image that will be displayed together with the boot menu, add a `splash.png` picture of 640x480 pixels. Then, remove the `splash.svg` file.

621 Quando si tratta di fare modifiche ci sono varie possibilità. Per esempio i derivati di *syslinux* sono configurati con un timeout impostato a 0 (zero) in modo predefinito, significa che resteranno in pausa al loro splash screen fino a quando non si preme un tasto.

622 Per modificare il timeout di avvio di un'immagine *iso-hybrid* modificare un file **isolinux.cfg** predefinito specificando il timeout in unità di 1/10 di secondo. Un file **isolinux.cfg** modificato per effettuare il boot dopo cinque secondi sarebbe simile a questo:

623

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

11.2 Metadati ISO

624

Quando si crea un'immagine binaria ISO9660, si possono usare le seguenti opzioni per aggiungere vari metadati testuali. Questo può aiutare a identificare facilmente la versione o la configurazione di un'immagine senza avviarla.

625

- `LB_ISO_APPLICATION/--iso-application NAME:` describe l'applicazione che sarà nell'immagine. La lunghezza massima per questo campo è di 128 caratteri.

626

* `LB_ISO_PREPARER/--iso-preparer NAME:` describe il costruttore dell'immagine, solitamente con alcuni dettagli per contattarlo. L'impostazione predefinita è la versione di *live-build* che si sta usando, il quale potrà essere utile in seguito per il debugging. La lunghezza massima per questo campo è di 128 caratteri.

627

- `LB_ISO_PUBLISHER/--iso-publisher NAME:` describe l'editore dell'immagine, solitamente con qualche dettaglio per contattarlo. La lunghezza massima per questo campo è di 128 caratteri.

628

- `LB_ISO_VOLUME/--iso-volume NAME:` specifica l'ID del volume dell'immagine. Questa è utilizzata come etichetta visibile all'utente su alcune piattaforme, come Windows e Apple Mac OS. La lunghezza massima per questo campo è di 128 caratteri.

629

Personalizzare l'Installatore Debian

12. Personalizzare l'Installatore 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.

In questa sezione si presti attenzione all'uso delle lettere maiuscole quando si fa riferimento all'"Installatore Debian", quando usato ci si riferisce esclusivamente all'installatore ufficiale Debian. Spesso è abbreviato come "d-i".

12.1 Tipologie dell'Installatore Debian

I tre principali tipi dell'installer sono:

"Normal" Debian Installer : This is a normal live system 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".

In queste immagini, Debian è installata prendendo e installando i pacchetti .deb usando *debootstrap*, da supporti locali o dalla rete, risultante in un sistema Debian standard installato sul disco rigido.

L'intero processo può essere preimpostato e personalizzato in diversi modi; per ulteriori informazioni si vedano le corrispondenti pagine del manuale dell'Installatore Debian. Una volta che si ha un file preimpostato funzionante, *live-build* può inserirlo automaticamente nell'immagine e abilitarlo.

"Live" Debian Installer : This is a live system image with a separate kernel and initrd which (when selected from the appropriate bootloader) launches into an instance of the Debian Installer.

L'installazione procederà nello stesso modo di un'installazione "Normale" come descritto sopra, ma nella fase dell'installazione del pacchetto, invece di usare *debootstrap* per prelevare e installare i pacchetti, l'immagine del filesystem live viene copiata sulla destinazione. Questo si ottiene con uno speciale udeb chiamato *live-installer*.

Dopo questa fase, l'Installatore Debian continua normalmente, installando e configurando elementi come bootloader e utenti locali, ecc.

Nota: per supportare nel bootloader sia la voce normale che quella live dell'installatore sullo stesso supporto si deve disabilitare *live-installer* preconfigurando *live-installer/enable=false*.

Installatore Debian "Desktop" : indipendentemente dal tipo di Installatore Debian incluso, d-i può essere lanciato cliccando un'icona sul desktop, in alcune situazioni più semplice per l'utente. Per poterne usufruire deve essere incluso il pacchetto *debian-installer-launcher*.

Si noti che *live-build* non include l'Installatore Debian nell'immagine in modo predefinito, necessita di essere espressamente abilitato con *lb config*. Inoltre, affinché l'installatore "Desktop" funzioni, il kernel del sistema live deve corrispondere a quello usato dal d-i per l'architettura specificata. Per esempio:

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

12.2 Personalizzare il Debian Installer con la preconfigurazione

Come descritto nell'appendice B del manuale dell'Installatore Debian all'indirizzo <https://www.debian.org/releases/stable/i386/apb.html>, "La preconfigurazione fornisce un modo per impostare le risposte alle domande poste durante il processo d'installazione senza la necessità di inserirle manualmente. Ciò permette di automatizzare totalmente molti tipi di installazione offrendo anche alcune caratteristiche normalmente non disponibili." Questo tipo di personalizzazione è compiuta in modo ottimale con *live-build* mettendo la configurazione in un file `preseed.cfg` incluso in `config/includes.installer/`. Ad esempio per preconfigurare l'impostazione della localizzazione su `en_US`:

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

12.3 Personalizzare il contenuto dell'Installatore Debian

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/includes.installer/`.

Progetto

Contribuire al progetto

13. Contribuire al progetto

When submitting a contribution, please clearly identify its copyright holder and include any applicable licensing statement. Note that to be accepted, the contribution must be licensed under the same license as the rest of the documents, namely, GPL version 3 or later.

I contributi al progetto, come traduzioni e patch, sono estremamente benvenuti. Chiunque può eseguire il commit direttamente sul repository; tuttavia chiediamo di inviare le modifiche più corpose in mailing list, per poterne prima discutere. Per maggiori informazioni vedere la sezione [«Contatti»](#).

