

Manual de Live Systems

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

Copyright © 2006-2014 Live Systems Project

Este programa es software libre: puede ser redistribuido y/o modificado bajo los términos de la GNU General Public License publicada por la Free Software Foundation, bien de la versión 3 de la Licencia, o (a su elección) cualquier versión posterior.

Este programa se distribuye con la esperanza de que sea útil, pero SIN NINGUNA GARANTÍA, incluso sin la garantía implícita de COMERCIALIZACIÓN o IDONEIDAD PARA UN PROPÓSITO PARTICULAR. Consulte la GNU General Public License para más detalles.

Debería haber recibido una copia de la General Public License GNU junto con este programa. Si no, vea <http://www.gnu.org/licenses/>.

El texto completo de la GNU Licencia Pública General se pueden encontrar en `/usr/share/common-licenses/GPL-3`

Contents

Acerca de este manual	2
Acerca de este manual	3
1. Acerca de este manual	3
1.1 Para el impaciente.	3
1.2 Términos	3
1.3 Autores	5
1.4 Cómo contribuir a este documento	5
1.4.1 Aplicar cambios	5
1.4.2 Traducción	6
Contribuir al Live Systems Project	8
2. Acerca del Live Systems Project	8
2.1 Motivación	8
2.1.1 Desventajas de los sistemas en vivo actuales	8
2.1.2 El porqué de crear un sistema en vivo propio.	8
2.2 Filosofía	8
2.2.1 Solamente paquetes sin modificación alguna de Debian «main»	8
2.2.2 Sin configuración especial para el sistema en vivo	9
2.3 Contacto	9
Usuario	10

Instalación	11
3. Instalación	11
3.1 Requisitos	11
3.2 Instalación de live-build	11
3.2.1 Desde el repositorio Debian.	11
3.2.2 A partir del código fuente	11
3.2.3 A partir de «instantáneas»	12
3.3 Instalación de live-boot y live-config	12
3.3.1 Desde el repositorio Debian.	12
3.3.2 A partir del código fuente	12
3.3.3 A partir de «instantáneas»	13
Conceptos básicos	14
4. Conceptos básicos	14
4.1 ¿Qué es un sistema en vivo?	14
4.2 Descarga de imágenes prefabricadas	15
4.3 Uso del servicio de creación de imágenes web	15
4.3.1 Uso y advertencias del servicio de creación de imágenes web	15
4.4 Primeros pasos: creación de una imagen ISO híbrida	16
4.5 Usar una imagen ISO híbrida	16
4.5.1 Grabar una imagen ISO en un medio físico.	16
4.5.2 Copiar una imagen ISO híbrida a un dispositivo USB	17
4.5.3 Usar el espacio libre en el dispositivo USB	17
4.5.4 Arrancar el medio en vivo	18
4.6 Usar una máquina virtual para pruebas	18
4.6.1 Probar una imagen ISO con QEMU	18
4.6.2 Probar una imagen ISO con VirtualBox	19
4.7 Construir y utilizar una imagen HDD	19

4.8 Creación de una imagen de arranque en red . . .	20	Personalización de contenidos	30
4.8.1 Servidor DHCP	21	7. Descripción general de la personalización.	30
4.8.2 Servidor TFTP	21	7.1 Configuración en el momento de la creación vs en	
4.8.3 Servidor NFS	22	el momento del arranque	30
4.8.4 Cómo probar el arranque en red	22	7.2 Etapas de la creación	30
4.8.5 Qemu	22	7.3 Opciones para lb config en ficheros	31
4.9 Arrancar desde internet	22	7.4 Tareas de personalización	31
4.9.1 Conseguir los ficheros para arrancar desde		Personalización de la instalación de paquetes	32
internet	23	8. Personalización de la instalación de paquetes	32
4.9.2 Arrancar imágenes webboot	23	8.1 Origen de los paquetes	32
Descripción general de las herramientas	24	8.1.1 Distribución, áreas de archivo y modo	32
5. Descripción general de las herramientas	24	8.1.2 Réplicas de Distribución Debian	33
5.1 El paquete live-build	24	8.1.3 Réplicas de Distribution utilizadas durante	
5.1.1 El comando lb config	24	la creación	33
5.1.2 El comando lb build	25	8.1.4 Réplicas de distribución Debian utilizadas	
5.1.3 El comando lb clean	25	en la ejecución.	33
5.2 El paquete live-boot	25	8.1.5 Repositorios adicionales	34
5.3 El paquete live-config	25	8.2 Selección de los paquetes a instalar	34
Gestionar una configuración	27	8.2.1 Listas de paquetes	34
6. Gestionar una configuración	27	8.2.2 Utilizar metapaquetes	34
6.1 Gestionar cambios en la configuración	27	8.2.3 Listas de paquetes locales	35
6.1.1 ¿Por qué utilizar scripts auto? ¿Qué		8.2.4 Listas de paquetes locales para la etapa	
hacen?	27	binary	35
6.1.2 Usar scripts auto de ejemplo	27	8.2.5 Generar listas de paquetes	36
6.2 Clonar una configuración publicada a través de Git	28	8.2.6 Utilización de condiciones dentro de las	
		listas de paquetes	36
		8.2.7 Eliminación paquetes durante la	
		instalación	36
		8.2.8 Tareas de Escritorio e Idioma	37
		8.2.9 Versión y tipo de kernel	37
		8.2.10 Kernels personalizados	38

8.3 Instalar paquetes modificados o de terceros . . .	38	10.3 Persistencia	48
8.3.1 Método packages.chroot para instalar paquetes personalizados	39	10.3.1 El fichero persistence.conf	49
8.3.2 Método de repositorio APT para instalar paquetes personalizados	39	10.3.2 Utilizar varios medios persistentes	50
8.3.3 Paquetes personalizados y APT	39	10.4 Utilizar persistencia con cifrado	50
8.4 Configurar APT en la creación	39	Personalización de la imagen binaria	53
8.4.1 Utilizar apt o aptitude	40	11. Personalización de la imagen binaria	53
8.4.2 Utilización de un proxy con APT	40	11.1 Gestores de arranque	53
8.4.3 Ajuste de APT para ahorrar espacio	40	11.2 Metadatos ISO	53
8.4.4 Pasar opciones a apt o a aptitude	41	Personalización del Instalador de Debian	54
8.4.5 APT pinning	41	12. Personalización del Instalador de Debian	54
Personalización de contenidos	43	12.1 Tipos de imágenes según el instalador	54
9. Personalización de contenidos	43	12.2 Personalizando el Instalador de Debian mediante preconfiguración	55
9.1 Includes	43	12.3 Personalizar el contenido del Instalador de Debian	55
9.1.1 Includes locales en Live/chroot	43	Proyecto	56
9.1.2 Includes locales en Binary	44	Contribuir al proyecto	57
9.2 Scripts gancho (Hooks)	44	13. Contribuir al proyecto	57
9.2.1 Scripts gancho locales en Live/chroot	44	13.1 Realizar cambios	57
9.2.2 Scripts gancho en tiempo de arranque	44	Cómo informar acerca de errores.	59
9.2.3 Scripts gancho locales en Binary	45	14. Informes de errores.	59
9.3 Preconfiguración de las preguntas de Debconf	45	14.1 Problemas conocidos	59
Personalización del comportamiento en tiempo de ejecución.	46		
10. Personalización del comportamiento en tiempo de ejecución.	46		
10.1 Personalización del usuario por defecto del sistema en vivo	46		
10.2 Personalización de las variantes locales e idioma	46		

14.2 Reconstruir desde cero	59	Repositorios Git	67
14.3 Utilizar paquetes actualizados	59	17. Repositorios Git	67
14.4 Recopilar información	59	17.1 Manejo de múltiples repositorios	67
14.5 Aislar el fallo si es posible	60		
14.6 Utilizar el paquete correcto sobre el que informar del error	60	Ejemplos	69
14.6.1 En la preinstalación (bootstrap) en tiempo de creación.	61		
14.6.2 Mientras se instalan paquetes en tiempo de creación.	61	Ejemplos	70
14.6.3 En tiempo de arranque	61	18. Ejemplos	70
14.6.4 En tiempo de ejecución	61	18.1 Uso de los ejemplos	70
14.7 Hacer la investigación	61	18.2 Tutorial 1: Una imagen predeterminada	70
14.8 Dónde informar de los fallos	62	18.3 Tutorial 2: Una utilidad de navegador web	71
		18.4 Tutorial 3: Una imagen personalizada	71
Estilo de código	63	18.4.1 Primera revisión	71
15. Estilo de código	63	18.4.2 Segunda revisión	72
15.1 Compatibilidad	63	18.5 Un cliente VNC kiosk	73
15.2 Sangrado	63	18.6 Una imagen básica para un pendrive USB de 128MB	74
15.3 Ajuste de líneas	63	18.7 Un escritorio GNOME con variante local e instalador	75
15.4 Variables	63		
15.5 Miscelánea	64	Apéndice	77
Procedimientos	65	Style guide	78
16. Procedimientos	65	19. Guía de estilo	78
16.1 Principales lanzamientos	65	19.1 Instrucciones para los autores	78
16.2 Nuevas versiones	65	19.1.1 Aspectos lingüísticos	78
16.2.1 Última actualización de una versión Debian	65	19.1.2 Procedimientos	80
16.2.2 Plantilla para anunciar nuevas versiones.	65	19.2 Directrices para los traductores	81
		19.2.1 Consejos de traducción	82

SiSU Metadata, document information	83
--	-----------

1	Manual de Live Systems
---	-------------------------------

2	Acerca de este manual
---	------------------------------

Acerca de este manual

1. Acerca de este manual

El objetivo principal de este manual es servir como único punto de acceso a toda la documentación referente al Live Systems Project y en particular al software que el proyecto crea para Debian 8.0 “jessie”. Se puede encontrar siempre una versión actualizada en <http://live-systems.org/>

live-manual está principalmente enfocado a ayudar en la creación de un sistema en vivo y no está dirigido al usuario final de estos sistemas. Un usuario final puede encontrar información útil en las siguientes secciones: **«Conceptos básicos»** que cubre la descarga de imágenes prefabricadas y la preparación de imágenes para arrancar un sistema desde un medio de almacenamiento o desde una red local, ya sea utilizando el constructor web o ejecutando *live-build* directamente en el sistema. **«Personalización del comportamiento en tiempo de ejecución»** que describe algunas de las opciones que pueden especificarse en el indicador de arranque, como pueden ser la selección de la distribución del teclado, las variantes locales o la persistencia.

Algunos de los comandos mencionados en el texto deben ser ejecutados con privilegios de superusuario, que pueden ser obtenidos accediendo a la cuenta de root mediante la orden `su` o mediante la orden `sudo`. Para distinguir las ordenes que deben ser ejecutadas como usuario no privilegiado de las que si requieren privilegios de superusuario se ha prefijado con `$` las primeras y con `#` las segundas. Estos símbolos no son parte de la orden.

1.1 Para el impaciente.

Aunque se cree que todo lo descrito en este manual es

importante para la mayoría de los usuarios, es cierto que hay mucho material a interiorizar y que los usuarios desean experimentar con las herramientas de forma rápida y satisfactoria antes de entrar en detalles. Por lo tanto, se sugiere leer siguiendo el siguiente orden.

Primero, leer el capítulo **«Acerca de este manual»**, desde el principio y terminando en la sección **«Términos»**. Después, saltar hasta los tres tutoriales que están al principio de la sección **«Ejemplos»** pensados para aprender a configurar y construir imágenes de forma básica. Se deberá leer primero **«Uso de los ejemplos»**, seguido de **«Tutorial 1: Una imagen predeterminada»** y **«Tutorial 2: Una utilidad de navegador web»**, para finalizar con **«Tutorial 3: Una imagen personalizada»**. Al final de estos tutoriales, el lector tendrá una visión de lo que se puede hacer con los sistemas en vivo.

Se anima a profundizar en el estudio del manual con la lectura detenida del siguiente capítulo: **«Conceptos básicos»**, y de una manera más somera el capítulo **«La creación de una imagen netboot»**, para acabar con la lectura de **«Descripción general de la personalización»** y los capítulos que le siguen. Se espera que, en este punto, el lector esté lo suficientemente motivado con lo que se puede hacer con los sistemas en vivo para leer el resto del manual, de principio a fin.

1.2 Términos

- **Sistema en vivo** : Se entiende por sistema en vivo aquel sistema operativo que se puede arrancar sin instalación previa en el disco duro. Un sistema en vivo no altera ningún sistema operativo previamente instalado ni ningún fichero existente en el disco duro de la máquina a menos que se le instruya para hacerlo. Los sistemas en vivo son arrancados típicamente desde medios extraíbles como CDs, DVDs o

llaves USB. Algunos pueden también arrancar desde la red local (utilizando imágenes tipo netboot, ver <Creación de una imagen de arranque en red>), o incluso desde internet utilizando el parámetro de arranque `fetch=URL`, ver <Arrancar desde internet>).

- 14 • **Medio en vivo** : A diferencia de sistema en vivo, el medio en vivo se refiere al CD, DVD o memoria USB donde se copia el fichero binario producido por *live-build* y utilizado para arrancar el sistema en vivo. Más ampliamente, el término también se refiere a cualquier lugar en el que reside el fichero binario a los efectos de iniciar el sistema en vivo, tales como la ubicación de los ficheros de arranque de red.
- 15 • **Live Systems Project** : Es un proyecto que mantiene, entre otros, los paquetes *live-boot*, *live-build*, *live-config*, *live-tools* y *live-manual*.
- 16 • **Sistema huésped** : Es el conjunto de herramientas y equipo utilizado para crear el sistema en vivo.
- 17 • **Sistema objetivo** : Es el conjunto de herramientas y equipo donde se ejecutará el sistema en vivo.
- 18 • **live-boot** : Es una colección de scripts que serán responsables de arrancar el sistema en vivo.
- 19 • **live-build** : Una colección de scripts utilizados para construir sistemas en vivo personalizados.
- 20 • **live-config** : Es una colección de scripts utilizados para configurar un sistema en vivo durante el proceso de arranque.
- 21 • **live-tools** : Una colección de scripts adicionales que se utilizan para realizar tareas útiles en un sistema en vivo en ejecución.
- 22 • **live-manual** : Este documento forma parte de un paquete llamado *live-manual*.

- 23 • **Instalador de Debian (Debian Installer o d-i)** : Es el mecanismo oficial de instalación para la distribución Debian.
- 24 • **Parámetros de arranque** : Parámetros que son entregados al gestor de arranque (bootloader) para modificar el comportamiento del kernel o del conjunto de scripts *live-config*. Son llamados también Parámetros de kernel u Opciones de arranque.
- 25 • **chroot** : El programa *chroot*, *chroot(8)*, permite ejecutar diferentes instancias de un entorno GNU/Linux en un solo sistema de manera simultánea sin necesidad de reiniciar el sistema.
- 26 • **Imagen binaria** : Es un fichero que contiene el sistema en vivo. Su nombre puede ser, por ejemplo, *live-image-i386.hybrid.iso* o *live-image-i386.img* dependiendo de su formato.
- 27 • **Distribución objetivo** : Es la versión de la distribución Debian en la cual estará basado el sistema en vivo que puede diferir de la versión de la distribución en el sistema huésped.
- 28 • **stable/testing/unstable** : La distribución **stable**, actualmente llamada **wheezy**, contiene la última distribución Debian publicada oficialmente. La distribución **testing**, temporalmente llamada **jessie**, está en la rampa de salida para ser la próxima distribución **stable**. La principal ventaja de utilizar esta distribución es que tiene versiones de programas más recientes si se compara con la versión **stable**. La distribución **unstable**, permanentemente llamada **sid**, es dónde se realiza el desarrollo de Debian. Generalmente esta distribución es usada por los desarrolladores y aquellos que les gusta vivir al filo de lo imposible. A lo largo del manual, se usan sus nombres en clave, como por ejemplo **jessie** o **sid**, ya que es lo que las mismas herramientas reconocen.

1.3 Autores

Lista de autores (en orden alfabético):

- Ben Armstrong
- Brendan Sleight
- Carlos Zufferri
- Chris Lamb
- Daniel Baumann
- Franklin Piat
- Jonas Stein
- Kai Hendry
- Marco Amadori
- Mathieu Geli
- Matthias Kirschner
- Richard Nelson
- Trent W. Buck

1.4 Cómo contribuir a este documento

Este manual se ha creado como un proyecto comunitario y cualquier propuesta para su mejora u otras contribuciones son siempre bienvenidas. Ver la sección [«Contribuir al proyecto»](#) para obtener información detallada sobre cómo obtener la clave pública y hacer buenos commits.

1.4.1 Aplicar cambios

Para realizar cambios en el manual en Inglés se debe editar los ficheros adecuados en `manual/en/` pero antes de enviar una contribución se debería realizar una visualización del trabajo realizado. Para ello asegurarse de tener instalados los paquetes necesarios para la construcción de *live-manual* ejecutando la siguiente orden:

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

Se puede realizar la construcción del manual posicionándose en el directorio de nivel superior, o sea, en el directorio clonado mediante Git y ejecutando la siguiente orden:

```
$ make build
```

Ya que construir el manual completo en todos los idiomas disponibles cuesta bastante rato, los autores seguramente estaran interesados en utilizar alguno de los siguientes atajos a la hora de revisar la documentación que hayan añadido al manual en inglés. Utilizando `PROOF=1` se crea *live-manual* en formato html, pero sin los documentos html segmentados, y utilizando `PROOF=2` se crea *live-manual* en formato pdf pero sólo en retrato A4 y carta. Por este motivo, utilizar cualquiera de las opciones `PROOF=` puede llegar a ahorrar una cantidad de tiempo considerable, por ejemplo.

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

Cuando se revisa alguna de las traducciones, es posible construir sólo un idioma ejecutando, por ejemplo:

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

Es posible generar el documento por formato:

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

O combinar ambos, por ejemplo:

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

Después de revisar el trabajo y asegurarse de que todo está bien, no ejecutar `make commit` a menos de que se actualicen las traducciones al mismo tiempo, y en ese caso, no mezclar los cambios al manual en inglés con las traducciones en el mismo commit, sino en commits separados. Ver la sección [«Traducción»](#) para más detalles.

1.4.2 Traducción

Para traducir *live-manual*, seguir estos pasos, dependiendo de si se está comenzando una traducción desde cero o si se continúa trabajando en una traducción ya comenzada:

- Empezar una nueva traducción desde cero
 - Traducir los ficheros **about_manual.ssi.pot**, **about_project.ssi.pot** y **index.html.in.pot** de `manual/pot/` al idioma deseado con cualquier editor

(como puede ser *poedit*) . Enviar los ficheros `.po` traducidos a la lista de correo para revisar su integridad. La comprobación de integridad de *live-manual* no sólo se asegura de que los ficheros `.po` estén 100% traducidos sino que también detecta posibles errores.

- Una vez comprobados, para activar una nueva lengua en el autobuild basta con añadir los ficheros traducidos inicialmente a `manual/po/${LANGUAGE}/` y ejecutar `make commit`. Y entonces, editar `manual/_sisu/home/index.-html` añadiendo el nombre de la lengua y su nombre en inglés entre paréntesis.
- Continuar con una traducción ya comenzada
 - Si el nuevo idioma ya ha sido añadido, se puede continuar la traducción de los ficheros `.po` restantes en `manual/po/${LANGUAGE}/` de manera aleatoria utilizando el editor preferido (como por ejemplo *poedit*) .
 - No se debe olvidar la ejecución del comando `make commit` para actualizar los manuales traducidos a partir de los ficheros `.po`. Entonces se puede revisar los cambios ejecutando `make build` antes de `git add .`, `git commit -m "Translating..."` y `git push`. Recordar que como `make build` puede tardar una cantidad considerable de tiempo, se pueden revisar las diferentes lenguas de forma individual como se explica en [«Aplicar cambios»](#)

Después de ejecutar `make commit` se podrá observar bastante texto en la pantalla. Básicamente son mensajes informativos sobre el estado del proceso y también algunas pistas sobre lo que se puede hacer para mejorar *live-manual*. A menos que se vea un error fatal, generalmente se puede proceder y enviar la contribución.

live-manual incluye dos utilidades que pueden ser de gran

ayuda para los traductores a la hora de encontrar mensajes sin traducir y mensajes difusos. La primera es “make translate”. Esta activa un script que muestra en detalle cuántos mensajes sin traducir hay en cada fichero .po. La segunda, “make fixfuzzy”, sólo actúa sobre los mensajes difusos pero ayuda a encontrarlos y corregirlos uno por uno.

70 Hay que tener en cuenta que aunque estas utilidades pueden ser de gran ayuda para traducir en la línea de comandos, se recomienda el uso de una herramienta especializada como por ejemplo *poedit*. Además, es una buena idea leer la documentación de *debian* sobre localización (l10n) y, específicamente para *live-manual*, las **Directrices para los traductores**.

71 **Nota:** Se puede utilizar `make clean` para limpiar el árbol git antes de enviar los cambios. Este paso no es obligatorio, gracias al fichero `.gitignore`, pero es una buena práctica para evitar enviar ficheros involuntariamente.

Contribuir al Live Systems Project

2. Acerca del Live Systems Project

2.1 Motivación

2.1.1 Desventajas de los sistemas en vivo actuales

Cuando se comenzó a trabajar en el Live Systems Project, ya existían varios sistemas en vivo disponibles basados en la distribución Debian y todos hacían un buen trabajo. Desde la perspectiva de Debian, la mayoría de estos sistemas tenían alguna de estas desventajas:

- No eran proyectos de Debian y por lo tanto no contaban con soporte desde dentro de Debian.
- Mezclaban paquetes de diferentes versiones, por ejemplo **testing** y **unstable**.
- Solamente soportaban la arquitectura i386.
- Modificaban el comportamiento y/o la apariencia de los paquetes, eliminando contenido para reducirlos de tamaño.
- Incluían paquetes de fuera del archivo de Debian.
- Utilizaban kernels personalizados con parches que no eran parte de Debian.
- Eran demasiado lentos, debido a su gran tamaño, para ser utilizados como sistemas de rescate.
- No disponían de diferentes medios de instalación, como CDs, DVDs, llaves USB o imágenes de arranque por red netboot.

2.1.2 El porqué de crear un sistema en vivo propio.

Debian es el Sistema Operativo Universal: Debian

tiene un sistema en vivo para mostrar y representar el auténtico y verdadero Debian con las siguientes ventajas fundamentales:

- Es un subproyecto de Debian. 87
- Refleja el estado (actual) de una versión Debian. 88
- Se ejecuta en tantas arquitecturas como es posible. 89
- Consiste solamente de paquetes Debian sin modificar. 90
- No contiene ningún paquete que no forma parte del archivo de Debian. 91
- Utiliza kernels que provienen de Debian inalterados sin parches adicionales. 92

2.2 Filosofía

2.2.1 Solamente paquetes sin modificación alguna de Debian «main»

Solamente se utilizarán paquetes del repositorio de Debian de la sección «main». La sección non-free no es parte de Debian y por lo tanto no puede ser utilizada de ninguna de las maneras para generar imágenes de sistema oficiales. 95

No se modificará ningún paquete. Siempre que se necesite modificar algo, se hará en coordinación con el correspondiente mantenedor del paquete en Debian. 96

Como excepción, los paquetes del proyecto como son *live-boot*, *live-build* o *live-config*, pueden ser utilizados temporalmente desde el repositorio del proyecto, por razones de desarrollo (por ejemplo para crear instantáneas de pruebas). Estos paquetes serán actualizados en Debian de manera regular. 97

2.2.2 Sin configuración especial para el sistema en vivo

En esta fase, no se creará o instalarán configuraciones alternativas o de ejemplo. Se utilizarán todos los paquetes con su configuración por defecto, tal y como quedan después de una instalación normal de Debian.

Siempre que se necesite una configuración diferente a la de por defecto, se hará en coordinación con el mantenedor del paquete Debian correspondiente.

Se puede emplear un sistema para configurar paquetes que utiliza debconf, permitiendo la personalización de la configuración de los paquetes que van a ser instalados en la imagen en vivo que se genere, pero las **«imágenes en vivo prefabricadas»** solamente utilizarán la configuración por defecto, a menos que sea absolutamente necesario hacer cambios para que funcionen en los sistemas en vivo. Siempre que sea posible, preferimos adaptar los paquetes en el archivo de Debian para que funcionen mejor en un sistema en vivo en lugar de realizar cambios en nuestra cadena de herramientas o en **«las configuraciones de las imágenes prefabricadas»**. Para más información, ver **«Descripción general de la personalización»**.

2.3 Contacto

- **Lista de correo** : El sistema de contacto principal del proyecto es la lista de correo en <https://lists.debian.org/debian-live/>. Se puede enviar un correo a la lista directamente dirigiéndolo a debian-live@lists.debian.org Los archivos históricos de la lista están disponibles en <https://lists.debian.org/debian-live/>.
- **IRC** : Un número importante de usuarios y desarrolladores suele estar presente en el canal #debian-live de

irc.debian.org (OFTC). Por favor, se debe ser paciente cuando se espera una respuesta en el IRC. Si la respuesta no llega, se puede enviar la pregunta a la lista de correos.

- **BTS** : El **«Informes de errores»**.

105

Usuario

107 Instalación

108 3. Instalación

109 3.1 Requisitos

110 Para crear las imágenes en vivo son necesarios los siguientes requisitos:

- 111 • Acceso de superusuario (root)
- 112 • Una versión actualizada de *live-build*
- 113 • Un intérprete de comandos compatible con POSIX, como por ejemplo *bash* o *dash*
- 114 • *debootstrap* o *cdebootstrap*
- 115 • Linux 2.6.x o superior.

116 Tener en cuenta que no es necesario el uso de Debian o una distribución derivada de Debian - *live-build* funcionará en casi cualquier distribución que cumpla con los requisitos anteriores.

117 3.2 Instalación de live-build

118 Se puede instalar *live-build* de varias maneras diferentes:

- 119 • Desde el repositorio Debian
- 120 • A partir del código fuente
- 121 • Usando instantáneas

122 Si se usa Debian, el método recomendado es instalar *live-build* a través del repositorio de Debian.

123 3.2.1 Desde el repositorio Debian.

124 Simplemente instalar *live-build* como cualquier otro paquete:

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

126 3.2.2 A partir del código fuente

127 *live-build* se desarrolla utilizando el sistema de control de versiones Git. En los sistemas basados en Debian se encuentra el paquete *git*. Para ver el último código, ejecutar:

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

Se puede crear e instalar el paquete Debian ejecutando:

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

Si se desea, se podrá instalar cualquiera de los paquetes *.deb* recién creados con el procedimiento anterior, p.ej.

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

También se puede instalar *live-build* directamente en el sistema

ejecutando:

```
# make install
```

y desinstalarlo con:

```
# make uninstall
```

3.2.3 A partir de «instantáneas»

Si no se desea crear o instalar *live-build* a partir del código fuente, se puede usar instantáneas. Estas se generan automáticamente a partir de la última versión de Git y están disponibles en <http://live-systems.org/debian/>.

3.3 Instalación de live-boot y live-config

Nota: No es necesario instalar *live-boot* o *live-config* en el sistema para crear sistemas personalizados en vivo. Sin embargo, eso no causará ningún daño y es útil por motivos de referencia. Si únicamente se desea tener la documentación, es posible instalar los paquetes *live-boot-doc* y *live-config-doc* de forma independiente.

3.3.1 Desde el repositorio Debian.

Tanto *live-boot* como *live-config* están disponibles en el repositorio Debian siguiendo un procedimiento similar al explicado en la [«Instalación de live-build»](#).

3.3.2 A partir del código fuente

Para utilizar el último código fuente a partir de git, se puede seguir el proceso siguiente. Asegurarse de estar familiarizado con los términos mencionados en [«Términos»](#).

- Comprobar el código fuente de *live-boot* y *live-config*

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

Si se desea generar estos paquetes a partir del código fuente, se puede consultar las páginas del manual para más detalles sobre la personalización de *live-boot* y *live-config*.

- Creación de los paquetes .deb de *live-boot* y *live-config*

Se debe crear ya sea en la distribución de destino o en un entorno chroot que contenga la plataforma de destino: es decir, si el objetivo es **jessie** entonces se debe crear usando **jessie**.

Utilizar un programa creador personal como *pbuilder* o *sbuid* si se necesita crear *live-boot* para una distribución de destino diferente del sistema de creación. Por ejemplo, para las imágenes en vivo de **jessie**, crear *live-boot* en un entorno chroot **jessie**. Si la distribución de destino coincide con la distribución actual, se puede crear directamente sobre el sistema de creación con `dpkg-buildpackage` (proporcionada por el paquete *dpkg-dev*):

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

- Utilizar los ficheros .deb generados que proceda

Como *live-boot* y *live-config* son instalados por el sistema de construcción *live-build*, la instalación de esos paquetes en el sistema anfitrión no es suficiente: se debe tratar los .deb generados como si fueran paquetes personalizados. Puesto que el propósito de la construcción de estos paquetes a partir del código fuente es probar cosas nuevas a corto plazo antes de su lanzamiento oficial, seguir las instrucciones de «*Instalar paquetes modificados o de terceros*» para incluir temporalmente los ficheros necesarios en la configuración. En particular, observar que ambos paquetes se dividen en una parte genérica, una parte de documentación y uno o más back-ends. Incluir la parte genérica, sólo uno de los back-ends que coincida con la configuración y opcionalmente, la documentación. Suponiendo que se está construyendo una imagen en vivo en el directorio actual y se han generado todos los .deb para una única versión de los dos paquetes en el directorio superior, estos comandos bash copiarán todos los paquetes necesarios, incluyendo sus back-ends por defecto:

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

3.3.3 A partir de «instantáneas»

Se puede dejar que *live-build* utilice automáticamente las últimas instantáneas de *live-boot* y *live-config* mediante la configuración del repositorio de terceros live-systems.org en el directorio de configuración de *live-build*.

Conceptos básicos

4. Conceptos básicos

Este capítulo contiene una breve descripción del proceso de creación de las imágenes en vivo y las instrucciones para el uso de los tres tipos de imágenes más utilizadas. El tipo de imagen más versátil, *iso-hybrid*, se puede utilizar en una máquina virtual, en medios ópticos u otros dispositivos de almacenamiento USB. En ciertos casos especiales, como se explica más adelante, las imágenes *hdd*, pueden ser las más adecuadas. El capítulo incluye instrucciones detalladas para crear y utilizar una imagen de tipo *netboot*, que es un poco más complicado debido a la configuración necesaria en el servidor. Es un tema ligeramente avanzado para cualquier persona que no esté familiarizada con el arranque en red, pero se incluye aquí porque una vez que se realiza toda la configuración, es una forma muy conveniente para probar y desplegar imágenes de arranque en red local sin la molestia de tratar con los dispositivos de almacenamiento de la imagen.

La sección termina con una rápida introducción al **«arranque desde internet»**, que es, quizás, la manera más rápida de utilizar diferentes imágenes para diferentes propósitos, cambiando de una a otra según las necesidades, utilizando internet como medio.

A lo largo de todo el capítulo se hace a menudo referencia al nombre de las imágenes producidas por defecto por *live-build*. Si se **«descarga una imagen ya creada»** el nombre puede variar.

4.1 ¿Qué es un sistema en vivo?

Por lo general, un sistema en vivo se refiere a un sistema operativo que arranca en un equipo desde un medio extraíble,

como un CD-ROM, dispositivo USB, o desde una red, listo para usar sin ningún tipo de instalación en la unidad de costumbre, con configuración automática en tiempo de ejecución (Ver **«Términos»**).

Con los sistemas en vivo, es un sistema operativo, creado para una de las arquitecturas soportadas (actualmente amd64 y i386). Se compone de las siguientes partes:

- **Imagen del kernel de Linux** , normalmente llamada `vmlinuz*`
- **Imagen del Disco RAM inicial (initrd)** : Un Disco RAM configurado para el arranque de Linux, que incluya los módulos posiblemente necesarios para montar la imagen del sistema y algunos scripts para ponerlo en marcha.
- **Imagen del sistema** : La imagen del sistema de ficheros raíz. Por lo general, se utiliza un sistema de ficheros comprimido SquashFS para reducir al mínimo el tamaño de la imagen en vivo. Hay que tener en cuenta que es de sólo lectura. Por lo tanto, durante el arranque del sistema en vivo se utiliza un disco RAM y un mecanismo de «unión» que permite escribir ficheros en el sistema en funcionamiento. Sin embargo, todas las modificaciones se perderán al apagar el equipo a menos que se use de modo opcional la persistencia (ver **«Persistencia»**).
- **Gestor de arranque** : Una pequeña pieza de código diseñada para arrancar desde el medio de almacenamiento escogido, posiblemente mostrando un menú o un indicador de arranque para permitir la selección de opciones/configuración. Carga el kernel de Linux y su initrd para funcionar con un sistema de ficheros asociado. Se pueden usar soluciones diferentes, dependiendo del medio de almacenamiento de destino y el formato del sistema de ficheros que contenga los componentes mencionados anteriormente: *isolinux* para arrancar desde un CD o DVD

en formato ISO9660, syslinux para arrancar desde el disco duro o unidad USB desde una partición VFAT, extlinux para formatos ext2/3/4 y particiones btrfs, pxelinux para arranque de red PXE, GRUB para particiones ext2/3/4 , etc.

169 Se puede utilizar *live-build* para crear la imagen del sistema a partir de ciertas especificaciones, incluir un kernel de Linux, su initrd y un gestor de arranque para ponerlos en funcionamiento, todo ello en un formato que depende del medio de almacenamiento elegido (imagen ISO9660, imagen de disco, etc.)

170 4.2 Descarga de imágenes prefabricadas

171 Si bien el objetivo de este manual es el desarrollo y la construcción de imágenes en vivo propias, puede que simplemente se desee probar una de nuestras imágenes prefabricadas, ya sea como una iniciación a su uso o como paso previo a la construcción de imágenes personalizadas. Estas imágenes están construidas utilizando nuestro [«repositorio git live-images»](https://github.com/live-images) y las versiones estables oficiales se publican en [«https://www.debian.org/CD/live/»](https://www.debian.org/CD/live/). Además, las versiones antiguas y las futuras, así como las imágenes no oficiales que contienen firmware y drivers no libres están disponibles en [«http://live-systems.org/cdimage/release/»](http://live-systems.org/cdimage/release/).

172 4.3 Uso del servicio de creación de imágenes web

173 Como un servicio a la comunidad, se ofrece una interfaz web de construcción de imágenes en vivo en [«http://live-build.debian.net/»](http://live-build.debian.net/). Este sitio se mantiene en base al mejor esfuerzo. Es decir, aunque nos esforzamos por mantenerlo al día y de que esté operativo en todo momento, así como de emitir anuncios

de interrupciones importantes en el servicio, no podemos garantizar un 100% de disponibilidad o una creación de imágenes rápida, y el servicio de vez en cuando puede tener problemas que tardan algún tiempo en resolverse. Si se tiene problemas o preguntas acerca de este servicio, ponerse [«en contacto»](#) con nosotros, proporcionando el enlace a la página dónde se recoge la información pertinente a la imagen.

4.3.1 Uso y advertencias del servicio de creación de imágenes web

175 La interfaz web actualmente no puede prevenir el uso de combinaciones de opciones no válidas, y en particular, cuando el cambio de una opción que normalmente (es decir, utilizando *live-build* directamente) cambiaría los valores predeterminados de otras opciones que figuran en el formulario web, el constructor web no cambia estos valores predeterminados. En particular, si se cambia `--architectures` del valor por defecto `i386` a `amd64`, se debe cambiar la opción correspondiente `--linux-flavours` del valor por defecto `486` a `amd64`. Ver la página de manual de *lb_config* para para más detalles sobre la versión de *live-build* instalada en el constructor web. El número de versión de *live-build* aparece en la parte inferior de la página web del servicio de creación de imágenes.

176 El tiempo de creación de la imagen mostrado en la web es sólo una estimación aproximada y puede no reflejar con exactitud la duración que la construcción de la imagen realmente necesita. Tampoco se actualiza esta estimación una vez mostrada. Hay que tener un poco de paciencia. No volver a recargar la página, ya que esto puede volver a lanzar una nueva creación de otra imagen con los mismos parámetros. [«Ponerse en contacto»](#) con nosotros si no se recibe la notificación de que la imagen está terminada una vez que se esté seguro de que se ha esperado lo suficiente y verificado que la notificación por correo electrónico

no ha ido a parar a la bandeja de spam.

El servicio web está limitado en el tipo de imágenes que se pueden construir. Esto lo hace simple y a la vez eficiente de usar y mantener. Si se desea realizar personalizaciones que no se contemplan en la interfaz web, en el resto de este manual se explica cómo crear imágenes propias con *live-build*.

4.4 Primeros pasos: creación de una imagen ISO híbrida

Independientemente del tipo de imagen, cada vez se tendrá que realizar los mismos pasos básicos para construir una imagen. Como primer ejemplo, crear un directorio de trabajo, cambiar a ese directorio y ejecutar la siguiente secuencia de comandos *live-build* para crear una imagen ISO híbrida básica que contiene sólo el sistema por defecto de Debian sin X.org. Es adecuada para grabarla en un CD o DVD y también para copiarla en un dispositivo USB.

El nombre del directorio de trabajo es indiferente, pero si se da un vistazo a los ejemplos utilizados en *live-manual*, es una buena idea utilizar un nombre que ayude a identificar la imagen con la que está trabajando en cada directorio, especialmente si se está trabajando o experimentando con distintos tipos de imágenes. En este caso, vamos a construir un sistema utilizando los valores por defecto, así que lo vamos a llamar, por ejemplo, *live-default*.