The Live Systems Project uses Git as version control system and source code management. As explained in [«Git repositories»](#) there are two main development branches: **debian** and **debian-next**. Everybody can commit to the debian-next branches of the *live-boot*, *live-build*, *live-config*, *live-images*, *live-manual* and *live-tools* repositories.

Tuttavia ci sono alcune restrizioni. Il server rifiuta:

- push non fast-forward,
- commit merge,
- aggiunta o rimozione di tag e branch.

Anche se tutti i commit possono essere corretti, chiediamo di usare il buon senso ed eseguire buoni commit con dei buoni messaggi.

- Si scrivano messaggi costituiti da frasi in inglese esaurienti e utili, iniziati con una lettera maiuscola e terminanti con un punto. Solitamente cominceranno con la forma “Fixing/Adding/Removing/Correcting/Translating/...”.

- Scrivere buoni messaggi nei commit. La prima riga deve contenere un sunto accurato del contenuto del commit in quanto verrà incluso nel changelog. Se si necessita di aggiungere ulteriori spiegazioni, scriverle sotto lasciando una riga vuota dopo la prima e quindi un'altra vuota dopo ogni paragrafo. Le righe non devono superare gli 80 caratteri.

- Eseguire commit atomici, ovvero non mescolare cose non inerenti tra loro nello stesso commit ma farne uno per ogni modifica apportata.

13.1 Applicare le modifiche

Per eseguire il push al repository è necessario seguire la seguente procedura. Verrà usato *live-manual* come esempio per cui rimpiazzalo con il nome del repository su cui si vuole lavorare. Per informazioni dettagliate su come modificare *live-manual* si veda [«Contribuire a questo documento»](#).

- Prelevare la chiave pubblica:

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

- Aggiungere la seguente sezione alla propria configurazione di openssh-client:

```
$ cat >> ~/.ssh/config << EOF
Host live-systems.org
```

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

- Scaricare tramite ssh un clone del manuale:

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

- Assicurarsi di avere impostato autore e indirizzo email:

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

Importante: Notare che tutte le modifiche vanno eseguite sul ramo **debian-next** .

- Apportare le modifiche. In questo esempio si scrive prima una nuova sezione che si occupa di applicare patch e quindi prepararla al commit aggiungendo i file e scrivendo il messaggio in questo modo:

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

- Inviare il commit al server:

```
$ git push
```

680 Segnalare bug

681 14. Segnalare bug

682 Live systems are 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 on how to file good bug reports.

683 Per gli impazienti

- 684 • Per i problemi noti verificare sempre lo stato degli aggiornamenti dell'immagine sulla nostra pagina iniziale <http://live-systems.org/>.
- 685 • Before submitting a bug report always try to reproduce the bug with the **most recent versions** of the branch of *live-build*, *live-boot*, *live-config* and *live-tools* that you're using (like the newest 4.x version of *live-build* if you're using *live-build* 4).
- 686 • Si cerchi di fornire **informazioni il più dettagliate possibile** riguardo il bug. Questo comprende (almeno) la versione di *live-build*, *live-boot*, *live-config* e *live-tools* utilizzata e la distribuzione del sistema live che si sta creando.

687 14.1 Problemi noti

688 Giacché Debian **testing** e Debian **unstable** subiscono cambiamenti continui, quando si specifica l'una o l'altra come sistema di destinazione, può non essere sempre possibile una compilazione che vada a buon fine.

689 Se questo causa troppe difficoltà, non creare un sistema basato su **testing** o **unstable** ma usare piuttosto **stable**. *live-build* si basa su **stable** in modo predefinito.

690 I problemi noti al momento sono elencati sotto la sezione "status"

della nostra pagina iniziale <http://live-systems.org/>

Questo manuale non intende insegnare come identificare e risolvere correttamente i problemi dei pacchetti delle distribuzioni di sviluppo, tuttavia ci sono un paio di cose da provare: se la creazione di **testing** non va a buon fine provare con **unstable**; se non funziona nemmeno **unstable** tornare a **testing** ed effettuare il pinning da **unstable** alla nuova versione del pacchetto corrotto (si veda [APT pinning](#) per i dettagli).

14.2 Ricompilare da zero

Per essere certi che un particolare bug non sia causato dalla creazione di un sistema non pulito, ricostruire sempre l'intero sistema da zero per vedere se il bug sia riproducibile.

14.3 Usare pacchetti aggiornati

L'utilizzo di pacchetti datati può causare notevoli complicazioni nel tentativo di riprodurre (e alla fine risolvere) il problema. Assicurarsi che il sistema creato sia aggiornato e ogni pacchetto incluso nell'immagine lo sia a sua volta.

14.4 Raccogliere informazioni

Nella segnalazione si invita a fornire informazioni sufficienti. Dovrebbe almeno contenere l'esatta versione di *live-build* nella quale si è trovato il bug e i passi per riprodurlo. Con un po' di buon senso si può includere qualsiasi altro dettaglio rilevante che si ritiene utile per la risoluzione del problema.

Affinché la segnalazione del bug sia migliore possibile, si richiedono almeno le seguenti informazioni:

- Architettura del sistema ospitante
- Distribution of the host system
- Versione di *live-build* sul sistema ospitante
- Version of Python on the host system
- Versione di *debootstrap* o *cdebootstrap* sul sistema ospitante
- Architettura del sistema live
- Distribuzione del sistema live
- Versione di *live-boot* sul sistema ospitante
- Versione di *live-config* sul sistema live
- Versione di *live-tools* sul sistema ospitante

È possibile generare un registro del processo di costruzione usando il comando `tee`. Si raccomanda di farlo automaticamente con uno script `auto/build`; (si veda [«Gestire una configurazione»](#) per i dettagli).