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

Entonces, ejecutar el comando `lb config`. Esto creará una jerarquía «config/» en el directorio actual que será usada por otros comandos:

```
$ lb config
```

Al no pasar ningún parámetro a estos comandos, se utilizarán todas las opciones por defecto. Ver «El comando `lb config`» para más detalles.

Ahora que existe una jerarquía «config/», se puede crear la imagen con el comando `lb build`:

```
# lb build
```

Este proceso puede llevar un tiempo, dependiendo de la velocidad del ordenador y de la conexión de red. Cuando haya terminado, debería haber un fichero `live-image-i386.hybrid.iso` listo para ser usado en el directorio actual.

Nota: Si se está construyendo en un sistema amd64 el nombre de la imagen resultante será `live-image-amd64.hybrid.iso`. Tener en cuenta esta convención a lo largo del manual.

4.5 Usar una imagen ISO híbrida

Después de construir o descargar una imagen ISO híbrida, las cuales se pueden obtener en <https://www.debian.org/CD/live/>, el siguiente paso habitual es preparar el medio de almacenamiento, ya sea medios ópticos CD-R(W) o DVD-R(W) o llaves USB.

4.5.1 Grabar una imagen ISO en un medio físico.

Grabar una imagen ISO es fácil. Simplemente instalar *xorriso* y

usarlo desde el intérprete de comandos para grabar la imagen. Por ejemplo:

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

4.5.2 Copiar una imagen ISO híbrida a un dispositivo USB

Las imágenes ISO preparadas con xorriso, pueden sencillamente copiarse a una llave USB con la orden `cp` o con un programa equivalente. Insertar una llave USB con un tamaño suficiente para la imagen y determinar qué dispositivo es, al cual nos referiremos de ahora en adelante como `${USBSTICK}`. Este nombre de «dispositivo» se refiere a la llave entera como por ejemplo `/dev/sdb` y ¡No a una partición como `/dev/sdb1`! Se puede encontrar el nombre del dispositivo correcto mirando la salida de `dmesg` después de conectar la llave, o mejor aún, ejecutando `ls -l /dev/disk/by-id`.

Cuando se esté seguro de tener el nombre del dispositivo correcto, usar la orden `cp` para copiar la imagen a la llave. **¡Esto borrará de forma definitiva cualquier contenido previo en la llave!**

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

Nota: El comando `sync` se utiliza para asegurarse de que todos los datos, que el kernel almacena en la memoria mientras se copia la imagen, se escriben en la llave USB.

4.5.3 Usar el espacio libre en el dispositivo USB

Después de copiar la `live-image-i386.hybrid.iso` en una llave USB, la primera partición del dispositivo será utilizada por el sistema en vivo. Si se desea utilizar el espacio libre, se puede utilizar un programa de particionado como *gparted* o *parted* para crear una partición nueva en la llave.

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

Después de crear la partición, dónde `${PARTITION}` es el nombre de la partición, por ejemplo `/dev/sdb2` se tiene que crear un sistema de ficheros en él. Una opción posible sería `ext4`.

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

Nota: Si se desea usar el espacio extra con Windows, según parece, ese sistema operativo no puede acceder normalmente a otra partición más que a la primera. Se han comentado algunas soluciones a este problema en nuestra [lista de correo](#) pero según parece no hay una solución fácil.

Recordar: Cada vez que se instale una nueva `live-image-i386.hybrid.iso` en el dispositivo, todos los datos del dispositivo se perderán debido a que la tabla de particiones se sobrescribe con el contenido de la imagen, así pues, realizar primero una copia de seguridad de la partición para poder restaurarla tras actualizar la imagen en vivo.

4.5.4 Arrancar el medio en vivo

207 La primera vez que se arranque desde el medio de almacenamiento en vivo, ya sea CD, DVD, llave USB, o de arranque en red PXE, primero puede ser necesario algún tipo de configuración en la BIOS de la máquina. Dado que las BIOS varían mucho en sus características y combinaciones de teclas, no se puede entrar en el tema en profundidad aquí. Algunas BIOS proporcionan una tecla para abrir un menú de dispositivos de arranque que es la manera más fácil de hacerlo si se encuentra disponible en el sistema. De lo contrario, se tiene que entrar en el menú de configuración de la BIOS y cambiar el orden de arranque y colocar el dispositivo de arranque del sistema en vivo antes que el dispositivo de arranque habitual.

208 Una vez que se haya arrancado desde el medio de almacenamiento, se accede a un menú de arranque. Si se pulsa la tecla «enter», el sistema arrancará usando el modo por defecto Live y las opciones predeterminadas. Para obtener más información acerca de las opciones de arranque, ver la opción «help» del menú y también las páginas del manual de *live-boot* y *live-config* que se encuentran en el sistema en vivo.

209 Suponiendo que se ha seleccionado Live y arrancado una imagen en vivo por defecto con escritorio gráfico, después de que los mensajes de arranque hayan pasado, se habrá iniciado automáticamente una sesión como usuario user y se verá el escritorio preparado para ser usado. Si se ha arrancado una imagen sólo de consola como por ejemplo **«las imágenes prefabricadas»**, standard o rescue se habrá iniciado automáticamente una sesión como usuario user y se verá el cursor del intérprete de comandos preparado para ser usado.

206

4.6 Usar una máquina virtual para pruebas

210

Ejecutar las imágenes en vivo en una máquina virtual (VM) puede ser un gran ahorro de tiempo para su desarrollo. Esto no está exento de advertencias:

- Para ejecutar una máquina virtual se requiere tener suficiente memoria RAM para el sistema operativo huésped y el anfitrión y se recomienda una CPU con soporte de hardware para la virtualización. 212
- Existen algunas limitaciones inherentes a la ejecución en una máquina virtual, por ejemplo, rendimiento de video pobre o limitada gama de hardware emulado. 213
- Cuando se desarrolla para un hardware específico, no hay sustituto mejor que el propio hardware. 214
- A veces hay errores causados únicamente por la ejecución en una máquina virtual. En caso de duda, probar la imagen directamente en el hardware. 215

Siempre que se pueda trabajar dentro de estas limitaciones, mirar que software VM hay disponible y elegir uno que sea adecuado según las necesidades. 216

4.6.1 Probar una imagen ISO con QEMU

217

La máquina virtual más versátil en Debian es QEMU. Si el procesador tiene soporte de hardware para virtualización, utilizar el paquete *qemu-kvm*. En la descripción del paquete *qemu-kvm* se enumera brevemente la lista de requisitos. 218

En primer lugar, instalar *qemu-kvm* si el procesador lo soporta. Si no es así, instalar *qemu*, en cuyo caso el nombre del programa será *qemu* en vez de *kvm* en los siguientes ejemplos. El paquete *qemu-utils* también es útil para la creación de imágenes virtuales de disco con *qemu-img*. 219

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

Arrancar una imagen ISO es sencillo:

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

Consultar las páginas del manual para más detalles.

4.6.2 Probar una imagen ISO con VirtualBox

Para probar una imagen ISO con *virtualbox*:

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

Crear una nueva máquina virtual, cambiar la configuración de almacenamiento para utilizar *live-image-i386.hybrid.iso* como dispositivo CD/DVD y arrancar la máquina.

Nota: Para probar los sistemas en vivo con soporte X.org en *virtualbox*, se puede incluir el paquete del driver de VirtualBox X.org, *virtualbox-guest-dkms* y *virtualbox-guest-x11*, en la configuración de *live-build*. De lo contrario, la resolución se limita a 800x600.

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

Para que el paquete *dkms* funcione, hace falta tener instalados también los *kernel-headers* para la variante del kernel utilizado. En lugar de enumerar manualmente el paquete *linux-headers* correspondiente en la lista de paquetes creados anteriormente, *live-build* puede seleccionarlo automáticamente.

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

4.7 Construir y utilizar una imagen HDD

Crear una imagen HDD es similar a una de tipo ISO híbrida en todos los aspectos, con la diferencia de que hay que especificar *-b hdd* y de que el nombre de la imagen final es *live-image-i386.img* que no se puede copiar en medios ópticos. Es adecuada para el arranque desde dispositivos USB, discos duros USB y otros sistemas de almacenamiento portable. Normalmente, se puede utilizar para este propósito una imagen ISO híbrida, pero es posible que la BIOS no maneje adecuadamente las imágenes híbridas, entonces es mejor utilizar una imagen *hdd*.

Nota: si se ha creado una imagen ISO híbrida con el ejemplo anterior, se tendrá que limpiar el directorio de trabajo con el comando *lb clean* (ver [El comando lb clean](#)):

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

Ejecutar el comando *lb config* como antes pero esta vez especificando el tipo de imagen HDD:

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

Crear ahora la imagen con el comando `lb build`:

```
# lb build
```

Cuando termine el proceso de creación, debe haber un fichero llamado `live-image-i386.img` en el directorio actual .

La imagen binaria generada contiene una partición VFAT y el gestor de arranque `syslinux`, lista para ser copiada directamente en un dispositivo USB. De nuevo, dado que utilizar una imagen HDD es igual a usar una imagen ISO híbrida en un USB, seguir las instrucciones de [«Usar una imagen ISO híbrida»](#) con la diferencia del nombre, `live-image-i386.img` en lugar de `live-image-i386.hybrid.iso`.

Del mismo modo, para probar una imagen HDD con Qemu, instalar `qemu` como se describe más arriba en [«Probar una imagen ISO con QEMU»](#). A continuación, ejecutar `kvm` o `qemu`, según qué versión necesita el sistema anfitrión y especificando `live-image-i386.img` como primer disco duro.

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

4.8 Creación de una imagen de arranque en red

La siguiente secuencia de comandos creará una imagen de arranque en red básica que contendrá el sistema por defecto de Debian sin X.org. Se puede usar para el arranque en red.

Nota: si se ha seguido alguno de los ejemplos anteriores, se

tendrá que limpiar el directorio de trabajo con el comando `lb clean`:

```
# lb clean
```

En este caso concreto, un `lb clean --binary` no sería suficiente para eliminar las etapas necesarias. La razón de esto es que en las configuraciones de arranque en red, se debe utilizar una configuración `initramfs` diferente que *live-build* ejecuta automáticamente al crear imágenes `netboot`. Ya que la creación del `initramfs` pertenece a la etapa `chroot`, realizar el cambio a `netboot` en un directorio de construcción ya existente significa reconstruir la etapa `chroot` también. Por lo tanto, se tiene que ejecutar un `lb clean` (que también eliminará la etapa `chroot`).

Ejecutar el comando `lb config` de la siguiente manera para configurar la imagen de arranque en red:

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

A diferencia de las imágenes ISO y HDD, el sistema de arranque en red en sí mismo no envía la imagen del sistema de ficheros al cliente, por eso los ficheros se deben enviar mediante NFS. Con `lb config` se puede seleccionar diferentes sistemas de ficheros en red. Las opciones `--net-root-path` y `--net-root-server` especifican la ubicación y el servidor, respectivamente, del servidor NFS en el que se encuentra la imagen del sistema de ficheros en el arranque. Se debe asegurar que estos se ajustan a los valores adecuados para la red y el servidor deseados.

Crear ahora la imagen con el comando `lb build`:

```
# lb build
```

En un arranque en red, el cliente ejecuta una pequeña pieza de software que generalmente se encuentra en la EPROM de la tarjeta Ethernet. Este programa envía una solicitud de DHCP para obtener una dirección IP e información sobre qué hacer a continuación. Por lo general, el siguiente paso es conseguir un gestor de arranque de alto nivel a través del protocolo TFTP. Este gestor podría ser PXELINUX, GRUB, o incluso arrancar directamente un sistema operativo como Linux.

Por ejemplo, si se descomprime el archivo generado `live-image-i386.netboot.tar` en el directorio `/srv/-debian-live`, se verá la imagen del sistema de ficheros en `live/filesystem.squashfs` y el kernel, `initrd` y el gestor de arranque `pxelinux` en `tftpbboot/`.

Ahora se debe configurar tres servicios en el servidor para el arranque en red: el servidor DHCP, el servidor TFTP y el servidor NFS.

4.8.1 Servidor DHCP

Hay que configurar el servidor DHCP de red para asegurar que proporciona una dirección IP al cliente, y para anunciar la ubicación del gestor de arranque PXE.

He aquí un ejemplo que puede servir de inspiración. Fue escrito para el servidor ISC DHCP `isc-dhcp-server` en su fichero de configuración `/etc/dhcp/dhcpd.conf`:

252

```
# /etc/dhcp/dhcpd.conf - fichero de configuración para 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 Servidor TFTP

Se encarga de suministrar el kernel y el Disco RAM inicial para el sistema.

Se debe instalar el paquete `tftpd-hpa`. Este servidor podrá suministrar todos los ficheros contenidos de un directorio raíz, normalmente `/srv/tftp`. Para permitirle que pueda servir los ficheros de `/srv/debian-live/tftpbboot`, se debe ejecutar el siguiente comando con privilegios de superusuario:

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

y escribir el directorio del nuevo servidor tftp cuando sea requerido.

4.8.3 Servidor NFS

Una vez el equipo cliente ha descargado y arrancado el kernel de Linux junto a su initrd, intentará montar el sistema de archivos de la imagen en vivo a través de un servidor NFS.

Se debe instalar el paquete *nfs-kernel-server*.

Entonces, se debe hacer que la imagen del sistema de archivos esté disponible a través de NFS añadiendo una línea como la siguiente para */etc/exports*:

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

e informar al servidor NFS sobre esta nueva exportación con el siguiente comando:

```
# exportfs -rv
```

La configuración de estos tres servicios puede ser un poco difícil. Será necesario un poco de paciencia para conseguir que todos ellos funcionen juntos. Para obtener más información, ver el wiki de syslinux en <http://www.syslinux.org/wiki/index.php/PXELINUX> o la sección sobre TFTP Net Booting del Manual del Instalador de Debian en <http://d-i.alioth.debian.org/manual/es.i386/ch04s05.html> Esto puede ser útil, ya que sus procesos son muy similares.

4.8.4 Cómo probar el arranque en red

La creación de una imagen de arranque en red es sencilla con

live-build, pero probar las imágenes en máquinas físicas puede ser un proceso mucho más lento.

Para hacer nuestra vida más fácil, se puede utilizar la virtualización.

4.8.5 Qemu

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

Se debe editar el fichero */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
```

Obtener o crear un grub-floppy-netboot.

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

4.9 Arrancar desde internet

Arrancar desde internet, o Webbooting, es una manera muy adecuada de descargar y arrancar sistemas en vivo utilizando internet como medio, ya que hay muy pocos requisitos para arrancar desde internet utilizando webbooting. Por un lado, se necesita un medio en vivo con un gestor de arranque, un disco ram inicial y un kernel. Por otro lado, un servidor web para almacenar los ficheros squashfs que contienen el sistema de ficheros.

4.9.1 Conseguir los ficheros para arrancar desde internet

Como de costumbre, se puede construir las imágenes uno mismo o utilizar alguna de las imágenes prefabricadas, disponibles en la página web del proyecto <http://live-systems.org/>. Utilizar las imágenes prefabricadas es muy práctico para hacer pruebas hasta que se está seguro de cuales son las necesidades reales. Si ya se ha construido una imagen, los ficheros necesarios para el arranque desde internet se encuentran en el directorio `binary/live/`. Los ficheros se llaman `mlinuz`, `initrd.img` y `filesystem.squashfs`.

También es posible extraer los ficheros de una imagen iso ya existente. Para ello, hay que montar la imagen de la siguiente manera:

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

Los ficheros se encuentran en el directorio `live/`. En este caso concreto, sería `/mnt/live/`. Este método tiene la desventaja de que es necesario ser root para poder montar la imagen. Sin embargo, tiene la ventaja de que es fácil hacerlo con un script y por lo tanto, fácil de automatizar.

Pero, sin duda alguna, la forma más fácil de extraer los ficheros de una imagen iso y subirlos al servidor web al mismo tiempo, es utilizando el midnight commander o `mc`. Si se tiene el paquete `genisoimage` instalado, este administrador de ficheros de dos paneles permite examinar el contenido de un archivo iso en un panel y subir los ficheros a través de ftp en el otro panel. A pesar de que este método requiere un trabajo manual, no requiere privilegios de root.

4.9.2 Arrancar imágenes webboot

Aunque algunos usuarios pueden preferir la virtualización para probar el arranque desde internet, en este caso se utiliza hardware real para ilustrar el caso de uso que se explica a continuación y que debe considerarse sólo como un ejemplo.

Para arrancar una imagen webboot es suficiente copiar los elementos mencionados anteriormente, es decir, `mlinuz` y `initrd.img` en una llave usb dentro de un directorio llamado `live/` e instalar `syslinux` como gestor de arranque. Entonces, arrancar desde la llave usb y teclear `fetch=URL/RUTA/AL/-FICHERO` en las opciones de arranque. *live-boot* se encargará de descargar el archivo `squashfs` y almacenarlo en la memoria ram. De este modo, es posible utilizar el sistema de ficheros comprimido descargado como si fuera un sistema en vivo normal. Por ejemplo:

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

Caso de uso: Se tiene dos archivos `squashfs` almacenados en un servidor web, uno que contiene un escritorio completo, como `gnome`, y uno de rescate. Si se necesita un entorno gráfico para una máquina, se puede insertar la llave usb y arrancar desde internet la imagen `gnome`. Si se necesitan las herramientas de rescate del segundo tipo de imagen, quizás para otra máquina, se puede arrancar desde internet la de rescate.

Descripción general de las herramientas

5. Descripción general de las herramientas

Este capítulo contiene una descripción general de las tres herramientas principales utilizadas en la creación de sistemas en vivo: *live-build*, *live-boot* y *live-config*.

5.1 El paquete live-build

live-build es una colección de scripts para generar los sistemas en vivo. A estos scripts también se les conoce como «comandos».

La idea detrás de *live-build* es ser un marco que utiliza un directorio de configuración para automatizar completamente y personalizar todos los aspectos de la creación de una imagen de un sistema en vivo.

Muchos conceptos son similares a los utilizados para crear paquetes Debian con *debhelper*:

- Los scripts tienen una ubicación central para la configuración de su funcionamiento. En *debhelper*, éste es el subdirectorio *debian/* de un árbol de paquetes. Por ejemplo, *dh_install* buscará, entre otros, un fichero llamado *debian/install* para determinar qué ficheros deben existir en un paquete binario en particular. De la misma manera, *live-build* almacena toda su configuración bajo un subdirectorio *config/*.
- Los scripts son independientes - es decir, siempre es seguro ejecutar cada comando.

A diferencia de *debhelper*, *live-build* contiene las herramientas para crear un directorio de configuración en esqueleto. Esto podría ser considerado como similar a herramientas tales

como *dh-make*. Para obtener más información acerca de estas herramientas, seguir leyendo, ya que el resto de esta sección trata sobre los cuatro comandos más importantes. En interesante notar que están precedidos por *lb* que es una función genérica para todos los comandos de *live-build*.

- **lb config** : Responsable de inicializar un directorio de configuración para la creación de un sistema en vivo. Ver [«El comando lb config»](#) para más información. 306
- **lb build** : Responsable de iniciar la creación de un sistema en vivo. Ver [«El comando lb build»](#) para más información. 307
- **lb clean** : Responsable de la eliminación de partes de la creación de un sistema en vivo. Ver [«El comando lb clean»](#) para más información. 308

5.1.1 El comando lb config 309

Como se comentó en [«live-build»](#), los scripts que componen *live-build* obtienen su configuración gracias al comando *source* desde un único directorio llamado *config/*. Como la creación de este directorio a mano sería largo y propenso a errores, se puede utilizar el comando *lb config* para crear el esqueleto de directorios de configuración inicial. 310

Ejecutar *lb config* sin argumentos crea el subdirectorio *config/* que se completa con algunas opciones por defecto en ficheros de configuración y dos árboles de subdirectorios en forma de esqueleto llamados *auto/* y *local/*. 311

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

Utilizar *lb config* sin ningún argumento sería conveniente 312

para los usuarios que necesitan una imagen muy básica, o que tienen intención de proporcionar, más adelante, una configuración más completa a través de `auto/config` (ver [«Gestionar una configuración»](#) para más detalles).

Normalmente, se tendrá que especificar algunas opciones. Por ejemplo, para especificar que gestor de paquetes se desea utilizar durante la construcción de la imagen:

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

Es posible especificar muchas opciones, tales como:

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

Una lista completa de opciones está disponible en la página de manual `lb_config`.

5.1.2 El comando `lb build`

El comando `lb build` lee la configuración del directorio `config/`. A continuación, ejecuta los comandos de nivel inferior necesarios para crear el sistema en vivo.

5.1.3 El comando `lb clean`

El comando `lb clean` es el encargado de eliminar varias partes de una creación de forma que las creaciones posteriores puedan comenzar de forma limpia. Por defecto se eliminan las etapas `chroot`, `binary` y `source` pero se deja el caché intacto. Además, se pueden limpiar etapas de forma individual. Por

ejemplo, si se han realizado cambios que sólo afectan a la etapa `binary`, se debe usar `lb clean --binary` antes de crear una nueva `binary`. Si los cambios modifican el `bootstrap` y/o los cachés de paquetes, por ejemplo, cambios en las opciones `--mode`, `--architecture` o `--bootstrap`, se debe utilizar `lb clean --purge`. Ver el manual de `lb_clean` para una lista detallada de todas sus opciones.

5.2 El paquete `live-boot`

live-boot es una colección de scripts que proporcionan ganchos (hooks) para *initramfs-tools*, que sirve para generar un `initramfs` capaz de arrancar sistemas en vivo, tales como los creados por *live-build*. Esto incluye imágenes ISO, archivos comprimidos en formato tar para el arranque en red, e imágenes para llaves USB.

En el momento del arranque, buscará en los medios de almacenamiento de sólo lectura un directorio `/live/` donde se encuentra un sistema de ficheros raíz (a menudo una imagen del sistema de ficheros comprimidos como `squashfs`). Si lo encuentra, creará un entorno de escritura, utilizando `aufs`, para que arranquen los sistemas tipo Debian.

Se puede encontrar más información sobre `ramfs` inicial en Debian en el Manual del kernel Debian Linux en <http://kernel-handbook.alioth.debian.org/> concretamente en el capítulo sobre `initramfs`.

5.3 El paquete `live-config`

live-config consiste en una serie de scripts que se ejecutan en el arranque después de *live-boot* para configurar el sistema en vivo de forma automática. Se ocupa de tareas como la creación del nombre del equipo (`hostname`), las variantes

locales y la zona horaria, crear el usuario en vivo, la inhibición de trabajos de cron y el inicio de sesión automático del usuario en vivo.

Gestionar una configuración

6. Gestionar una configuración

Este capítulo explica como gestionar una configuración para crear un sistema en vivo desde el principio, pasando por sucesivas versiones tanto de la herramienta *live-build* como de la imagen del sistema en vivo propiamente dicha.

6.1 Gestionar cambios en la configuración

Las configuraciones en vivo rara vez son perfectas al primer intento. Puede estar bien pasar opciones a `lb config` en la línea de comandos para realizar una construcción única, pero es más típico revisar esas opciones y construir de nuevo hasta quedar satisfecho. Para gestionar estos cambios, se pueden utilizar scripts `auto` que garanticen que la configuración se mantiene en un estado coherente.

6.1.1 ¿Por qué utilizar scripts `auto`? ¿Qué hacen?

El comando `lb config` almacena las opciones que se le pasan en ficheros en el directorio `config/*`, junto con muchas otras opciones que figuran en sus valores predeterminados. Si se ejecuta `lb config` una vez más, no restablecerá ninguna opción que se estableció como por defecto en función de las opciones iniciales. Así, por ejemplo, si se ejecuta `lb config` otra vez con un nuevo valor para `--binary-images`, todas las opciones que se establecieron como predeterminadas según la opción anterior ya no pueden funcionar con la nueva. Estos ficheros tampoco están destinados a ser leídos o editados. Almacenan valores para más de cien opciones, y nadie es capaz de ver las opciones que se especificó realmente. Y por último, si se ejecuta `lb config` y a continuación se actualiza

live-build y hay alguna opción que cambió de nombre, `config/*` todavía tendrá variables con las opciones viejas que ya no son válidas.

Por todas estas razones, los scripts `auto/*` nos hacen la vida más fácil. Son simples envoltorios para los comandos `lb config`, `lb build` y `lb clean` diseñados para ayudar a gestionar una configuración. El script `auto/config` contiene el comando `lb config` con todas las opciones que se desea, el script `auto/clean` elimina los ficheros que contienen variables de configuración y el fichero `auto/build` crea un `build.log` de cada creación. Cada uno de estos scripts se ejecuta automáticamente cada vez que se ejecuta la orden `lb` correspondiente. Mediante el uso de estos scripts, la configuración es más fácil de leer y se mantiene internamente coherente de una revisión a la siguiente. Además, será mucho más fácil identificar y corregir las opciones que necesitan cambiarse tras actualizar *live-build* y leer la documentación actualizada.

6.1.2 Usar scripts `auto` de ejemplo

Para mayor comodidad, *live-build* incluye scripts `auto` de ejemplo que se pueden copiar y editar. Iniciar una nueva configuración por defecto y a continuación, copiar los ejemplos:

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

Editar `auto/config`, añadiendo las opciones que se desee. Por ejemplo:

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

Ahora, cada vez que se utilice `lb config`, `auto/config` reiniciará la configuración basándose en estas opciones. Cuando se desee realizar cambios, se deben editar las opciones en este fichero en lugar de pasarlas a `lb config`. Cuando se utilice `lb clean`, `auto/clean` limpiará los ficheros en `config/*` junto a los otros productos de construcción. Y, por último, cuando se utilice `lb build`, `auto/build` creará un log del proceso de construcción llamado `build.log`.

Nota: Aquí se utiliza `noauto`, un parámetro especial para suprimir otra llamada a `auto/config`, evitando así una repetición infinita. Asegurarse de no eliminarlo accidentalmente al hacer cambios en el fichero. Tener cuidado al dividir el comando `lb config` en varias líneas para facilitar la lectura, como se muestra en el ejemplo anterior, ya que no debe olvidarse la barra invertida (al final de cada línea que sigue en la siguiente.

6.2 Clonar una configuración publicada a través de Git

Utilizar la opción `lb config --config` para clonar un repositorio Git que contenga una configuración de un sistema en vivo. Si se desea basar la configuración en una mantenida por el Live Systems Project, visitar el repositorio en <http://live-systems.org/gitweb/> con el nombre `live-images` bajo el título `Packages`. Este repositorio contiene las configuraciones que se utilizan para las **imágenes prefabricadas**

Por ejemplo, para construir una imagen de rescate, utilizar el repositorio `live-images` de la siguiente manera:

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

Editar `auto/config` y cualquier otra cosa que se necesite en el árbol `config` para adaptarlo a las propias necesidades. Por ejemplo, las imágenes prefabricadas con paquetes de la sección `non-free` se crean simplemente añadiendo `--archive-areas "main contrib non-free"`.

Si se desea, se puede definir un método abreviado en la configuración de Git, añadiendo lo siguiente al fichero `$(HOME)/.gitconfig`:

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

Esto permite utilizar `lso`: en cualquier lugar en que se tenga que especificar la dirección de un repositorio git de `live-systems.org`. Si se omite el sufijo `.git`, comenzar una nueva imagen con esta configuración es tan fácil como:

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

Clonar el repositorio `live-images` completo copiará todas las configuraciones utilizadas para varias imágenes. Si se quiere construir una imagen diferente después de haber terminado con la primera, cambiar a otro directorio y de nuevo,

y opcionalmente, hacer los cambios necesarios para adaptarlo según las necesidades.

354

En cualquier caso, recordar que cada vez que se tiene que construir una imagen hay que hacerlo como superusuario: `lb build`

Personalización de contenidos

7. Descripción general de la personalización.

Este capítulo presenta un resumen de las diversas formas en que se puede personalizar un sistema en vivo.

7.1 Configuración en el momento de la creación vs en el momento del arranque

Las opciones de configuración de un sistema Debian Live se pueden dividir en opciones que se aplican en el momento de la creación de la imagen del sistema en vivo y opciones que se tendrán en cuenta cuando el sistema en vivo arranque. Estas últimas se pueden dividir a su vez en opciones que se ejecutan en la etapa inicial del arranque, aplicadas por el paquete *live-boot*, y otras que se llevarán a cabo posteriormente y que son aplicadas por el paquete *live-config*. Cualquier opción en tiempo de arranque puede ser modificada por el usuario indicándola en los parámetros de arranque del kernel mediante el indicador de arranque. La imagen puede ser creada por defecto con los parámetros de arranque adecuados, de manera que los usuarios solamente tendrán que arrancar el sistema en vivo, directamente, sin necesidad de especificar ninguna opción adicional, ya que las opciones por defecto serán las adecuadas. En particular, la opción `lb --bootappend-live` permite introducir cualquier parámetro del kernel para el sistema en vivo, como pueden ser la persistencia, distribución del teclado, zonas horarias, etc. Ver un ejemplo en [«Personalización de las variantes locales e idioma»](#).

Las opciones de configuración en tiempo de creación se describen en la página de manual del comando `lb config`. Las opciones en tiempo de arranque se describen en las

páginas de manual de los paquetes *live-boot* y *live-config*. Aunque los paquetes *live-boot* y *live-config* se instalan en el sistema en vivo que se está creando, también se recomienda que sean instalados en el sistema huésped, que se utiliza para crear la imagen del sistema en vivo, con el fin de facilitar la referencia cuando se trabaja en una configuración. No hay ningún problema en hacerlo, ya que ninguno de los scripts que contiene el sistema huésped será ejecutado, a menos que se configure el sistema huésped como sistema en vivo.

7.2 Etapas de la creación

El proceso de creación de la imagen está dividido en etapas en las que se aplican diferentes personalizaciones en cada una de ellas. La primera etapa que se ejecuta es la etapa **bootstrap**. Esta fase inicial crea y rellena el directorio `chroot` con paquetes que constituyen un sistema Debian básico. A continuación la etapa **chroot** completa la creación del directorio `chroot`, rellenándolo con todos los paquetes que han sido listados en la configuración y material adicional. En esta etapa se utiliza la mayoría de las personalizaciones de contenido. La etapa **binary** es la etapa final en la que se prepara la imagen del sistema en vivo utilizando el contenido del directorio `chroot` para construir el sistema de ficheros raíz del futuro sistema en vivo, se incluye el instalador y cualquier otro material adicional de la imagen que no es parte del sistema de ficheros raíz, como puede ser el gestor de arranque (bootloader) o ficheros de documentación. Posteriormente, en la etapa opcional **source** se creará el fichero comprimido (tarball) que contiene los ficheros de código fuente de los paquetes utilizados.

En cada una de estas etapas hay una secuencia particular en la se aplican las acciones a realizar. Estas acciones son organizadas en forma de capas de tal manera que aseguran la personalización de una manera razonable. Por ejemplo,

dentro de la etapa **chroot** , las preconfiguraciones (preseeds) se aplican antes que cualquier paquete sea instalado, los paquetes son instalados antes de incluir ningún fichero localmente y los scripts gancho (hooks) serán ejecutados al final de todo, una vez que todos los materiales están ubicados en su lugar.

variantes locales e idioma cubren solamente unas pocas de las tareas que pueden realizarse.

364 7.3 Opciones para lb config en ficheros

365 Aunque la orden `lb config` crea un esqueleto de configuración en el directorio `config/`, quizás sea necesario escribir ficheros de configuración adicionales dentro de la jerarquía de subdirectorios de `config/` con el fin de alcanzar los objetivos propuestos. En el proceso de creación de la imagen estos ficheros adicionales serán copiados o en el sistema de ficheros que se utilizará en el sistema en vivo, o en el sistema de ficheros de la propia imagen binaria o quizás podrán suministrar opciones de configuración al sistema en vivo que sería incomodo pasar en la línea de parámetros del kernel. Esto dependerá de en qué parte de la jerarquía de subdirectorios de `config/` se copian estos ficheros. Se puede incluir cosas como listas de paquetes personalizadas, imágenes gráficas personalizadas o scripts gancho (hook scripts) para ejecutar o en el momento de creación de la imagen o en el momento de arranque del sistema en vivo, aumentando la ya por otra parte considerable flexibilidad de Debian Live con código creado ex profeso.

366 7.4 Tareas de personalización

367 Los siguientes capítulos se organizan por tareas de personalización que el usuario realiza típicamente: Los capítulos de **«Personalización de la instalación de paquetes»**, **«Personalización de contenidos»** y **«Personalización de las**

Personalización de la instalación de paquetes

8. Personalización de la instalación de paquetes

Quizás la tarea más básica de personalización de un sistema en vivo es la selección de paquetes que serán incluidos en la imagen. Este capítulo orienta a través de las diferentes opciones de *live-build* que, en el momento de la creación de la imagen, personalizan la instalación de paquetes. Las opciones que seleccionan la distribución base y las áreas del archivo a utilizar son las que más influyen a la hora de conocer qué paquetes estarán disponibles para su instalación en la imagen. Para asegurar una buena velocidad de descarga de paquetes, se debería elegir el repositorio más cercano. Se pueden añadir repositorios para backports, experimentales, paquetes personalizados o incluir ficheros de paquetes directamente. Se pueden definir listas de paquetes personalizadas, incluyendo metapaquetes que instalarán muchos paquetes relacionados, como por ejemplo paquetes de un entorno de escritorio o lenguaje particular. Por último existen varias opciones que dan algún control sobre cuando son instalados los paquetes por la herramienta *apt* o la herramienta *aptitude*, según sea la elegida. Estas opciones pueden ser útiles si se utiliza un proxy, se quiere desactivar la instalación de paquetes recomendados para ahorrar espacio o se necesita controlar las versiones de los paquetes a instalar mediante APT pinning, por nombrar algunas posibilidades.

8.1 Origen de los paquetes

8.1.1 Distribución, áreas de archivo y modo

La distribución seleccionada tiene gran impacto en qué paquetes están disponibles para incluir en la imagen. Se debe indicar el nombre en clave de la distribución, que por defecto

es **jessie** para la versión **jessie** de *live-build*. Se puede especificar cualquier nombre de distribución disponible en los repositorios indicando su nombre en clave. (Para más detalles ver <Términos>). La opción `--distribution` no solamente influencia la fuente de los paquetes dentro del archivo, sino que instruye a *live-build* a comportarse tal y como se necesita para construir cada una de las distribuciones. Por ejemplo, para construir la versión **inestable**, **sid**, se debe indicar:

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

Las áreas del archivo Debian son la principal división de paquetes dentro de una distribución dada. En Debian las áreas del archivo establecidas son `main`, `contrib` y `non-free`. Solamente los paquetes contenidos en `main` son parte de la distribución Debian. Ésta es el área definida por defecto en *live-build*. Se pueden indicar uno o más valores tal y como se muestra en el siguiente ejemplo:

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

Experimentalmente se da soporte a alguna distribución derivada de Debian mediante la opción `--mode`. Por defecto, esta opción toma el valor `debian` sólo si se está construyendo en un sistema Debian o en un sistema desconocido. Si se utiliza *lb config* en cualquiera de las distribuciones derivadas a las que se da soporte, por defecto se construirá una imagen de esa distribución derivada. Por ejemplo, si *lb config* se ejecuta en modo `ubuntu` se utilizará el nombre de esa distribución y las áreas de archivos específicas de esa distribución derivada en lugar de los propios de Debian

y *live-build* modificará su comportamiento para adecuarlo al modo seleccionado.

Nota: La ayuda a los usuarios de las distribuciones para las cuales se añadieron estos modos son responsabilidad de los desarrolladores de dichas distribuciones. El Live Systems Project proporciona ayuda al desarrollo de la mejor manera posible, basándose en la información recogida de dichas distribuciones derivadas a pesar de que no desarrolla ni da soporte a las mismas.

8.1.2 Réplicas de Distribución Debian

Los repositorios de Debian están replicados en una gran red alrededor del mundo, de manera que se puede seleccionar la réplica más cercana con el fin de obtener la mejor velocidad de descarga. Cada una de las opciones `--mirror-*` gobierna qué réplica de repositorio Debian se utiliza en las diferentes etapas de creación. Si se recuerda de **«Etapas de la creación»**, en la etapa **bootstrap** es cuando se crea el directorio chroot con un sistema mínimo mediante la herramienta *debootstrap*, y en la etapa **chroot** es cuando el directorio chroot es completado con los paquetes necesarios para crear el sistema de ficheros que será utilizado en el sistema en vivo. A cada una de estas etapas le corresponde su propia opción `--mirror-*`. Posteriormente, en la etapa **binary** se utilizarán las réplicas Debian indicadas en los valores de las opciones `--mirror-binary` y `--mirror-binary-security` en lugar de utilizar los indicados para las etapas anteriores.

8.1.3 Réplicas de Distribution utilizadas durante la creación

Para indicar qué réplicas deben ser utilizadas en el momento

de crear la imagen es suficiente con utilizar las opciones `--mirror-bootstrap`, `--mirror-chroot-security` y `--mirror-chroot-backports` como se muestra a continuación.

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

El valor indicado en `--mirror-chroot` es utilizado como valor por defecto para la opción `--mirror-bootstrap` si esta no es especificada.