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

All'avvio, *live-boot* e *live-config* conservano i loro registri in `/var/log/live/`. Controllarvi gli errori.

Inoltre, per escludere altri errori è sempre una buona idea creare un tar della propria directory `config/` e caricarlo da qualche parte (**non** inviarlo come allegato alla mailing list), in modo che sia per noi possibile riprodurre gli errori incontrati. Se ciò causa problemi (ad esempio a causa della dimensione) si può utilizzare l'output di `lb config --dump` che produce un sommario dell'albero di configurazione (elenca i file nelle sottodirectory di `config/` ma non le include).

71699

Ricordarsi che i file di registro da inviare vanno creati con l'impostazione della lingua inglese, ad esempio eseguendo il comando *live-build* proponendo `LC_ALL=C` oppure `LC_ALL=en_US`.

14.5 Se possibile isolare il caso non andato a buon fine

714

Se possibile, isolare il caso non andato a buon fine alla variazione più piccola che lo causa. Non è sempre facile da fare, perciò non preoccupatevi se non riuscite a gestirlo per la vostra segnalazione. Tuttavia, se si pianifica bene il ciclo di sviluppo adottando piccole modifiche per ogni iterazione, si riuscirà ad isolare il problema creando una configurazione semplificata che si avvicina all'attuale con l'aggiunta delle sole modifiche problematiche. Se si incontrano serie difficoltà nel trovare la causa, potrebbe essere che sono stati inseriti troppi cambiamenti in una sola volta e bisogna cambiare approccio.

715

14.6 Segnalare il bug del pacchetto giusto

716

Se non si sa quale sia il componente responsabile del bug o se il bug è uno generico riguardante il sistema live, si può fare una segnalazione per lo pseudo-pacchetto `debian-live`.

717

Tuttavia vi saremmo grati se tentate di restringere il campo in base a dove appare il bug.

718

14.6.1 Durante la compilazione mentre esegue il bootstrap

719

live-build avvia inizialmente un sistema Debian di base con *debootstrap* o *cdebootstrap*; può fallire a seconda dello strumento utilizzato e della distribuzione Debian che si sta avviando. Se il

720

bug appare a questo punto controllare che l'errore sia relativo ad uno specifico pacchetto Debian (più probabile) o allo strumento di avvio stesso.

721 In both cases, this is not a bug in the live system, but rather in Debian itself and probably we cannot fix it directly. Please report such a bug against the bootstrapping tool or the failing package.

722 14.6.2 Durante la compilazione mentre installa i pacchetti

723 *live-build* installa pacchetti aggiuntivi dall'archivio Debian e può fallire a seconda della distribuzione Debian e lo stato dell'archivio giornaliero. Se il bug appare a questo punto, controllare che l'errore sia riproducibile su un sistema normale.

724 If this is the case, this is not a bug in the live system, 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.

725 Se si verifica un problema utilizzando un mirror locale o un qualsiasi tipo di proxy è bene riprodurlo avviando da un mirror ufficiale.

726 14.6.3 In fase di avvio

727 Se l'immagine non si avvia segnalarlo alla mailing list con le informazioni richieste in [«Raccogliere informazioni»](#). Non dimenticare di menzionare esattamente come e quando l'immagine fallisce, utilizzando la virtualizzazione o hardware reale. Se si utilizza un qualsiasi sistema di virtualizzazione provare sempre su hardware reale prima di segnalare un bug; anche fornire un'istantanea dello schermo può essere molto utile.

14.6.4 In fase di esecuzione

728

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

729

14.7 Fare la ricerca

730

Prima di riportare il bug si prega di cercare sul web il messaggio d'errore o il sintomo ottenuti. Poiché è altamente improbabile essere l'unica persona ad incontrare un certo problema, c'è sempre la possibilità che sia stato discusso altrove e che siano stati proposti una soluzione, una patch o soluzione temporanea.

731

You should pay particular attention to the live systems 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.

732

In aggiunta bisogna controllare l'attuale elenco dei bug riguardanti *live-build*, *live-boot*, *live-config* e *live-tools* per vedere se sia già stato segnalato qualcosa di simile.

733

14.8 Dove segnalare i bug

734

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

735

In genere bisogna riportare gli errori in fase di compilazione verso il pacchetto *live-build*, quelli di avvio verso *live-boot* e quelli in fase di esecuzione a *live-config*. Se non siete certi di quale sia il pacchetto

736

appropriato o serve maggiore aiuto prima della segnalazione, inviate una segnalazione per lo pseudo-pacchetto debian-live. Ce ne occuperemo riassegnandolo dove più appropriato.

737

Si noti che i bug trovati nelle distribuzioni derivate da Debian (come Ubuntu e altre) **non** vanno segnalati a Debian BTS a meno che non siano riproducibili anche su un sistema Debian utilizzando pacchetti ufficiali Debian.

738 **Lo stile nello scrivere codice**

739 **15. Lo stile nello scrivere codice**

740 This chapter documents the coding style used in live
systems.

741 **15.1 Compatibilità**

- 742 • Non usare sintassi o semantiche mirate alla shell Bash. Ad
esempio l'uso di costrutti array.
- 743 • Utilizzare solo il sottoinsieme POSIX - ad esempio, usare \$(foo)
invece di 'foo'.
- 744 • È possibile verificare i propri script con “sh -n” e “checkbashisms”.
- 745 Assicurarsi che tutto il codice giri con 'set -e'.

746 **15.2 Rientri**

- 747 • Usare sempre i tab piuttosto che gli spazi.