8.1.4 Réplicas de distribución Debian utilizadas en la ejecución.

Las opciones `--mirror-binary*` gobiernan las réplicas configuradas en la imagen binaria que serán utilizadas para instalar paquetes adicionales mientras se ejecuta el sistema en vivo. Por defecto se utiliza `http.debian.net`, que es un servicio que selecciona la réplica más cercana basándose, entre otras cosas, en la familia de la IP del usuario y de la disponibilidad de la réplica. Es una elección bastante acertada siempre que no se pueda predecir que réplica será la mejor para todos los usuarios. También se puede especificar valores personalizados como se muestra en el siguiente ejemplo. Una imagen construida con esta configuración solamente sería accesible a los usuarios de una red donde “mirror” fuese alcanzable.

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


8.1.5 Repositorios adicionales

Se pueden añadir más repositorios, ampliando la lista de paquetes seleccionables más allá de aquellos disponibles para la distribución indicada, como pueden ser paquetes de backports, paquetes experimentales o personalizados. Para configurar repositorios adicionales se debe crear los ficheros `config/archives/your-repository.list.chroot` y `config/archives/your-repository.list.binary`. Al igual que en las opciones `--mirror-*`, estos ficheros gobiernan los repositorios utilizados en las etapas **chroot** y **binary** respectivamente, esto es, los repositorios que serán utilizados cuando se ejecute el sistema en vivo.

Por ejemplo, `config/archives/live.list.chroot` permite instalar paquetes de las instantáneas del repositorio Debian Live en el momento de crear la imagen.

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

Si se añade la misma línea a `config/archives/live.list.binary`, el repositorio será añadido al directorio `/etc/apt/sources.list.d/` del sistema en vivo.

Estos ficheros serán seleccionados automáticamente si existen.

Se debería también incluir en el fichero `config/archives/your-repository.key.{binary,chroot}` la clave GPG a utilizar para firmar dicho repositorio.

En caso de necesitar un APT pinning personalizado, las preferencias de APT se pueden colocar mediante ficheros `config/archives/your-repository.pref.{binary,chroot}`, y serán añadidos automáticamente al sistema en vivo en el directorio `/etc/apt/preferences.d/`.

8.2 Selección de los paquetes a instalar

Hay varias maneras de seleccionar qué paquetes serán instalados por *live-build* en la imagen que cubren una variedad de necesidades diversas. Se puede nombrar paquetes individuales para instalar en una lista de paquetes. También se puede utilizar metapaquetes en esas listas, o seleccionarlas utilizando campos de ficheros de control de paquetes. Por último, también se pueden utilizar ficheros de paquetes de prueba o experimentales obtenidos antes de que aparezcan en los repositorios oficiales simplemente depositando estos ficheros directamente en el árbol de directorios `config/`.

8.2.1 Listas de paquetes

Las listas de paquetes proporcionan una potente forma de expresar qué paquetes deberían ser instalados. La sintaxis de las listas soporta las expresiones condicionales, que facilitan la creación de listas, adaptando su utilización a diversas configuraciones. También se pueden añadir nombre de paquetes en la listas utilizando shell helpers en tiempo de construcción.

Nota: El comportamiento de *live-build* cuando se especifica un paquete que no existe es determinado por lo que se haya configurado en la utilidad APT. Para más detalles ver [«Utilizar apt o aptitude»](#).

8.2.2 Utilizar metapaquetes

La manera más sencilla de rellenar una lista de paquetes es utilizar una tarea metapaquete mantenida por una distribución. Por ejemplo:

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

Esto reemplaza el antiguo método de listas predefinidas compatible con live-build 2.x. A diferencia de las listas predefinidas, los metapaquetes de tareas no son específicos del proyecto Live Systems. Por el contrario, son mantenidas por grupos de especialistas que trabajan en la distribución y por lo tanto, reflejan el consenso de cada grupo acerca de qué paquetes sirven mejor a las necesidades de los usuarios. Además, abarcan una gama mucho más amplia de casos de uso que las listas predefinidas a las que sustituyen.

Todos los metapaquetes de tareas tienen el prefijo `task-`, por lo que una forma rápida de determinar cuáles están disponibles (aunque puede contener un puñado de entradas falsas que coincidan con el nombre, pero que no son metapaquetes) es buscar el nombre del paquete con:

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

Además de éstos, se encuentran otros metapaquetes con diversos fines. Algunos son subconjuntos de paquetes de tareas más amplias, como `gnome-core`, mientras que otros son partes especializadas individuales de un Debian Pure Blend, como los metapaquetes `education-*`. Para tener una lista de todos los metapaquetes en el archivo, instalar el paquete `debtags` y listar todos los paquetes con la etiqueta `role::metapackage` de la siguiente manera:

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

8.2.3 Listas de paquetes locales

Ya sea incluyendo metapaquetes en una lista, paquetes individuales, o una combinación de ambos, todas las listas de paquetes locales se deben almacenar en `config/package-lists/`. Ya que se puede utilizar más de una lista, esto se presta muy bien a los diseños modulares. Por ejemplo, se puede dedicar una lista a una elección particular de escritorio, la otra a una colección de paquetes relacionados que puedan ser fácilmente utilizados sobre un escritorio diferente. Esto permite experimentar con diferentes combinaciones de conjuntos de paquetes con un mínimo esfuerzo, así como compartir listas comunes entre diferentes proyectos de imágenes en vivo.

Para que sean procesadas, las listas de paquetes que se depositen en este directorio deben tener la extensión `.list` además de la extensión de la etapa `.chroot` o `.binary` para indicar a qué etapa corresponde la lista.

Nota: Si no se especifica el sufijo, la lista será usada en las dos etapas. En consecuencia, es conveniente especificar `.list.chroot` de modo que los paquetes se instalen únicamente en el sistema en vivo y no exista otra copia extra del paquete `.deb`.

8.2.4 Listas de paquetes locales para la etapa binary

Para crear una lista para la etapa «binary» crear un fichero con el sufijo `.list.binary` en `config/package-lists/`. Estos paquetes no son instalados en el sistema en vivo, pero son incluidos en `pool1/`. El uso típico de una de estas lista sería para una de las variantes de instalador normal («non-live» N.del T.). Tal y como se mencionaba anteriormente, si se desea usar la misma lista para la etapa «chroot» basta con solamente añadir el sufijo `.list`

8.2.5 Generar listas de paquetes

A veces ocurre que la mejor manera de crear una lista es generarla con un script. Cualquier línea que comience con un signo de exclamación indica un comando que se ejecutará dentro del chroot cuando la imagen se construya. Por ejemplo, se podría incluir la línea `! grep-aptavail -n -sPackage -FPriority standard |sort` en una lista de paquetes para producir una lista ordenada de los paquetes disponibles con `Priority: standard`.

De hecho, la selección de paquetes con la orden `grep-aptavail` (del paquete `dctrl-tools`) es tan útil que `live-build` proporciona un script de ayuda llamado `Packages`. Este script acepta dos argumentos: `field` y `pattern`. Por lo tanto, se puede crear una lista con los siguientes contenidos:

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

8.2.6 Utilización de condiciones dentro de las listas de paquetes

En las sentencias condicionales de las listas de paquetes pueden utilizarse cualquier variable disponible en `config/*` (excepto las que tienen el prefijo `LB_`). En general esto significa que puede utilizarse cualquier opción válida para `lb config` cambiando las letras minúsculas por mayúsculas y los guiones por barras bajas. En la práctica solamente tiene sentido utilizar aquellas variables relacionadas con la selección de paquetes, como pueden ser `DISTRIBUTION`, `ARCHITECTURES` o `ARCHIVE_AREAS`.

Por ejemplo, para instalar el paquete `ia32-libs` si se ha especificado la arquitectura `amd64` (`--architectures amd64`) se puede utilizar:

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

En la expresión condicional pueden utilizarse varios valores. Por ejemplo para instalar el paquete `memtest86+` si la arquitectura es `i386` (`--architectures i386`) o es `amd64` (`--architectures amd64`) se puede especificar:

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

En la expresión condicional también pueden utilizarse variables que pueden contener más de un valor. Por ejemplo para instalar `vrms` si se utilizan las áreas del archivo `contrib` o `non-free` mediante la opción `--archive-areas` se puede indicar:

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

No se permite el anidamiento de estructuras condicionales.

8.2.7 Eliminación paquetes durante la instalación

Se puede crear listas de paquetes en ficheros con los sufijos

`.list.chroot_live` y `.list.chroot_install` dentro del directorio `config/package-lists`. Si existe una lista «live» y una lista «install» los paquetes de la lista `.list.chroot_live` se eliminan con un script gancho después de la instalación (si el usuario utiliza el instalador). Los paquetes de la lista `.list.chroot_install` estarán presentes tanto en el sistema en vivo como en el sistema instalado. Este es un caso especial para el instalador y puede ser útil si se tiene `--debian-installer live` establecido en la configuración y se desea eliminar paquetes específicos del sistema en vivo durante la instalación.

8.2.8 Tareas de Escritorio e Idioma

Las tareas de escritorio y de idioma son casos especiales que necesitan un poco de planificación y configuración extra. Si el medio de instalación fue preparado para una clase particular de entorno de escritorio, el Instalador de Debian instalará automáticamente la tarea de entorno de escritorio correspondiente. Para ello existen las tareas internas `gnome-desktop`, `kde-desktop`, `lxde-desktop` y `xfce-desktop` pero ninguna de ellas son presentadas en el menú de `tasksel`. De igual forma, las tareas para idiomas tampoco son presentadas en el menú de `tasksel`, pero la selección del idioma, al inicio de la instalación repercute en la selección de las correspondientes tareas del idioma.

Cuando se desarrolla una imagen de escritorio, la imagen normalmente arranca directamente a un escritorio de trabajo, las opciones de escritorio y de idioma por defecto han sido elegidas en tiempo de creación, no en tiempo de ejecución como en el caso del instalador de Debian. Eso no quiere decir que una imagen en vivo no pueda ser creada para admitir múltiples escritorios o varios idiomas y ofrecer al usuario una elección, pero ese no es un comportamiento por defecto de

live-build.

Ya que no se ha previsto la instalación automática de tareas de idiomas, que incluyen cosas tales como tipos de letra específicos de cada lengua o paquetes de métodos de entrada, si se quiere incluirlos, es necesario especificarlo en la configuración. Por ejemplo, una imagen de escritorio GNOME que contenga soporte para el alemán podría incluir los siguientes metapaquetes de tareas:

```
$ 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 Versión y tipo de kernel

Dependiendo de la arquitectura, se incluyen por defecto en las imágenes uno o más tipos de kernels. Se puede elegir entre diferentes tipos utilizando la opción `--linux-flavours`. Cada tipo tiene el sufijo de la raíz predeterminada `linux-image` para formar el nombre de cada metapaquete que a su vez depende del paquete del kernel exacto que debe incluirse en la imagen.

Así, por defecto, una imagen de arquitectura `amd64` incluirá el metapaquete `linux-image-amd64` y una imagen de arquitectura `i386` incluirá los metapaquetes `linux-image-486` y `linux-image-686-pae`. En el momento de escribir esto, los paquetes dependen, respectivamente, de `linux-image-3.2.0-4-amd64`, `linux-image-3.2.0-4-486` y `linux-image-3.2.0-4-686-pae`.

Cuando hay más de una versión del paquete del kernel

disponible en los archivos configurados, se puede especificar el nombre de un paquete del kernel diferente con la opción `--linux-packages`. Por ejemplo, suponer que se está construyendo una image de arquitectura amd64 y se quiere añadir el archivo experimental a fin de realizar pruebas para que se pueda instalar el kernel `linux-image-3.7-trunk-amd64`. Se podría configurar la imagen de la siguiente manera:

```
$ 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 Kernels personalizados

Se pueden crear e incluir kernels personalizados, pero hay que tener en cuenta que *live-build* sólo soporta los kernels que se integran en el sistema de gestión de paquetes de Debian y no es compatible con kernels que no esten en paquetes `.deb`.

La manera apropiada y recomendada de implementar los propios paquetes del kernel es seguir las instrucciones del `kernel-handbook`. Recordar modificar el ABI y los sufijos de los tipos del kernel e incluir los paquetes del kernel completo en un repositorio que coincidan con los paquetes `linux` y `linux-latest`.

Si se opta por construir los paquetes del kernel sin los metapaquetes adecuados, es necesario especificar una raíz `--linux-packages` apropiada como se indica en **«Versión y tipo de kernel»**. Tal y como se explica en **«Instalar paquetes modificados o de terceros»**, es mejor si se incluyen los paquetes del kernel personalizado en un repositorio propio, aunque las alternativas discutidas en esa sección también

funcionan.

Está más allá del alcance de este documento dar consejos sobre cómo personalizar un kernel. Sin embargo, se debe por lo menos, asegurarse de que la configuración cumple los siguientes requisitos mínimos:

- Utilizar un ramdisk inicial.
- Incluir el módulo de unión de sistemas de ficheros (normalmente aufs).
- Incluir todos los módulos de sistemas de ficheros requeridos por la configuración (normalmente squashfs).

8.3 Instalar paquetes modificados o de terceros

Si bien está en contra de la filosofía de un sistema en vivo, en ocasiones es necesario crear un sistema con versiones de paquetes modificados a partir de los disponibles en el repositorio de Debian. Estos paquetes pueden modificar características existentes o dar soporte a características adicionales, idiomas y marcas, o eliminar elementos existentes en los paquetes que no son de interés. De manera similar, se pueden incluir paquetes «de terceros» para añadir funcionalidades a medida o propietarias.

En esta sección no se describe la creación o mantenimiento de paquetes personalizados. Puede ser interesante una lectura del método descrito por Joachim Breitner ‘How to fork privately’ en <http://www.joachim-breitner.de/blog/archives/282-How-to-fork-privately.html>. La guía del nuevo desarrollador de Debian en <https://www.debian.org/doc/maint-guide/> describe la creación de paquetes a medida.

Existen dos formas de instalar paquetes personalizados:

- `packages.chroot`

- Utilizando un repositorio APT personalizado

El método `packages.chroot` es el más simple para añadir paquetes personalizados. Es muy útil para personalizaciones «rápidas» pero tiene unos cuantos inconvenientes mientras que la utilización de un repositorio APT personalizado es más lento de poner en marcha.

8.3.1 Método `packages.chroot` para instalar paquetes personalizados

Para instalar paquetes personalizados solamente hay que copiar el paquete en el directorio `config/packages.-chroot/`. Los paquetes contenidos en este directorio serán automáticamente instalados en el sistema en vivo durante el proceso de creación. No es necesario especificar nada más.

Los paquetes **deben** nombrarse de la forma prescrita. La forma más simple es usar `dpkg-name`.

El método `packages.chroot` para la instalación de paquetes personalizados tiene desventajas:

- No es posible utilizar secure APT.
- Se deben depositar todos los paquetes apropiados en el directorio `config/packages.chroot/`.
- No es adecuado para almacenar configuraciones en vivo en un control de versiones.

8.3.2 Método de repositorio APT para instalar paquetes personalizados

A diferencia del método `packages.chroot`, cuando se utiliza el método de repositorio APT personalizado se debe asegurar

que se especifica dónde se deben buscar los paquetes a instalar. Para más información ver [«Selección de los paquetes a instalar»](#).

Aunque crear un repositorio APT para instalar paquetes personalizados puede parecer un esfuerzo innecesario, la infraestructura puede ser fácilmente reutilizada posteriormente para ofrecer nuevas versiones de los paquetes.

8.3.3 Paquetes personalizados y APT

live-build utiliza APT para instalar todos los paquetes en el sistema en vivo, así que hereda sus comportamientos. Un punto a resaltar es que (asumiendo una configuración de APT por defecto) dado un paquete en dos repositorios diferentes con diferentes números de versiones, APT seleccionará para instalar el paquete con número de versión superior.

Esta sería una buena razón para incrementar el número de versión en los ficheros `debian/changelog` de los paquetes personalizados y así asegurar que serán estos los paquetes instalados en lugar de los contenidos en los repositorios oficiales de Debian. Esto puede también lograrse alterando las preferencias de pinning de APT del sistema en vivo. Para más información ver [«APT pinning»](#).

8.4 Configurar APT en la creación

Se puede configurar APT mediante varias opciones que se aplicarán en el momento de crear la imagen. (La configuración que APT utilizará cuando se ejecute el sistema en vivo puede ser configurada de la manera que habitualmente se utiliza para introducir contenidos del sistema en vivo, esto es, incluyendo las configuraciones apropiadas en el directorio `config/includes.chroot/`.) Se puede encontrar una lista

completa de las opciones para configurar APT en la página de manual de `lb_config`. Son aquellas opciones que comienzan con `apt`.

8.4.1 Utilizar `apt` o `aptitude`

Se puede seleccionar qué herramienta se utilizará para instalar paquetes, *apt* o *aptitude*, en el momento de crear la imagen mediante la opción `--apt` de `lb config`. Esta selección definirá el comportamiento preferido en la instalación de paquetes, siendo la mayor diferencia la manera de tratar los paquetes no disponibles.

- `apt`: Con este método, si se especifica un paquete no existente, la instalación fallará. Es el comportamiento por defecto.
- `aptitude`: Con este método, si se especifica un paquete no existente, la instalación continuará sin error.

8.4.2 Utilización de un proxy con APT

Un problema habitual en la configuración de APT es tratar con la creación de una imagen desde detrás de un proxy. Se puede especificar dicho proxy con las opciones `--apt-ftp-proxy` o `--apt-http-proxy`. Por ejemplo:

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

8.4.3 Ajuste de APT para ahorrar espacio

En ocasiones es necesario ahorrar un poco de espacio

en el medio de instalación. Las dos opciones descritas a continuación pueden ser de interés.

Si no se desea incluir los índices de APT en la imagen creada se puede utilizar la siguiente opción:

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

Esto no modificará el comportamiento de las entradas definidas en `/etc/apt/sources.list`, sino que solo afecta a si existirán o no ficheros de índice en el directorio `/var/lib/apt`. El compromiso viene de que APT necesita estos ficheros índices para funcionar en el sistema en vivo, así que, si no existen, el usuario deberá ejecutar la orden `apt-get update` para crear estos índices antes de poder ejecutar una orden del tipo `apt-cache search` o `apt-get install`.

Si la instalación de los paquetes recomendados aumenta demasiado el tamaño de la imagen, siempre y cuando se esté preparado para hacer frente a las consecuencias que se mencionan a continuación, se puede desactivar el valor por defecto de esta opción de APT con:

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

La consecuencia más importante de desactivar los «recommends» es que `live-boot` y `live-config` recomiendan algunos paquetes que proporcionan una funcionalidad importante y que son utilizados por la mayoría de las configuraciones en vivo, como por ejemplo `user-setup` recomendado por `live-config` que se utiliza para crear el usuario en vivo. En todas menos en las circunstancias más excepcionales es necesario volver a añadir por lo menos

algunos de los «recommends» en las listas de paquetes o de lo contrario la imagen no funcionará como se espera, si es que funciona en lo más mínimo. Mirar los paquetes recomendados por cada uno de los paquetes `live-*` incluidos en la construcción y si no se está seguro de que es lo que se puede omitir, volver a agregarlo utilizando las listas de paquetes.

485 La consecuencia más general es que, si no se instalan los paquetes recomendados para un paquete dado, esto es «los paquetes que supuestamente deberían encontrarse instalados si un paquete ya lo está» (Debian Policy Manual, sección 7.2), algún paquete que supuestamente debería estar instalado será omitido. Por lo tanto, se sugiere que si se desactiva esta opción, se revise las diferencias en las listas de paquetes instalados (ver el fichero `binary.packages` generado por `lb build`) y que se vuelva a incluir en la lista cualquier paquete que deba ser instalado. Si se considera que el número de paquetes a descartar es pequeño, se recomienda que la opción se deje activada y que se utilice una prioridad pin negativa de APT en dichos paquetes y así evitar que sean instalados tal y como se explica en [«APT pinning»](#).

486 8.4.4 Pasar opciones a apt o a aptitude

487 Si no hay una opción `lb config` para modificar el comportamiento de APT en la forma que se necesita, utilizar `--apt-options` o `--aptitude-options` para pasar opciones a la herramienta APT configurada. Consultar las páginas de manual `apt` y `aptitude` para más detalles. Tener en cuenta que ambas opciones tienen valores por defecto que tendrán que mantenerse, además de las opciones que se pueden especificar. Así, por ejemplo, supongamos que se ha incluido algo con fines de prueba de `snapshot.debian.org` y se desea especificar `Acquire::Check-Valid-Until=false`

para que APT esté feliz con el fichero `Release` caducado, se haría como en el ejemplo siguiente, añadiendo la opción de nuevo después del valor por defecto `--yes`:

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

Consultar las páginas de manual para entender completamente estas opciones y cuándo utilizarlas. Esto es sólo un ejemplo y no debe ser interpretado como consejo para configurar la imagen. Esta opción no sería apropiada para, por ejemplo, una versión final de una imagen en vivo.

Para configuraciones más complicadas que implican opciones `apt.conf` puede ser necesario crear un fichero `config/-apt/apt.conf`. Ver también las otras opciones `apt-*` para tener algunos atajos convenientes para las opciones que se necesitan con frecuencia.

8.4.5 APT pinning

Como información básica, sería recomendable leer la página de manual `apt_preferences(5)`. APT pinning puede ser configurado o en tiempo de creación de la imagen, creando los ficheros `config/archives/*.pref`, `config/archives/*.pref.chroot`, y `config/apt/preferences`. o en tiempo de ejecución del sistema en vivo creando el fichero `config/includes.chroot/etc/apt/preferences`.

Supongamos que se está creando un sistema en vivo basado en **jessie** pero se necesita instalar todos los paquetes “live” que terminan instalados en la imagen binaria final desde la versión inestable «**sid**» en el momento de crear la imagen. Se deberá añadir **sid** a los orígenes (sources) de APT y fijar (pin) los paquetes live con una prioridad más alta pero todos

los otros paquetes con una prioridad más baja que la prioridad por defecto de manera que solamente los paquetes fijados sean instalados desde **sid** mientras que el resto será obtenido desde la distribución base, **jessie** . Esto se puede realizar de la siguiente forma:

494

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

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

495

Una prioridad pin negativa previene la instalación de un paquete, como puede ser el caso de que no se desee que un paquete recomendado por otro sea instalado al instalar el primero. Supongamos que se está creando una imagen LXDE añadiendo `task-lxde-desktop` en `config/package-lists/desktop.list.chroot`, pero no se desea preguntar al usuario si desea almacenar las claves wifi en el keyring. Este metapaquete depende de *lxde-core*, el cual recomienda *gksu* que a su vez recomienda *gnome-keyring*. Así que el objetivo es omitir la instalación del paquete *gnome-keyring*, que puede conseguirse añadiendo un fichero con el siguiente contenido a `config/apt/preferences`:

496

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

Personalización de contenidos

9. Personalización de contenidos

Este capítulo trata, no solamente de una mera descripción de cómo seleccionar los paquetes a incluir en el sistema en vivo, sino que además presenta cómo hacer el «ajuste fino» de la personalización de los contenidos del propio sistema. Los «includes» permiten adjuntar o reemplazar cualquier fichero en la imagen en vivo a crear, los scripts gancho (hooks) permiten ejecutar cualquier orden en las diferentes etapas de creación y en el momento del arranque y por último, la preconfiguración permite configurar paquetes cuando son instalados, suministrando las respuestas a las preguntas de debconf.

9.1 Includes

Idealmente, un sistema en vivo debería incluir solamente los ficheros proporcionados por los paquetes sin modificar. Sin embargo, algunas veces es conveniente incluir o modificar algún contenido mediante ficheros. La utilización de includes posibilita la inclusión, modificación o cambio de cualquier fichero en la imagen en vivo a crear. *live-build* utiliza dos mecanismos:

- Includes locales en chroot : Estos includes permiten incluir o reemplazar ficheros en el sistema de ficheros chroot. Para más información ver <Includes locales en Live/chroot>
- Includes locales en Binary: Estos includes permiten incluir o reemplazar ficheros en la propia imagen binaria generada. Para más información ver <Includes locales en Binary>

Para más información acerca de la diferencia entre las imágenes “Live” y “binary” ver <Términos>

9.1.1 Includes locales en Live/chroot

Los includes locales en chroot se utilizan para incluir o reemplazar ficheros en el sistema de ficheros Live/chroot de manera que puedan ser utilizados en el sistema en vivo. Una utilización típica de estos includes puede ser rellenar el directorio (/etc/skel) usado por el sistema Live para crear el directorio home del usuario. Otra utilización típica es suministrar ficheros de configuración que pueden ser incluidos o reemplazados en la imagen sin necesidad de realizar procesado alguno; Si se necesita realizar algún procesado de estos ficheros ver la sección <Scripts gancho locales en Live/chroot>

Para incluir ficheros solamente hace falta añadirlos al directorio de configuración `config/includes.chroot`. Habrá una relación directa entre este directorio y el directorio raíz / del sistema en vivo. Por ejemplo, si se desea añadir un fichero para que sea el fichero `/var/www/index.html` del sistema en vivo se puede hacer lo siguiente:

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

El directorio de configuración presentará la siguiente jerarquía:

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

Los includes locales en chroot serán instalados después de la instalación de los paquetes de manera que los includes sobrescribirán cualquier fichero que los paquetes puedan haber instalado.

9.1.2 Includes locales en Binary

Se puede incluir material como documentación, videos, etc en el sistema de ficheros del medio (USB, CDROM, etc) donde se grabará la imagen de manera que sea accesible nada más insertar el medio sin necesidad de arrancar el sistema en vivo. Para esto se utilizan los includes locales en Binary. Funciona de manera similar a los includes locales en chroot comentados anteriormente. Por ejemplo, supongamos que en el medio de instalación se desea añadir unos ficheros con videos de demostración `~/video_demo.*` sobre el funcionamiento del sistema en vivo de manera que el usuario pueda acceder a ellos a través de la página de índice HTML. Simplemente se debe copiar el material en `config/includes.binary/` de la siguiente manera:

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

Los ficheros aparecerán ahora en el directorio raíz del medio en vivo.

9.2 Scripts gancho (Hooks)

Los scripts gancho permiten ejecutar órdenes para personalizar la imagen en las etapas chroot y binary.

9.2.1 Scripts gancho locales en Live/chroot

Para ejecutar órdenes en la etapa chroot se deben crear scripts gancho (hooks) con el sufijo `.hook.chroot` que contengan dichas ordenes a ejecutar y depositarlos en el directorio `config/hooks/`. Estos scripts serán ejecutados en el entorno del chroot después de que el resto de las tareas de preparación del chroot han sido realizadas. Se debe asegurar que previamente se han instalado en el entorno chroot cualquier paquete, fichero u orden que necesiten los scripts gancho. El paquete *live-build* instala en el directorio `/usr/share/doc/live-build/examples/hooks` del sistema huésped unos cuantos scripts gancho para realizar tareas habituales de personalización del entorno chroot que pueden ser copiados o referenciados mediante enlace simbólico en la propia configuración.

9.2.2 Scripts gancho en tiempo de arranque

Para ejecutar ordenes en el arranque del sistema en vivo, se puede suministrar scripts gancho a *live-config* depositándolos en el directorio `config/includes.chroot/lib/live/config/`, tal y como se explica en la sección de “Personalización” de la página de manual de *live-config*. Es interesante examinar los scripts gancho que trae de serie *live-config* que pueden verse en `/lib/live/config/` y fijarse en la secuencia de números. Cuando se vaya a utilizar scripts propios deben ser prefijados con un número para indicar el orden de ejecución. Otra posibilidad es utilizar un paquete personalizado tal y como se describe en [«Instalar paquetes modificados o de terceros»](#).

9.2.3 Scripts gancho locales en Binary

522

523 Para ejecutar comandos en la etapa Binary se deben crear scripts gancho con el sufijo `.hook.binary` que contengan las ordenes y depositarlos en el directorio `config/hooks/`. Los scripts gancho se ejecutarán después de finalizar el resto de procesos de la etapa pero antes de crear los checksum con `binary_checksum` que es el último proceso que se ejecuta en esta etapa. Los scripts gancho no se ejecutan en el entorno del chroot, así que hay que tener cuidado de no modificar cualquier fichero fuera del árbol de creación, o se dañará el sistema de creación. En `/usr/share/doc/live-build/examples/hooks` se pueden ver varios ejemplos de scripts gancho genéricos que permiten tareas de personalización para la etapa Binary. Estos scripts pueden ser utilizados en la propia configuración copiándolos o creando enlaces simbólicos.

9.3 Preconfiguración de las preguntas de Debconf

524

525 Los ficheros del directorio `config/preseed/` con el sufijo `.cfg` seguido por la etapa (`.chroot` o `.binary`) son ficheros de preconfiguración para debconf. *live-build* instalará estos ficheros mediante `debconf-set-selections` durante la etapa correspondiente.

526 Ver `debconf(7)` en el paquete *debconf* para obtener más información acerca de debconf.

Personalización del comportamiento en tiempo de ejecución.

10. Personalización del comportamiento en tiempo de ejecución.

Toda la configuración que se hace en tiempo de ejecución es realizada por *live-config*. Éstas son algunas de las opciones más comunes de *live-config* en las que los usuarios están más interesados. Se puede encontrar una lista completa de todas las posibilidades en la página de manual de *live-config*.

10.1 Personalización del usuario por defecto del sistema en vivo

Una consideración importante es que el usuario por defecto del sistema en vivo es creado por *live-boot* en el arranque y no *live-build* durante la creación de la imagen. Ésto no sólo influye dónde se introducen los materiales relacionados con este usuario durante la creación de la imagen tal y como se explica en [«Includes locales en Live/chroot»](#) sino también a cualquier grupo y a los permisos asociados con el usuario por defecto del sistema en vivo.

Se pueden especificar grupos adicionales a los que pertenecerá el usuario por defecto del sistema en vivo mediante el uso de cualquiera de las posibilidades de configuración de *live-config*. Por ejemplo, para agregar el usuario al grupo fuse, se puede agregar el fichero siguiente a `config/includes.chroot/etc/live/config/user-setup.conf`:

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

o utilizar `live-config.user-default-groups=audio,cdrom,dip,floppy,video` como parámetro de arranque.

Además, es posible cambiar el usuario por defecto "user" y la contraseña por defecto "live". Si se desea cambiarlos por cualquier motivo, se puede conseguir de forma sencilla tal y como se explica a continuación:

Cambiar el nombre del usuario por defecto es tan sencillo como especificarlo en la configuración:

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

Una posible forma de cambiar la contraseña por defecto es usando un script gancho (hook) tal y como se describe en [«Scripts gancho en tiempo de arranque»](#). Para conseguirlo se puede usar el script gancho «passwd» de `/usr/share/doc/live-config/examples/hooks`, ponerle un prefijo adecuado (p.ej. 2000-passwd) y añadirlo a `config/includes.chroot/lib/live/config/`

10.2 Personalización de las variantes locales e idioma

Cuando el sistema en vivo arranca, el idioma está implicado en dos pasos:

- Generar las variantes locales
- Establecer la distribución del teclado

La variante local predeterminada en la creación de un sistema en vivo es `locales=en_US.UTF-8`. Para definir la variante local que se debe generar, se puede utilizar el parámetro `locales` en la opción `--bootappend-live` de `lb config`, p.ej.

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

Se pueden especificar diversas variantes locales separándolas con comas.

Este parámetro se puede utilizar en la línea de comandos del kernel, al igual que los parámetros de configuración del teclado indicados a continuación. Es posible configurar una variante local con idioma_país (en cuyo caso se utiliza el tipo de codificación por omisión) o también con la expresión completa idioma_país.codificación. La lista de todas las variantes locales está en /usr/share/i18n/SUPPORTED.

live-config se encarga de la configuración del teclado de la consola y del entorno gráfico X utilizando el paquete console-setup. Para configurarlos se puede utilizar los parámetros de arranque keyboard-layouts, keyboard-variants, keyboard-options y keyboard-model a través de la opción --bootappend-live. Se puede encontrar una lista de opciones válidas para estos parámetros en /usr/share/X11/xkb/rules/base.lst. Para hallar la distribución del teclado y la variante que corresponde a un idioma se puede buscar el nombre en inglés de la nación donde se habla el idioma, por ejemplo:

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

Cada variante muestra una descripción de la disposición que aplica. 549

Normalmente, sólo es necesario configurar la disposición del teclado. Por ejemplo, para obtener los ficheros de la variante local de la disposición del teclado alemán y suizo-alemán en X utilizar: 550

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

Sin embargo, para casos de uso muy específicos, se puede incluir otros parámetros. Por ejemplo, para configurar un sistema Francés con una disposición French-Dvorak (también llamado Bepo) en un teclado USB TypeMatrix EZ-Reach 2030, utilizar: 551

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

Para cada una de las variables de configuración del teclado keyboard-* se puede especificar varios valores separados por comas. A excepción de keyboard-model, que sólo acepta un valor. En la página de manual keyboard(5) se explican los detalles y algunos ejemplos de cómo utilizar las variables XKBMODEL, XKBLAYOUT, XKBVARIANT y XKBOPTIONS. Si se especifican diferentes valores en keyboard-variants estos se corresponderán uno a uno con los valores keyboard-layouts (ver setxkbmap(1) opción -variant). Se admiten valores vacíos; por ejemplo para definir dos distribuciones de teclado, la que se usa por omisión US QWERTY y otra US Dvorak, utilizar: 552

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

10.3 Persistencia

Un paradigma de un cd en vivo («live cd» N. del T.) es ser un sistema pre-instalado que funciona desde medios de almacenamiento de sólo lectura, como un CD-ROM, donde los cambios y las modificaciones no se guardan tras reiniciar el sistema en que se ejecuta.

Un sistema en vivo es una generalización de este paradigma pero que es compatible con otros medios de almacenamiento, no sólo en CDs. Aún así, en su comportamiento predeterminado, se debe considerar un sistema de sólo lectura y todos los cambios en tiempo de ejecución del sistema se pierden al apagar el equipo.

La «persistencia» es un nombre común que se da a los diferentes tipos de soluciones para guardar algunos o todos los cambios realizados durante la ejecución tras reiniciar el sistema. Para entender cómo funciona es útil saber que incluso si el sistema se inicia y se ejecuta desde los medios de almacenamiento de sólo lectura, las modificaciones de los ficheros y directorios se escriben en medios de escritura, por lo general en la memoria ram (tmpfs) y los datos guardados en la ram se pierden al reiniciar.

Los datos almacenados en esta memoria ram se pueden guardar en un soporte grabable, como un medio de almacenamiento local, un recurso compartido en red o incluso en una sesión de un CD/DVD regrabable en multisesión. Todos estos medios son compatibles de diferentes maneras y

todos, menos el último, requieren un parámetro de arranque especial que se especificará en el momento del arranque: persistence.

Si se usa el parámetro de arranque persistence (y no se usa la opción nopersistence), se busca en los medios de almacenamiento locales (p.ej. discos duros, llaves USB) volúmenes con persistencia durante el arranque. Es posible restringir qué tipos de volúmenes persistentes se pueden usar especificando ciertos parámetros de arranque descritos en la página del manual de *live-boot*(7). Un volumen persistente es cualquiera de los siguientes:

- una partición, identificada por su nombre GPT.
- Un sistema de ficheros, identificado por su etiqueta de sistema de ficheros.
- una fichero imagen situado en la raíz de cualquier sistema de ficheros que pueda ser leído (incluso una partición NTFS de otro sistema operativo), identificado por su nombre de fichero.

La etiqueta del volumen para las overlays debe ser persistence pero será ignorado a menos que contenga en su raíz un fichero llamado persistence.conf que se utiliza para personalizar la persistencia del volumen, esto es, especificar los directorios que se desea guardar en un volumen de persistencia después de reiniciar. Ver «El fichero persistence.conf» para más detalles.

He aquí algunos ejemplos de cómo preparar un volumen para ser usado para la persistencia. Puede ser, por ejemplo, una partición en un disco duro o en una llave usb creada con, p.ej.

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

Ver [«Usar el espacio libre en el dispositivo USB»](#).

Si ya existe una partición en el dispositivo, sólo se tiene que cambiar la etiqueta con uno de los siguientes:

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

Un ejemplo de cómo crear un fichero imagen basado en ext4 para ser usado para la persistencia:

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

Después de crear el fichero imagen, a modo de ejemplo, para hacer `/usr` persistente pero únicamente guardando los cambios que se realizan en ese directorio en lugar de todos los contenidos de `/usr`, se puede utilizar la opción “union”. Si el fichero imagen se encuentra en el directorio `home`, copiarlo a la raíz del sistema de ficheros del disco duro y montarlo en `/mnt` como se explica a continuación:

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

Después, crear el fichero `persistence.conf` añadiendo contenido y desmontar el fichero imagen.

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

Ahora, reiniciar y arrancar el medio en vivo con el parámetro de arranque “persistence”.

10.3.1 El fichero `persistence.conf`

Un volumen con la etiqueta `persistence` debe ser configurado a través de un fichero `persistence.conf` para crear directorios arbitrarios persistentes. Ese fichero, situado en el sistema de ficheros raíz del volumen, controla que directorios hace persistentes y también de que manera.