748 **15.3 Ritorno a capo**

- 749 • Generalmente le righe sono composte da un massimo di 80
caratteri.
- 750 • Utilizzare lo “stile Linux” per le interruzioni di riga:

751 Sbagliato:

752

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

Corretto:

753

754

```
if foo
then
    bar
fi
```

- Lo stesso vale per le funzioni:

755

Sbagliato:

756

757

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

Corretto:

758

759

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

15.4 Variabili

760

- Le variabili vanno sempre scritte in maiuscolo. 761
- Le variabili usate in *live-build* iniziano sempre con il prefisso LB_. 762
- Le variabili interne temporanee in *live-build* dovrebbero iniziare
con il prefisso `<=underscore>LB_`. 763
- Le variabili locali iniziano con il prefisso *live-build* `<=underscore><=underscore>LB_`. 764

- Le variabili in *live-config* relative ai parametri di avvio iniziano con LIVE_.

765

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

- Tutte le altre variabili in *live-config* iniziano con il prefisso _.

766

- Intorno alle variabili utilizzare le graffe; ad esempio scrivere \${F00} invece di \$F00.

767

- Proteggere sempre le variabili con le virgolette per rispettare potenziali spaziature: scrivere "\${F00}" e non \${F00}.

768

- Per coerenza usare sempre le virgolette quando si assegnano valori alle variabili:

769

Sbagliato:

770

771

```
F00=bar
```

Corretto:

772

773

```
F00="bar"
```

- Utilizzando variabili multiple, quotare l'intera espressione:

774

Sbagliato:

775

776

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

Corretto:

777

778

15.5 Varie

779

- Per le chiamate a sed utilizzare "|" (senza virgolette intorno) come separatore, ad esempio "sed -e `s|'|`" (senza "").

780

- Non utilizzare il comando test per prove o confronti, usare "[" "]" (senza ""); ad esempio "if [-x /bin/foo]; ..." e non "if test -x /bin/foo; ...".

781

- Ove possibile utilizzare case invece di test, essendo più facile da leggere e più veloce in esecuzione.

782

- Per le funzioni utilizzare nomi che iniziano con la maiuscola per limitare i problemi con le variabili d'ambiente dell'utente.

783

Procedure

16. Procedure

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

16.1 Rilasci importanti

Rilasciare una nuova versione stabile di Debian implica che molti team differenti lavorino insieme; ad un certo punto si inserisce il team Live che prepara le immagini del sistema live. I requisiti per fare ciò sono:

- Un mirror contenente le versioni rilasciate per l'archivio debian e debian-security al quale possa accedere il debian-live build.
- Vanno resi noti i nomi dell'immagine (debian-live-VERSION-ARCH-FLAVOUR.iso).
- Bisogna sincronizzare i dati dal cd Debian (gli udeb escludono gli elenchi).
- Le immagini vengono create e ospitate su cimage.debian.org.

16.2 Rilasci minori

- Bisogna nuovamente aggiornare i mirror di debian e debian-security.
- Le immagini vengono create e ospitate su cimage.debian.org.
- Inviare email di annuncio.

16.2.1 Ultimo rilascio minore di un rilascio di Debian.

Quando si crea l'ultima serie di immagini per un rilascio di Debian che è stato spostato da ftp.debian.org a archive.debian.org, ricordarsi di sistemare i mirror del chroot e dei binari. In questo modo le vecchie immagini live create saranno ancora utili senza la necessità di modifiche da parte dell'utente.

16.2.2 Modello per l'annuncio di un rilascio minore.

Si può generare un'email per l'annuncio dei rilasci minori usando il modello sottostante e il seguente comando:

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

Si prega di controllare attentamente l'email prima di inviarla e passarla ad altri per le correzioni.

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

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

The images are available for download at:

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

and later at:

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

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

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

Additionally it includes the following Live-specific changes:

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

About Live Systems

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

About Debian

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

Contact Information

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

Repository Git

17. Repository Git

The list of all the available repositories of the Live Systems Project can be found at <http://live-systems.org/gitweb/>. The project's git URLs have the form: `protocol://live-systems.org/git/repository`. Thus, in order to clone *live-manual* read-only, launch:

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

oppure

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

oppure

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

The cloning addresses with write permission have the form: `git@live-systems.org:/repository`.

Quindi per clonare *live-manual* via ssh si userà:

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

Il ramo git del progetto Debian Live è costituito da molteplici branch differenti. I branch **debian** e **debian-next** sono particolarmente

degni di nota in quanto contengono il lavoro attuale che verrà incluso in ogni nuovo rilascio.

Dopo aver clonato uno dei repository esistenti sarete nel branch **debian**. Questo è adatto per prendere visione dello stato dell'ultimo rilascio del progetto ma prima di iniziare a lavorarci è cruciale passare a **debian-next**. Per farlo eseguire:

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

Il branch **debian-next**, che non è sempre soggetto al fast-forward, è dove si fa il commit di tutte le modifiche prima di essere incluse nel branch **debian**. È come un terreno di test, per fare un'analogia. Se si sta lavorando in questo branch e si necessita di eseguire il pull, bisogna usare `git pull --rebase` in modo che le modifiche locali siano preparate per il commit (stage) quando si fa il pull dal server, in questo modo saranno poste in cima a tutto il resto.

17.1 Gestire repository multipli

If you intend to clone several of the live systems repositories and want to switch to the **debian-next** branch right away to check the latest code, write a patch or contribute with a translation you ought to know that the git server provides a `mrconfig` file to ease the handling of multiple repositories. In order to use it you need to install the *mr* package and after that, launch:

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

Il comando clonerà e farà il checkout al ramo **debian-next** del repository di sviluppo dei pacchetti Debian prodotti dal progetto.

Questi includono tra gli altri il repository *live-images* che contiene le configurazioni usate per le immagini precompilate che il progetto pubblica per uso generico. Per maggiori informazioni su come utilizzare questo repository si veda <[Clonare una configurazione pubblicata tramite Git](#)>.