En la página de manual de `persistence.conf`(5) se explica en detalle cómo se configura el montaje de las overlays, pero un sencillo ejemplo es suficiente para la mayoría de los casos. Supongamos que queremos crear nuestro directorio `home` y `APT` cache persistentes en un sistema de ficheros `ext4` en la partición `/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
```

Entonces reiniciamos. Durante el primer arranque los contenidos de `/home` y `/var/cache/apt` se copiarán en el volumen persistente y a partir de ese momento todos los cambios en esos directorios se guardarán allí. Tener en cuenta que las rutas listadas en el fichero `persistence.conf` no pueden contener espacios en blanco ni los componentes especiales `.` y `...`. Además, ni `/lib`, `/lib/live` (o ninguno de sus sub-directorios) ni `/` pueden hacerse persistentes montándolos de forma personalizada. Una posible alternativa a esta limitación es añadir `/ union` al fichero `persistence.conf` para conseguir una persistencia completa.

10.3.2 Utilizar varios medios persistentes

Existen diferentes métodos para utilizar múltiples volúmenes de persistencia para diferentes casos de uso. Por ejemplo, utilizar varios volúmenes al mismo tiempo o seleccionar sólo uno, entre varios, para fines muy específicos.

Se puede usar diferentes volúmenes de overlays al mismo tiempo (con sus propios ficheros `persistence.conf`) pero si varios volúmenes hacen que un mismo directorio sea persistente, sólo uno de ellos será usado. Si dos unidades montadas están “anidadas” (es decir, una es un sub-directorio de la otra) el directorio superior será montado antes que el inferior de este modo no quedará uno escondido por el otro. La personalización de los montajes anidados es problemática si están listados en el mismo fichero `persistence.conf`. Consultar la página de manual de `persistence.conf`(5) para ver como manejar ese caso si realmente es necesario. (aclaración: normalmente no lo es).

Un posible caso de uso: Si se desea guardar los datos del usuario, es decir `/home` y los datos del superusuario, es decir `/root` en particiones diferentes, crear dos particiones con la etiqueta `persistence` y añadir un fichero `persistence.conf` en cada una de este modo, `# echo "/home" > persistence.conf` para la primera partición que guardará los ficheros del usuario y `# echo "/root" > persistence.conf` para la segunda partición que almacenará los ficheros del superusuario. Finalmente, utilizar el parámetro de arranque `persistence`.

Si un usuario necesita un almacenamiento persistente múltiple del mismo tipo para diferentes lugares o pruebas, tales como `private` y `work`, el parámetro de arranque `persistence-label` usado junto con el parámetro de arranque `persistence` permitirá medios de almacenamiento persistentes múltiples pero únicos. Un ejemplo sería, si un usuario desea utilizar

una partición persistente etiquetada `private` para datos de uso personal como los marcadores de un navegador o similares utilizaría los parámetros de arranque: `persistence persistence-label=private`. Y para almacenar datos relacionados con el trabajo, como documentos, proyectos de investigación o de otro tipo, utilizaría los parámetros de arranque: `persistence persistence-label=work`.

Es importante recordar que cada uno de estos volúmenes, `private` y `work`, necesita también un fichero `persistence.conf` en su raíz. La página de manual de *live-boot* contiene más información acerca de cómo utilizar estas etiquetas con los antiguos nombres que se utilizaban en anteriores versiones.

10.4 Utilizar persistencia con cifrado

Utilizar la persistencia significa que algunos datos sensibles pueden estar expuestos a riesgo. Especialmente si los datos persistentes se almacenan en un dispositivo portable, como una memoria USB o un disco duro externo. Es entonces cuando el cifrado cobra sentido. Incluso aunque todo el procedimiento puede parecer complicado debido a la cantidad de pasos que hay que hacer, es muy fácil manejar particiones cifradas con *live-boot*. Para utilizar **luks**, que es el tipo de cifrado soportado, se necesita instalar *cryptsetup* tanto en la máquina que va a crear la partición cifrada como en el sistema en vivo con que se va a utilizar la partición persistente cifrada.

Para instalar *cryptsetup* en nuestra máquina:

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

Para instalar *cryptsetup* en nuestro sistema en vivo, lo añadimos a una package-lists:

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

Una vez se tiene el sistema en vivo con *cryptsetup*, básicamente, sólo se necesita crear una nueva partición, cifrarla y arrancar con los parámetros persistence y persistence-encryption=luks. Podríamos habernos anticipado a este paso y haber añadido esos parámetros de arranque siguiendo el procedimiento habitual:

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

Vamos a entrar en detalles para quien que no esté familiarizado con el cifrado. En el siguiente ejemplo vamos a utilizar una partición en un dispositivo usb que corresponde a `/dev/sdc2`. Tener en cuenta que es necesario determinar qué partición es la que se va a utilizar en cada caso específico.

El primer paso es conectar la memoria usb y determinar de qué dispositivo se trata. El método recomendado para los dispositivos en *live-manual* es utilizando `ls -l /dev/disk/by-id`. Después de eso, crear una nueva partición y, a continuación, cifrarla con una frase de contraseña de la siguiente manera:

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

A continuación, abrir la partición luks en el mapeador de dispositivos virtuales. Se puede utilizar cualquier nombre que se desee. Aquí utilizamos **live** como ejemplo:

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

El siguiente paso es llenar el dispositivo con ceros antes de crear el sistema de ficheros:

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

Ahora, estamos listos para crear el sistema de ficheros. Nótese que estamos añadiendo la etiqueta persistence para que el dispositivo se monte como almacén de persistencia durante el arranque.

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

Para continuar con nuestra configuración, necesitamos montar el dispositivo, por ejemplo, en `/mnt`.

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

Y crear el fichero `persistence.conf` en la raíz de la partición. Esto es, como se ha explicado antes, estrictamente necesario. Ver [El fichero persistence.conf](#).

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

Entonces, desmontar el punto de montaje:

```
# umount /mnt
```

Y opcionalmente, aunque puede ser una buena manera de salvaguardar los datos que acabamos de agregar a la partición, podemos cerrar el dispositivo:

```
# cryptsetup luksClose live
```

Vamos a resumir el proceso. Hasta ahora, hemos creado un sistema vivo capaz de utilizar el cifrado, que se puede copiar en una memoria usb como se explica en [«Copiar una imagen ISO híbrida en un dispositivo USB»](#). También hemos creado una partición cifrada, que se puede crear en la misma memoria usb para llevarla a todas partes y hemos configurado la partición cifrada para ser utilizada como almacén de persistencia. Así que ahora, sólo tenemos que arrancar el sistema en vivo. En el momento del arranque, *live-boot* nos preguntará la frase de contraseña y montará la partición cifrada para ser utilizada para la persistencia.

Personalización de la imagen binaria

11. Personalización de la imagen binaria

11.1 Gestores de arranque

live-build utiliza *syslinux* y algunos de sus derivados (en función del tipo de imagen) como gestores de arranque por defecto. Se pueden personalizar fácilmente para satisfacer todas las necesidades.

Para utilizar un tema completo, copiar `/usr/share/live/-build/bootloaders` en `config/bootloaders` y editar los ficheros allí. Si no se desea modificar todas las configuraciones de los gestores de arranque disponibles, es suficiente con sólo proporcionar una copia local personalizada de uno, por ejemplo, copiar la configuración de **isolinux** en `config/bootloaders/isolinux` es suficiente, dependiendo del caso de uso.

Cuando se modifica uno de los temas predeterminados, si se quiere utilizar una imagen de fondo personalizada que se mostrará junto con el menú de arranque, añadir una imagen `splash.png` de 640x480 píxeles. Y entonces, borrar el fichero `splash.svg`.

Hay muchas posibilidades a la hora de hacer cambios. Por ejemplo, los derivados de *syslinux* están configurados por defecto con un tiempo de espera de 0 (cero) lo que significa que harán una pausa indefinida en su pantalla de inicio hasta que se pulse una tecla.

Para modificar el tiempo de espera de arranque de una imagen *iso-hybrid* se puede editar el fichero **isolinux.cfg** especificando el tiempo en unidades de segundo 1/10. Un fichero **isolinux.cfg** modificado para arrancar después de cinco segundos sería así:

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

11.2 Metadatos ISO

Al crear una imagen binaria ISO9660 se pueden utilizar las siguientes opciones para añadir varios metadatos textuales a la imagen. Esto puede ayudar a identificar fácilmente la versión o la configuración de una imagen sin arrancarla.

- **LB_ISO_APPLICATION/--iso-application NAME:** Esto debería especificar la aplicación que estará en la imagen. La longitud máxima para este campo es de 128 caracteres.
- **LB_ISO_PREPARER/--iso-preparer NAME:** Esto debería identificar quién prepara la imagen, por lo general con algunos detalles de contacto. El valor predeterminado para esta opción es la versión de *live-build* que se está utilizando, lo que puede ayudar con la posterior depuración de errores. La longitud máxima para este campo es de 128 caracteres.
- **LB_ISO_PUBLISHER/--iso-publisher NAME:** Esto debería identificar quién publica la imagen, por lo general con algunos detalles de contacto. La longitud máxima para este campo es de 128 caracteres.
- **LB_ISO_VOLUME/--iso-volume NAME:** Esto debería especificar el volumen de identificación de la imagen. Esto se utiliza como etiqueta visible para el usuario en algunas plataformas como Windows y Apple Mac OS. La longitud máxima para este campo es de 32 caracteres.

Personalización del Instalador de Debian

12. Personalización del Instalador de Debian

Las imágenes de los sistemas en vivo pueden integrarse con el Instalador de Debian. Hay varios tipos de instalación que se diferencian en qué se incluye en la imagen y en cómo opera el instalador.

En esta sección se debe estar atento a la utilización de las mayúsculas. Cuando se utiliza «Instalador de Debian», con mayúsculas, se hace referencia explícita al instalador oficial del sistema Debian, y a nada más ni a ningún otro instalador. A menudo se abrevia con «d-i».

12.1 Tipos de imágenes según el instalador

Principalmente existen tres tipos de imágenes según el instalador:

Imágenes con Instalador Debian «normal» : Esta imagen en vivo se puede considerar como la imagen habitual. Dispone de un kernel y un initrd diferenciados que, al ser seleccionados desde el gestor de arranque, ejecutan un Instalador de Debian estándar, de la misma manera que lo harían si se arrancase desde una imagen de CD descargada desde el sitio oficial de Debian. Las imágenes que contienen un sistema en vivo con otro instalador independiente se suelen llamar «imágenes combinadas».

En estas imágenes, el sistema operativo Debian se instala mediante la herramienta *debootstrap* que descarga paquetes .deb desde medios locales o por red. El resultado final es un sistema Debian por defecto instalado en el disco duro.

El conjunto de este proceso puede ser preconfigurado (preseeded) y personalizado de muchas maneras; Para más

información, ver las páginas relevantes en el manual del Instalador de Debian. Una vez que se ha generado el fichero de preconfiguración adecuado a las necesidades, *live-build* puede encargarse de depositarlo en la imagen y activarlo de forma automática.

Imágenes con Instalador Debian «Live» : Estas imágenes en vivo también disponen de un kernel y un initrd diferenciados que, al ser seleccionados desde el gestor de arranque, ejecutan un Instalador de Debian.

El procedimiento de instalación es idéntico al realizado por las imágenes «Regulares» pero, en lugar de utilizar *debootstrap* para obtener e instalar paquetes .deb, lo que hace es copiar al disco duro la imagen del sistema de ficheros que se había preparado para lanzar el sistema en vivo. Esto se logra mediante un .udeb especial llamado *live-installer*.

Una vez finalizada esta etapa, el Instalador de Debian continua normalmente, instalando y configurando los siguientes elementos como pueden ser gestor de arranque, creación de usuarios locales, etc.

Nota: Para poder incluir los dos tipos de instalador, «normal» y «live», en el mismo medio, se debe deshabilitar el *live-installer*. Esto se hace utilizando la variable de preconfiguración (preseed) *live-installer/enable=false*.

Instalador Debian «del escritorio» : Una vez el sistema en vivo está ejecutandose, se puede lanzar el Instalador de Debian haciendo clic en el icono correspondiente, sin importar el tipo de Instalador Debian utilizado en el arranque. Esta manera de instalar Debian es más sencilla para el usuario y aconsejable en algunas situaciones. Para poder realizar esta acción se debe instalar el paquete *debian-installer-launcher*.

Por defecto, *live-build* no incluye las imágenes que utilizan

el Instalador de Debian. Esto debe ser habilitado de forma específica en `lb config`. También hay que hacer notar que, para que la instalación desde «el escritorio» funcione, el kernel del sistema en vivo debe ser el mismo que el kernel que utiliza `d-i` en la arquitectura especificada. Por ejemplo:

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

12.2 Personalizando el Instalador de Debian mediante preconfiguración

Tal y como se describe en el apéndice B del manual del Instalador de Debian que puede consultarse en <https://www.debian.org/releases/stable/i386/apb.html>, «La preconfiguración permite asociar respuestas a preguntas que aparecen en el proceso de instalación, sin tener que responderlas manualmente en el momento se se ejecuta dicho proceso. Esto hace posible automatizar totalmente la mayoría de las instalaciones e incluso ofrece alguna característica que no está disponible durante una instalación normal.» Con *live-build* se puede llevar a cabo esta personalización depositando un fichero llamado `preseed.cfg` en el directorio de configuración `config/includes.installer/`. Por ejemplo, para preconfigurar la variante local a `en_US` se puede hacer:

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

12.3 Personalizar el contenido del Instalador de Debian

Es posible que, con propósitos experimentales o para depuración de errores, se desee incluir paquetes `udeb` creados localmente para el `d-i`. Estos paquetes `udeb` son componentes del Instalador de Debian que definen su comportamiento. Para incluirlos en la imagen, basta con depositarlos en el directorio de configuración `config/packages.binary/`. También pueden incluirse o reemplazarse ficheros y directorios en el `initrd` del instalador de una manera similar a la que se describe en [«Includes locales en Live/chroot»](#), depositando el material en el directorio `config/includes.installer/`.

651

Proyecto

Contribuir al proyecto

13. Contribuir al proyecto

Cuando se envía una contribución se debe identificar claramente al titular de los derechos de autor e incluir la declaración de las licencias aplicables. Se hace notar que para ser aceptada, una contribución debe ser publicada bajo la misma licencia que el resto del documento, es decir, GPL versión 3 o posterior.

Las contribuciones al proyecto, tales como traducciones y parches, son muy bienvenidas. Cualquiera puede hacer una entrega en los repositorios, sin embargo, a la hora de hacer grandes cambios, es conveniente enviarlos a la lista de correo para debatirlos primero. Ver la sección [«Contacto»](#) para más información.

El Live Systems Project utiliza Git como sistema de control de versiones y gestión de código fuente. Como se explica en [«Repositorios Git»](#) hay dos ramas principales de desarrollo: **debian** y **debian-next**. Todo el mundo puede hacer entregas a las ramas *debian-next* de los repositorios *live-boot*, *live-build*, *live-config*, *live-images*, *live-manual* y *live-tools*.

Sin embargo, existen ciertas restricciones. El servidor rechazará:

- Entregas que no sean fast-forward
- Entregas que hagan fusiones.
- Añadir o borrar etiquetas o ramas.

A pesar de que todas las entregas pueden ser revisadas, pedimos usar el sentido común y hacer buenos commits con mensajes de commit adecuados.

- Hay que escribir mensajes de entrega que consistan en

una frase en inglés con significado completo, comenzando con una letra mayúscula y acabando con un punto final. Es habitual comenzar estas frases con la forma 'Fixing/Adding/-Removing/Correcting/Translating/...'.
 663

- Escribir buenos mensajes de entrega. La primera frase debe ser un resumen exacto de los contenidos del commit, que se incluirá en la lista de cambios. Si se necesita hacer algunas aclaraciones, escribirlas debajo dejando una línea en blanco después de la primera y luego otra línea en blanco después de cada párrafo. Las líneas de los párrafos no deben superar los 80 caracteres de longitud.
- Hacer entregas de forma atómica, es decir, no mezclar cosas no relacionadas en el mismo commit. Hacer un commit diferente para cada cambio que se realice.

13.1 Realizar cambios

Para hacer una entrega a los repositorios, se debe seguir el siguiente procedimiento. Aquí se utiliza *live-manual* como ejemplo, por eso hay que sustituirlo por el nombre del repositorio con el que se desea trabajar. Para obtener información detallada sobre cómo editar *live-manual* ver [«Contribuir a este documento»](#).

- Obtener la clave pública de entrega:

```
$ 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*
```

- Añadir la siguiente sección en el fichero de configuración de

openssh-client:

679

670

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