Esempi

Esempi

18. Esempi

This chapter covers example builds for specific use cases with live systems. If you are new to building your own live system 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.

18.1 Usare gli esempi

Per usare questi esempi è necessario un sistema per costruirveli sopra che soddisfi i requisiti elencati in [«Requisiti»](#) e avere *live-build* installato come descritto in [«Installare live-build»](#).

È da notare che per brevità in questi esempi non specifichiamo un mirror locale da usare per la costruzione. Usando un mirror locale, si possono accelerare considerevolmente i download. Si possono specificare le opzioni quando si usa `lb config`, come descritto in [«Mirror delle distribuzioni usati in fase di compilazione»](#) o, più convenientemente, impostare il predefinito per il proprio sistema in `/etc/live/build.conf`. Si crei semplicemente questo file e si impostino in esso le corrispondenti variabili `LB_MIRROR_*` per il mirror desiderato. Tutti gli altri mirror utilizzati nella costruzione avranno questi valori, ad esempio:

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

18.2 Tutorial 1: un'immagine predefinita

Caso d'uso: creazione di una prima semplice immagine, imparando i fondamenti di *live-build*.

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

Non può essere più semplice:

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

Esaminare i contenuti della directory `config/`; si noterà uno scheletro di configurazione pronto per essere personalizzato o, in questo caso, usato immediatamente per costruire un'immagine predefinita.

Ora, come utente root, generare l'immagine salvando un log con tee.

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

Assuming all goes well, after a while, the current directory will contain `live-image-i386.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»](#), 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.

18.3 Tutorial 2: servizio browser web

Caso d'uso: creazione di un'immagine per servizio browser web, imparando come applicare le personalizzazioni.

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

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

La scelta di LXDE per questo esempio riflette il desiderio di fornire un ambiente desktop minimale, dato che il punto focale dell'immagine è il singolo uso che abbiamo in mente, il browser web. Potremmo anche spingerci oltre e fornire una configurazione predefinita per il browser web in `config/includes.chroot/etc/iceweasel/profile/`, o pacchetti aggiuntivi di supporto per la fruizione di vari tipi di contenuti web, ma lasciamo questo come esercizio per il lettore.

Generare l'immagine, ancora come utente root, conservando un log come in [Tutorial 1](#):

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

Di nuovo, verificare che l'immagine sia a posto e collaudarla, come in [Tutorial 1](#).

848

18.4 Tutorial 3: un'immagine personalizzata

Caso d'uso: creazione di un progetto per costruire un'immagine personalizzata che contiene i pacchetti preferiti da portare con sé in una chiavetta USB ovunque si vada, e che evolve in revisioni successive allorché i bisogni o le preferenze cambino.

Dal momento che la nostra immagine personalizzata cambierà con le successive revisioni e che vogliamo tener traccia di questi cambiamenti, andando per tentativi ed eventualmente tornando indietro se qualcosa non funziona, conserveremo la nostra configurazione nel popolare sistema di controllo di versione git. Useremo anche le migliori pratiche di auto-configurazione tramite gli script auto come descritto in [Gestire una configurazione](#).

18.4.1 Prima revisione

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

Modificare `auto/config` come segue:

```
#!/bin/sh

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

Eseguire `lb config` per generare l'albero di configurazione utilizzando lo script `auto/config` appena creato:


```
$ lb config
```

Popolare ora l'elenco locale dei pacchetti:

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

Per prima cosa, `--architectures i386` assicura che sul nostro sistema amd64 costruiamo una versione a 32-bit utilizzabile sulla maggior parte delle macchine. In secondo luogo, usiamo `--linux-flavours pae` dato che non prevediamo di usare questa immagine su sistemi troppo vecchi. Terzo, abbiamo scelto il metapacchetto `task-lxde` per avere un desktop minimale. Infine abbiamo aggiunto due pacchetti preferiti: *iceweasel* e *xchat*.

Costruire quindi l'immagine:

```
# lb build
```

Notare che diversamente dai primi due tutorial non occorre più digitare `2>&1 |tee build.log` dato che questo è ora incluso in `auto/build`.

Una volta che l'immagine è stata collaudata (come in <Tutorial 1>) e che si è sicuri che funzioni correttamente, è il momento di inizializzare il repository `git`, aggiungendo solo gli script `auto` appena creati, e di fare poi il primo commit:

856

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

18.4.2 Seconda revisione

In questa revisione ripuliremo la prima compilazione, aggiungeremo il pacchetto *vlc* alla configurazione, dunque avverrà una ricompilazione, verifica e commit.

Il comando `lb clean` ripulirà tutti i file ottenuti con la precedente generazione eccetto la cache, che ci evita un nuovo download dei pacchetti. Ciò assicura che il successivo `lb build` eseguirà di nuovo tutti i passaggi per rigenerare i file dalla nuova configurazione.

```
# lb clean
```

Ora inserire il pacchetto *vlc* all'elenco locale dei pacchetti `config/package-lists/my.list.chroot`:

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

Rigenerare nuovamente:

```
# lb build
```

Verificare e, quando soddisfatti, eseguire il commit della revisione

successiva:

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

Ovviamente sono possibili cambiamenti alla configurazione più complicati, magari aggiungendo file in sottodirectory di `config/`. Quando si esegue il commit di nuove revisioni, si faccia solo attenzione a non modificare manualmente o fare un commit dei file al livello superiore di `config` che contengono le variabili `LB_*`, giacché sono anche prodotti dell'assemblaggio, e che sono sempre ripuliti da `lb clean` e ricreati con `lb config` attraverso i loro rispettivi script auto.

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 live systems.

18.5 Un client Kiosk VNC

Caso d'uso: creazione di un'immagine con *live-build* per avviare direttamente un server VNC.

Creare una directory per la compilazione e una configurazione di base al suo interno disabilitando i raccomandati per ottenere un sistema minimale. Quindi creare due elenchi di pacchetti: il primo generato con uno script fornito da *live-build* chiamato `Packages` (vedere [«Elenchi di pacchetti generati»](#)) e il secondo che include *xorg*, *gdm3*, *metacity* e *xvnc4viewer*.

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

Come spiegato in [«Modificare APT per risparmiare spazio»](#) potrebbe essere necessario riaggiungere alcuni pacchetti raccomandati al fine di far funzionare l'immagine correttamente.

Un modo semplice per elencare i raccomandati è usare *apt-cache*, ad esempio:

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

In questo esempio abbiamo scoperto che dobbiamo iserire nuovamente svariati pacchetti raccomandati da *live-config* e *live-boot*: *user-setup* perché il login automatico funzioni e *sudo* come programma essenziale per spegnere il sistema. Oltretutto può essere comodo aggiungere *live-tools* per poter copiare l'immagine in RAM e *eject* per espellere il supporto live alla fine. Quindi:

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

Successivamente creare la directory `/etc/skel` in `config/includes.chroot` e inserirvi un `.xsession` personalizzato per l'utente predefinito che lancerà *metacity* e avvierà *xvncviewer* connesso alla porta 5901 su un server con indirizzo 192.168.1.2:

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

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

exit
EOF
```

Compilare l'immagine:

```
# lb build
```

Buon divertimento.

18.6 Un'immagine base per una chiavetta USB da 128MB

Caso d'uso: creazione di un'immagine predefinita con alcuni componenti rimossi affinché possa stare su una chiavetta USB da 128MB, con un po' di spazio libero da usarsi come meglio si crede.

Quando si cerca di ottimizzare un'immagine affinché sia contenuta in un supporto, è necessario capire il compromesso che si deve fare tra la dimensione e la funzionalità. In questo esempio, taglieremo solo quanto basta per far sì che il tutto stia in 128M, senza fare nient'altro che distrugga l'integrità dei pacchetti contenuti, come eliminare localizzazioni con il pacchetto *localepurge* o altre ottimizzazioni "intrusive". È da notare che per creare un sistema minimale da zero viene utilizzata l'opzione `--debootstrap-options`.

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

Affinché l'immagine funzioni correttamente dobbiamo riaggiungere almeno due pacchetti raccomandati lasciati fuori dall'opzione `--apt-recommends false`. Vedere [Modificare APT per risparmiare spazio](#)

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

Costruire quindi l'immagine nel modo consueto:

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

All'autore del sistema al momento di scrivere, la seguente configurazione ha prodotto un'immagine di 77MB. Comparabile favorevolmente con i 177MB prodotta dalla configurazione predefinita nel [Tutorial 1](#).

Ciò che fa risparmiare più spazio, comparato alla creazione di un'immagine predefinita su un sistema con architettura i386, è la selezione del solo kernel 486 invece che quello predefinito `-k "486 686-pae"`. Lasciando fuori anche gli indici di APT con `--apt-indices false` si può salvare una certa quantità di spazio, il compromesso è usare `apt-get update` prima di usare *apt* nel sistema live. Saltare i pacchetti raccomandati con `--apt-recommends false` salva altro spazio, a scapito di alcuni pacchetti che ci si aspetta di trovare. `--debootstrap-options "--variant=minbase"` fa il bootstrap di un sistema minimale partendo da zero. Evitare di includere automaticamente pacchetti di firmware con `--firmware-chroot false` fa risparmiare altro spazio. E infine,

--mentest none evita l'installazione del programma per testare la memoria. 908

Note: A minimal system can also be achieved using hooks, like for example the `stripped.hook.chroot` hook found in `/usr/share/doc/live-build/examples/hooks`. It may shave off additional small amounts of space and produce an image of 62MB. However, it does so by removal of documentation and other files from packages installed on the system. This violates the integrity of those packages and that, as the comment header warns, may have unforeseen consequences. That is why using a minimal *debootstrap* is the recommended way of achieving this goal.

18.7 Un desktop GNOME localizzato e l'installatore

Caso d'uso: creazione di un'immagine con il desktop GNOME, localizzato in svizzero e che includa l'installatore.

Si vuole creare un'immagine iso ibrida per architettura i386 usando il nostro desktop preferito, in questo caso GNOME, contenente tutti gli stessi pacchetti che verrebbero installati dall'installatore Debian standard per GNOME.

Il problema iniziale è di scoprire i nomi dei task della lingua appropriati, attualmente, *live-build* non aiuta in questo. Si può essere fortunati o arrivarci con vari tentativi, ma c'è uno strumento `grep-dctrl` il quale può essere utilizzato per scavare nelle descrizioni in `tasksel-data`, perciò assicurarsi di avere entrambi questi pacchetti:

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

Ora si possono cercare i task appropriati:

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

Con questo comando, si è chiaramente scoperto che il task si chiama `german`. Ora per trovare i task correlati:

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

Durante il boot verrà generata la localizzazione **de_CH.UTF-8** e selezionato il layout di tastiera `*{ch}`, mettiamo ora insieme questi pezzi. Ricordando che i metapacchetti task iniziano con `task-` (come descritto in [«Usare metapacchetti»](#)), specifichiamo questi parametri d'avvio per la lingua, quindi aggiungiamo i pacchetti con priorità standard e tutti i metapacchetti task al nostro elenco in questo modo:

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

913

Notare che è stato incluso il pacchetto *debian-installer-launcher* in modo da poter lanciare l'installer dal desktop della live, e che è stato anche specificato il kernel 486, dato che attualmente è necessario che il kernel dell'installer e quello del sistema live coincidano affinché il launcher funzioni correttamente.

Appendice

Style guide

19. Style guide

19.1 Guidelines for authors

This section deals with some general considerations to be taken into account when writing technical documentation for *live-manual*. They are divided into linguistic features and recommended procedures.

Note: Authors should first read [Contributing to this document](#)

19.1.1 Linguistic features

- *Use plain English*

Keep in mind that a high percentage of your readers are not native speakers of English. So as a general rule try to use short, meaningful sentences, followed by a full stop.

This does not mean that you have to use a simplistic, naive style. It is a suggestion to try to avoid, as much as possible, complex subordinate sentences that make the text difficult to understand for non-native speakers of English.

- *Variety of English*

The most widely spread varieties of English are British and American so it is very likely that most authors will use either one or the other. In a collaborative environment, the ideal variety would be “International English” but it is very difficult, not to say impossible, to decide on which variety among all the existing ones, is the best to use.

We expect that different varieties may mix without creating misunderstandings but in general terms you should try to be

coherent and before deciding on using British, American or any other English flavour at your discretion, please take a look at how other people write and try to imitate them.

- *Be balanced*

Do not be biased. Avoid including references to ideologies completely unrelated to *live-manual*. Technical writing should be as neutral as possible. It is in the very nature of scientific writing.

- *Be politically correct*

Try to avoid sexist language as much as possible. If you need to make references to the third person singular preferably use “they” rather than “he” or “she” or awkward inventions such as “s/he”, “s(he)” and the like.

- *Be concise*

Go straight to the point and do not wander around aimlessly. Give as much information as necessary but do not give more information than necessary, this is to say, do not explain unnecessary details. Your readers are intelligent. Presume some previous knowledge on their part.

- *Minimize translation work*

Keep in mind that whatever you write will have to be translated into several other languages. This implies that a number of people will have to do an extra work if you add useless or redundant information.

- *Be coherent*

As suggested before, it is almost impossible to standardize a collaborative document into a perfectly unified whole. However, every effort on your side to write in a coherent way with the rest of the authors will be appreciated.

937	<ul style="list-style-type: none"> • <i>Be cohesive</i> 		
938	Use as many text-forming devices as necessary to make your text cohesive and unambiguous. (Text-forming devices are linguistic markers such as connectors).		
939	<ul style="list-style-type: none"> • <i>Be descriptive</i> 		
940	It is preferable to describe the point in one or several paragraphs than merely using a number of sentences in a typical “changelog” style. Describe it! Your readers will appreciate it.		
941	<ul style="list-style-type: none"> • <i>Dictionary</i> 		
942	Look up the meaning of words in a dictionary or encyclopedia if you do not know how to express certain concepts in English. But keep in mind that a dictionary can either be your best friend or can turn into your worst enemy if you do not know how to use it correctly.		
943	English has the largest vocabulary that exists (with over one million words). Many of these words are borrowings from other languages. When looking up the meaning of words in a bilingual dictionary the tendency of a non-native speaker of English is to choose the one that sounds more similar in their mother tongue. This often turns into an excessively formal discourse which does not sound quite natural in English.		
944	As a general rule, if a concept can be expressed using different synonyms, it is a good advice to choose the first word proposed by the dictionary. If in doubt, choosing words of Germanic origin (Usually monosyllabic words) is often the right thing to do. Be warned that these two techniques might produce a rather informal discourse but at least your choice of words will be of wide use and generally accepted.		
945	Using a dictionary of collocations is recommended. They are extremely helpful when it comes to know which words usually occur together.		
	Again it is a good practice to learn from the work of others. Using a search engine to check how other authors use certain expressions may help a lot.		946
	<ul style="list-style-type: none"> • <i>False friends, idioms and other idiomatic expressions</i> 		947
	Watch out for false friends. No matter how proficient you are in a foreign language you cannot help falling from time to time in the trap of the so called “false friends”, words that look similar in two languages but whose meanings or uses might be completely different.		948
	Try to avoid idioms as much as possible. “Idioms” are expressions that may convey a completely different meaning from what their individual words seem to mean. Sometimes, idioms might be difficult to understand even for native speakers of English!		949
	<ul style="list-style-type: none"> • <i>Avoid slang, abbreviations, contractions...</i> 		950
	Even though you are encouraged to use plain, everyday English, technical writing belongs to the formal register of the language.		951
	Try to avoid slang, unusual abbreviations that are difficult to understand and above all contractions that try to imitate the spoken language. Not to mention typical irc and family friendly expressions.		952
	19.1.2 Procedures		953
	<ul style="list-style-type: none"> • <i>Test before write</i> 		954
	It is important that authors test their examples before adding them to <i>live-manual</i> to ensure that everything works as described. Testing on a clean chroot or VM can be a good starting point. Besides, it would be ideal if the tests were then carried out on different machines with different hardware to spot possible problems that may arise.		955

• *Examples*

When providing an example try to be as specific as you can. An example is, after all, just an example.

It is often better to use a line that only applies to a specific case than using abstractions that may confuse your readers. In this case you can provide a brief explanation of the effects of the proposed example.

There may be some exceptions when the example suggests using some potentially dangerous commands that, if misused, may cause data loss or other similar undesirable effects. In this case you should provide a thorough explanation of the possible side effects.

• *External links*

Links to external sites should only be used when the information on those sites is crucial when it comes to understanding a special point. Even so, try to use links to external sites as sparsely as possible. Internet links are likely to change from time to time resulting in broken links and leaving your arguments in an incomplete state.

Besides, people who read the manual offline will not have the chance to follow those links.

• *Avoid branding and things that violate the license under which the manual is published*

Try to avoid branding as much as possible. Keep in mind that other downstream projects might make use of the documentation you write. So you are complicating things for them if you add certain specific material.

live-manual is licensed under the GNU GPL. This has a number of implications that apply to the distribution of the material (of any

kind, including copyrighted graphics or logos) that is published with it.

• *Write a first draft, revise, edit, improve, redo if necessary*

- Brainstorm!. You need to organize your ideas first in a logical sequence of events.

- Once you have somehow organized those ideas in your mind write a first draft.

- Revise grammar, syntax and spelling. Keep in mind that the proper names of the releases, such as **jessie** or **sid**, should not be capitalized when referred to as code names. In order to check the spelling you can run the “spell” target. i.e. make spell

- Improve your statements and redo any part if necessary.

• *Chapters*

Use the conventional numbering system for chapters and subtitles. e.g. 1, 1.1, 1.1.1, 1.1.2 ... 1.2, 1.2.1, 1.2.2 ... 2, 2.1 ... and so on. See markup below.

If you have to enumerate a series of steps or stages in your description, you can also use ordinal numbers: First, second, third ... or First, Then, After that, Finally ... Alternatively you can use bulleted items.

• *Markup*

And last but not least, *live-manual* uses `<SiSU>` to process the text files and produce a multiple format output. It is recommended to take a look at `<SiSU's manual>` to get familiar with its markup, or else type:

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

Here are some markup examples that may prove useful:

- For emphasis/bold text:

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

produces: **foo** or **foo** . Use it to emphasize certain key words.

- For italics:

```
/ {foo} /
```

produces: *foo*. Use them e.g. for the names of Debian packages.

- For monospace:

```
# {foo} #
```

produces: `foo`. Use it e.g. for the names of commands. And also to highlight some key words or things like paths.

- For code blocks:

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

produces:

```
$ foo
# bar
```

Use `{` to open and `}` to close the tags. It is important to remember to leave a space at the beginning of each line of code.

19.2 Guidelines for translators

This section deals with some general considerations to be taken into account when translating the contents of *live-manual*.

As a general recommendation, translators should have read and understood the translation rules that apply to their specific languages. Usually, translation groups and mailing lists provide information on how to produce translated work that complies with Debian quality standards.

Note: Translators should also read [Contributing to this document](#). In particular the section [Translation](#)

19.2.1 Translation hints

• Comments

The role of the translator is to convey as faithfully as possible the meaning of words, sentences, paragraphs and texts as written by the original authors into their target language.

So they should refrain from adding personal comments or extra bits of information of their own. If they want to add a comment for other translators working on the same documents, they can leave

it in the space reserved for that. That is, the header of the strings in the **po** files preceded by a number sign # . Most graphical translation programs can automatically handle those types of comments.

• *TN, Translator's Note*

It is perfectly acceptable however, to include a word or an expression in brackets in the translated text if, and only if, that makes the meaning of a difficult word or expression clearer to the reader. Inside the brackets the translator should make evident that the addition was theirs using the abbreviation "TN" or "Translator's Note".

• *Impersonal sentences*

Documents written in English make an extensive use of the impersonal form "you". In some other languages that do not share this characteristic, this might give the false impression that the original texts are directly addressing the reader when they are actually not doing so. Translators must be aware of that fact and reflect it in their language as accurately as possible.

• *False friends*

The trap of "false friends" explained before especially applies to translators. Double check the meaning of suspicious false friends if in doubt.

• *Markup*

Translators working initially with **pot** files and later on with **po** files will find many markup features in the strings. They can translate the text anyway, as long as it is translatable, but it is extremely important that they use exactly the same markup as the original English version.

• *Code blocks*

Even though the code blocks are usually untranslatable, including them in the translation is the only way to score a 100% complete translation. And even though it means more work at first because it might require the intervention of the translators if the code changes, it is the best way, in the long run, to identify what has already been translated and what has not when checking the integrity of the .po files.

• *Newlines*

The translated texts need to have the exact same newlines as the original texts. Be careful to press the "Enter" key or type if they appear in the original files. These newlines often appear, for instance, in the code blocks.

Make no mistake, this does not mean that the translated text needs to have the same length as the English version. That is nearly impossible.

• *Untranslatable strings*

Translators should never translate:

- The code names of releases (which should be written in lowercase)
- The names of programs
- The commands given as examples
- Metadata (often between colons :**metadata**:)
- Links
- Paths

SiSU Metadata, document information

Titolo: Manuale di Live Systems

Autore: Live Systems Project <debian-live@lists.debian.org>

Diritti del lettore: Copyright: Copyright (C) 2006-2014 Live Systems Project

License: Questo programma è software libero: è possibile ridistribuirlo e modificarlo secondo i termini della GNU General Public License come pubblicata dalla Free Software Foundation, sia la versione 3 della licenza o (a scelta) una versione successiva.

Questo programma è distribuito nella speranza che possa essere utile, ma SENZA ALCUNA GARANZIA, nemmeno la garanzia implicita di COMMERCIALIZZABILITÀ o IDONEITÀ PER UN PARTICOLARE SCOPO. Vedere la GNU General Public License per ulteriori dettagli.

Si dovrebbe aver ricevuto una copia della GNU General Public License con questo programma. In caso contrario, vedere <<http://www.gnu.org/licenses/>>.

Il testo completo della GNU General Public License può essere trovato nel file /usr/share/common-licenses/GPL-3.

Casa editrice: Live Systems Project <debian-live@lists.debian.org>

Data: 2014-10-25

Version Information

Sorgente: live-manual.ssm.sst

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

Source Digest: SHA256(live-manual.ssm.sst)=12c8741531e9d09782665944-078216c35b9ba28f47f6e41c76ac1946b5342aa3

Generated

Data di ultima generazione (ao metaverse): 2014-10-25 13:15:27 +0000

Generato da: SiSU 5.7.1 of 2014w41/7 (2014-10-19)

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