```
$ git push
```

- Obtener un clon del manual mediante git utilizando ssh:

672

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

- Acordarse de configurar el autor y el email en Git:

674

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

675 **Importante:** Recordar que hay que enviar los cambios a la rama **debian-next** .

- Efectuar los cambios. En este ejemplo, primero se escribiría una nueva sección sobre cómo aplicar parches y luego se añadirían los ficheros y se escribiría el mensaje de la siguiente manera:

677

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

- Para finalizar se realizará la entrega al servidor:

678

Cómo informar acerca de errores.

14. Informes de errores.

Los sistemas en vivo están lejos de ser perfectos, pero queremos que sean lo más perfectos posible - con su ayuda. No dudar en informar de un error. Es mejor llenar un informe dos veces que no hacerlo nunca. Sin embargo, este capítulo incluye recomendaciones sobre cómo presentar buenos informes de errores.

Para los impacientes:

- Primero, siempre se debe comprobar el estado actualizado de la imagen en busca de problemas conocidos en la página web <http://live-systems.org/>.
- Antes de presentar un informe de errores, se debe intentar reproducir el error con las **versiones más recientes** de *live-build*, *live-boot*, *live-config* y *live-tools* de la rama de que se está utilizando (como la última versión 4.x de *live-build* si se utiliza *live-build* 4).
- Se debe intentar proporcionar **una información tan específica como sea posible** acerca del error. Esto incluye (al menos) la versión de *live-build*, *live-boot*, *live-config* y *live-tools* utilizada y la distribución del sistema en vivo que se está construyendo.

14.1 Problemas conocidos

Debido a que Debian **testing** y Debian **unstable** están cambiando continuamente, no siempre es posible crear un sistema con éxito cuando se especifica cualquiera de estas dos versiones como distribución objetivo.

Si esto causa mucha dificultad, no se debe crear un sistema

basado en **testing** o **unstable**, sino que debe utilizarse **stable**. *live-build* siempre crea, por defecto, la versión **stable**.

Los problemas detectados se especifican en la sección 'status' de la página web <http://live-systems.org/>.

Está fuera del alcance de este manual enseñar cómo identificar y solucionar correctamente problemas de los paquetes de las distribuciones en desarrollo, sin embargo, hay dos cosas que siempre se puede intentar: Si se detecta un error de creación cuando la distribución de destino es **testing**, se debe intentar con **unstable**. Si **unstable** no funciona bien, se debe volver a **testing** haciendo un pin con la nueva versión del paquete de **unstable** (véase [APT pinning](#) para más detalles).

14.2 Reconstruir desde cero

Para asegurarse de que un error en particular no es causado por crear el sistema basándose en los datos de un sistema anterior, se debe reconstruir el sistema en vivo entero, desde el principio y comprobar si el error es reproducible.

14.3 Utilizar paquetes actualizados

Utilizar paquetes obsoletos puede causar problemas importantes al tratar de reproducir (y en última instancia, solucionar) el problema. Hay que asegurarse de que el sistema de construcción está actualizado y cualquier paquete que se incluya en la imagen esté también al día.

14.4 Recopilar información

Se debe proporcionar información suficiente con el informe. Como mínimo, la versión exacta de *live-build* donde se encuentra el error y los pasos para reproducirlo. Se debe utilizar

el sentido común e incluir cualquier información pertinente si se cree que podría ayudar a resolver el problema.

Para sacar el máximo provecho de un informe de errores, se requerirá al menos la siguiente información:

- Arquitectura del sistema anfitrión
- Distribución del sistema anfitrión.
- La versión de *live-build* del sistema anfitrión.
- Versión de Python en el sistema anfitrión.
- Versión de *debootstrap* y/o *cdebootstrap* en el sistema anfitrión.
- Arquitectura del sistema en vivo.
- Distribución del sistema en vivo.
- Versión de *live-boot* en el sistema en vivo.
- Versión de *live-config* en el sistema en vivo.
- Versión de *live-tools* en el sistema en vivo.

Se puede generar un log del proceso de creación mediante el comando `tee`. Se recomienda hacer esto de forma automática con un script `auto/build` (ver [«Gestionar una configuración»](#) para más detalles).

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

En el momento del arranque, *live-boot* y *live-config* guardan sus logs en `/var/log/live/`. Comprobar si hay algún mensaje de error en estos ficheros.

Además, para descartar otros errores, siempre es una buena idea comprimir en un `.tar` el directorio `config/` y subirlo a algún lugar, para que el equipo de Debian Live pueda reproducir

el error (**No** se debe enviar como documento adjunto a la lista de correo). Si esto es difícil (por ejemplo, debido a su tamaño) se puede utilizar la salida del comando `lb config -dump` que produce un resumen del árbol de configuración (es decir, listas de archivos de los subdirectorios de `config /` pero no los incluye).

Hay que recordar que los informes a enviar se deben hacer en inglés, por lo que para generar los logs en este idioma se debe utilizar la variante local English, p.ej. ejecutar los comandos de *live-build* o cualquier otro precedidos de `LC_ALL=C` o `LC_ALL=en_US`.

14.5 Aislar el fallo si es posible

Si es posible, aislar el caso del fallo al menor cambio posible que lo produzca. No siempre es fácil hacer esto, así que si no se consigue para el informe, no hay que preocuparse. Sin embargo, si se planea el ciclo de desarrollo bien, con conjuntos de cambios lo bastante pequeños por iteración, puede ser posible aislar el problema mediante la construcción de una simple «base» de configuración que se ajuste a la configuración actual deseada, más el conjunto del cambio que falla añadido. Si resulta difícil determinar que cambios produjeron el error, puede ser que se haya incluido demasiado en cada conjunto de cambios y se deba desarrollar en incrementos más pequeños.

14.6 Utilizar el paquete correcto sobre el que informar del error

Si no se sabe qué componente es responsable del error o si el error es un error general relativo a los sistemas en vivo, se puede rellenar un informe de errores contra el pseudo-paquete `debian-live`.

718 Sin embargo, se agradece si se intenta limitar la búsqueda a donde aparece el error.

intentar reproducir siempre preinstalando desde una réplica oficial.

719 **14.6.1 En la preinstalación (bootstrap) en tiempo de creación.**

720 *live-build* crea primero un sistema Debian básico con *debootstrap* o *cdebootstrap*. Puede fallar dependiendo de la herramienta usada en la preinstalación y de la distribución Debian empleada. Si un error aparece en este momento, se debe comprobar si está relacionado con un paquete específico de Debian (es lo más probable), o si está relacionado con la herramienta de preinstalación en sí.

721 En ambos casos, esto no es un error en el sistema en vivo, sino de Debian en sí mismo, por lo cual nosotros probablemente no podamos solucionarlo directamente. Informar del error sobre la herramienta de preinstalación o el paquete que falla.

722 **14.6.2 Mientras se instalan paquetes en tiempo de creación.**

723 *live-build* instala paquetes adicionales del archivo de Debian que pueden fallar en función de la distribución Debian utilizada y del estado diario del archivo Debian. Se debe comprobar si el error es reproducible en un sistema Debian normal, si el fallo aparece en esta etapa.

724 Si este es el caso, esto no es un error en el sistema en vivo, sino de Debian - se debe informar sobre el paquete que falla. Se puede obtener más información ejecutando *debootstrap* de forma separada del sistema de creación en vivo o ejecutando `lb bootstrap --debug`.

725 Además, si se está utilizando una réplica local y/o cualquier tipo de proxy y se experimenta un problema, se debe

14.6.3 En tiempo de arranque

726

Si la imagen no arranca, se debería informar a la lista de correo, junto con la información solicitada en **«Recopilar información»**. No hay que olvidar mencionar, cómo y cuándo la imagen falla, si es utilizando virtualización o hardware real. Si se está utilizando una tecnología de virtualización de cualquier tipo, se debe probar la imagen en hardware real antes de informar de un error. Proporcionar una captura de pantalla del error también es muy útil.

727

14.6.4 En tiempo de ejecución

728

Si un paquete se ha instalado correctamente, pero falla cuando se ejecuta el sistema en vivo, esto es probablemente un error en el sistema en vivo. Sin embargo:

729

14.7 Hacer la investigación

730

Antes de presentar el informe de errores, buscar en la web el mensaje de error en particular o el síntoma que se está percibiendo. Como es muy poco probable que sea la única persona que tiene ese problema en concreto, siempre existe la posibilidad de que se haya discutido en otras partes y exista una posible solución, parche o se haya propuesto una alternativa.

731

Se debe prestar especial atención a la lista de correo del sistema en vivo, así como a su página web, ya que seguramente tienen la información más actualizada. Si esa información existe, se debe incluir una referencia a ella en el

732

informe de errores.

733 Además, se debe comprobar las listas de errores actuales de *live-build*, *live-boot*, *live-config* y *live-tools* y verificar si se ha informado ya de algo similar.

734 14.8 Dónde informar de los fallos

735 El Live Systems Project realiza un seguimiento de todos los errores en el sistema de seguimiento de errores de Debian (BTS). Para obtener información sobre cómo utilizar el sistema, consultar <<https://bugs.debian.org/>>. También se puede enviar los errores mediante el comando `reportbug` del paquete con el mismo nombre.

736 En general, se debe informar sobre los errores en tiempo de creación contra el paquete *live-build*. De los fallos en tiempo de arranque contra el paquete *live-boot*, y de los errores en tiempo de ejecución contra el paquete *live-config*. Si no se está seguro de qué paquete es el adecuado o se necesita más ayuda antes de presentar un informe de errores, lo mejor es enviar un informe contra el pseudo-paquete `debian-live`. Nosotros nos encargaremos de reasignarlo donde sea apropiado.

737 Hay que tener en cuenta que los errores que se encuentran en las distribuciones derivadas de Debian (como Ubuntu y otras) **no** deben enviarse al BTS de Debian a menos que también se puedan reproducir en un sistema Debian usando paquetes oficiales de Debian.

Estilo de código

15. Estilo de código

En este capítulo se documenta el estilo de código utilizado en los sistemas en vivo.

15.1 Compatibilidad

- No utilizar sintaxis o semántica que sea única para el intérprete de comandos Bash. Por ejemplo, el uso de arrays.
- Utilizar únicamente el subconjunto POSIX - por ejemplo, usar `$(foo)` en lugar de `'foo'`.
- Se puede comprobar las secuencias de comandos con `'sh -n'` y `'checkbashisms'`.
- Asegurarse de que el código funcione con `'set -e'`.

15.2 Sangrado

- Utilizar siempre los tabuladores en lugar de espacios.

15.3 Ajuste de líneas

- En general, las líneas contienen 80 caracteres como máximo.
- Utilizar los saltos de línea al «estilo Linux»:

Mal:

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

Bien:

```
if foo
then
    bar
fi
```

- Lo mismo vale para las funciones:

Mal:

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

Bien:

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

15.4 Variables

- Las variables deben escribirse siempre en letras mayúsculas.
- Las variables que se utiliza en *live-build* siempre comienzan con el prefijo `LB_`.
- Las variables temporales internas de *live-build* deben comenzar con el prefijo `<=underscore>LB_`.
- Las variables locales comienzan con el prefijo *live-build* `<=underscore><=underscore>LB_`.

- Las variables en relación a un parámetro de arranque en *live-config* comienzan con LIVE_.

- Todas las demás variables de *live-config* comienzan con el prefijo _

- Utilizar llaves para las variables, por ejemplo, escribir `${F00}` en lugar de `$F00`.

- Utilizar comillas dobles en las variables para evitar dejar espacios en blanco: Escribir `"${F00}"` en lugar de `${F00}`.

- Por motivos de coherencia, se debe utilizar siempre comillas en la asignación de valores a las variables:

Mal:

```
F00=bar
```

Bien:

```
F00="bar"
```

- Si se utilizan múltiples variables, incluir la expresión completa entre comillas dobles:

Mal:

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

Bien:

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

15.5 Miscelánea

- Se debe utilizar `|` (sin comillas) como separador cuando se invoque a `sed`, p.ej. `"sed -e `s|`'"` (Pero sin las comillas `"`)
- No se debe utilizar el comando `test` para hacer comparaciones o pruebas, usar `"[" "` (sin `"`); p.ej. `"if [-x /bin/foo]; ..."` en lugar de `"if test -x /bin/foo; ..."`.
- Se debe utilizar `case` siempre que sea posible en lugar de `test`, ya que es más fácil de leer y más rápido en la ejecución.
- Usar mayúsculas en los nombres de las funciones para evitar confusiones con el entorno de los usuarios.

Procedimientos

16. Procedimientos

Este capítulo documenta los procedimientos dentro del Live Systems Project para diversas tareas que requieren la cooperación con otros equipos de Debian.

16.1 Principales lanzamientos

El lanzamiento de una nueva versión estable de Debian involucra a una gran cantidad de equipos diferentes que trabajan juntos para conseguirlo. En un momento dado, el equipo Live aparece y desarrolla imágenes en vivo del sistema. Los requisitos para ello son:

- Una réplica de las versiones publicadas de los archivos de debian y debian-security a la que pueda acceder el build de debian-live.
- Es necesario conocer el nombre de la imagen (p.ej. debian-live-VERSION-ARCH-FLAVOUR.iso).
- Es necesario sincronizar los datos de debian-cd (lista de exclusión de udeb)
- Las imágenes se crean y se almacenan en cdimage.debian.org.

16.2 Nuevas versiones

- Una vez más, se necesita una réplica actualizada de Debian y debian-security.
- Las imágenes se crean y se almacenan en cdimage.debian.org.
- Correo para enviar notificaciones.

16.2.1 Última actualización de una versión Debian

Recordar que se deben ajustar tanto las réplicas de chroot como las de binary cuando se construye la última serie de imágenes para una versión de Debian después de haber sido trasladada de ftp.debian.org a archive.debian.org. De esta manera, las viejas imágenes prefabricadas siguen siendo útiles, sin modificaciones de los usuarios.

16.2.2 Plantilla para anunciar nuevas versiones.

Se puede generar un anuncio de nuevas versiones usando la siguiente plantilla y el siguiente 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'
```

Revisar el mensaje de correo con cuidado antes de enviarlo a otras personas para su corrección.

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

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

The images are available for download at:

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

and later at:

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


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

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

Additionally it includes the following Live-specific changes:

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

About Live Systems

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

About Debian

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

Contact Information

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

Repositorios Git

17. Repositorios Git

La lista de todos los repositorios disponibles del Live Systems Project está en <http://live-systems.org/gitweb/>. Las URLs git del proyecto tienen la forma: protocolo://live-systems.org/-git/repositorio. Por lo tanto, para clonar *live-manual* en sólo lectura, lanzar:

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

O,

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

O,

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

Las direcciones para clonar con permiso de escritura tienen la forma: git@live-systems.org:/repositorio.

Así que, de nuevo, para clonar *live-manual* a través de ssh escribir:

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

El árbol git se compone de varias ramas diferentes. Las ramas

debian y **debian-next** son particularmente notables porque contienen el trabajo real que, con el tiempo, será incluido en cada nueva versión.

Después de clonar cualquiera de los repositorios existentes, nos encontramos en la rama **debian**. Esto es apropiado para echar un vistazo al estado de la última versión del proyecto, pero antes de empezar a trabajar es fundamental cambiar a la rama **debian-next**. Para ello:

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

La rama **debian-next**, la cual no es siempre fast-forward, es donde se realizan todos los cambios antes de que se fusionen en la rama **debian**. Para hacer una analogía, es como un campo de pruebas. Si se está trabajando en esta rama y se necesita hacer un pull, se tendrá que hacer un `git pull --rebase` para que las modificaciones locales se guarden mientras se actualiza desde el servidor y entonces los cambios locales se pondrán encima de todos los demás.

17.1 Manejo de múltiples repositorios

Si se tiene la intención de clonar varios de los repositorios y cambiar a la rama **debian-next** de inmediato para comprobar el último código, escribir un parche o contribuir con una traducción se debe saber que el servidor proporciona un fichero `mrconfig` para facilitar el manejo de múltiples repositorios. Para utilizarlo es necesario instalar el paquete *mr* y a continuación, lanzar:

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

822

Este comando automáticamente clonará y cambiará a la rama **debian-next** los repositorios de desarrollo de los paquetes Debian producidos por el proyecto. Estos incluyen, entre otros, el repositorio *live-images*, que contiene las configuraciones utilizadas para las imágenes prefabricadas que el proyecto publica para uso general. Para obtener más información sobre cómo utilizar este repositorio, consultar [«Clonar una configuración publicada a través de Git»](#)

Ejemplos

Ejemplos

18. Ejemplos

Este capítulo ofrece ejemplos de creación de imágenes de sistemas en vivo para casos de uso específicos. Si se es nuevo en la creación de una imagen en vivo propia, se recomienda leer primero los tres tutoriales en secuencia, ya que cada uno enseña nuevas técnicas que ayudan a utilizar y entender los ejemplos restantes.

18.1 Uso de los ejemplos

Para poder seguir estos ejemplos es necesario un sistema donde crearlos que cumpla con los requisitos enumerados en **«Requisitos»** y tener *live-build* instalado tal y como se describe en **«Instalación de live-build»**.

Hay que tener en cuenta que, para abreviar, en estos ejemplos no se especifica una réplica local para la creación de la imagen. Es posible acelerar las descargas considerablemente si se utiliza una réplica local. Se puede especificar las opciones cuando se usa `lb config`, tal y como se describe en **«Réplicas de Distribution utilizadas durante la creación»**, o para más comodidad, establecer el valor por defecto para la creación del sistema en `/etc/live/build.conf`. Basta con crear este fichero y en el mismo, establecer las variables `LB_MIRROR_*` correspondientes a la réplica preferida. Todas las demás réplicas usadas en el proceso de creación usarán estos valores por defecto. Por ejemplo:

```
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: Una imagen predeterminada

Caso práctico: Crear una primera imagen sencilla, aprendiendo los fundamentos de *live-build*.

En este tutorial, vamos a construir una imagen ISO híbrida por defecto que contenga únicamente los paquetes base (sin Xorg) y algunos paquetes de soporte, como un primer ejercicio en el uso de *live-build*.

No puede ser más fácil que esto:

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

Si se examina el contenido del directorio `config/` se verá almacenada allí una configuración en esqueleto preparada para ser personalizada o en este caso para ser usada inmediatamente para construir una imagen por defecto.

Ahora, como superusuario, crear la imagen, guardando un log con `tee` mientras se crea.

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

Suponiendo que todo va bien, después de un rato, el directorio actual contendrá `live-image-i386.hybrid.iso`. Esta imagen ISO híbrida se puede arrancar directamente en una máquina virtual como se describe en **«Probar una imagen ISO con Qemu»** y en **«Probar una imagen ISO con VirtualBox»** o bien ser copiada a un medio óptico como un dispositivo USB tal y como se describe en **«Grabar una imagen ISO en un medio físico»** y **«Copiar una imagen ISO híbrida en un dispositivo USB»**, respectivamente.

18.3 Tutorial 2: Una utilidad de navegador web

Caso práctico: Crear una utilidad de navegador web, aprendiendo a aplicar personalizaciones.

En este tutorial, se creará una imagen adecuada para su uso como utilidad de navegador web, esto sirve como introducción a la personalización de las imágenes de sistemas en vivo.

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

La elección de LXDE para este ejemplo refleja el deseo de ofrecer un entorno de escritorio mínimo, ya que el enfoque de la imagen es el uso individual que se tiene en mente, el navegador web. Se podría ir aún más lejos y ofrecer una configuración por defecto para el navegador web en `config/includes.chroot/etc/iceweasel/profile/`, o paquetes adicionales de soporte para la visualización de diversos tipos de contenido web, pero se deja esto como un ejercicio para el lector.

Crear la imagen, de nuevo como superusuario, guardando un log como en el **<Tutorial 1>**:

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

De nuevo, verificar que la imagen está bien y probarla igual que en el **<Tutorial 1>**.

18.4 Tutorial 3: Una imagen personalizada

Caso práctico: Crear un proyecto para conseguir una imagen personalizada, que contenga el software favorito para llevárselo en una memoria USB donde quiera que se vaya, y hacerlo evolucionar en revisiones sucesivas, tal y como vayan cambiando las necesidades y preferencias.

Como nuestra imagen personalizada irá cambiando durante un número de revisiones, si se quiere ir siguiendo esos cambios, probar nuevas cosas de forma experimental y posiblemente volver atrás si no salen bien, se guardará la configuración en el popular sistema de control de versiones git. También se utilizarán las mejores prácticas de configuración automática a través de scripts auto como se describe en **<Gestionar una configuración>**.

18.4.1 Primera revisión

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

Editar `auto/config` del siguiente modo:

```
#!/bin/sh

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

Ejecutar `lb config` para generar el árbol de configuración,

utilizando el script `auto/config` que justo se acaba de crear:

```
$ lb config
```

Completar la lista de paquetes local:

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

En primer lugar con `--architectures i386` se asegura de que en un sistema de creación `amd64` se crea una versión de 32-bits adecuada para ser usada en la mayoría de máquinas. En segundo lugar, se usa `--linux-flavours 686-pae` porque no se espera usar esta imagen en sistemas mucho más viejos. En tercer lugar se elige el metapaquete `lxde` para proporcionar un escritorio mínimo. Y, por último, se añaden dos paquetes iniciales favoritos: *iceweasel* y *xchat*.

Ahora, crear la imagen:

```
# lb build
```

Tener en cuenta que a diferencia de los dos primeros tutoriales, ya no se tiene que escribir `2>&1 |tee build.log` ya que esto se incluye ahora en `auto/build`.

Una vez que se ha probado la imagen (como en el <Tutorial 1>) y se ha asegurado de que funciona, es el momento de iniciar el repositorio git, añadiendo sólo los scripts `auto` que se acaba de crear, y luego hacer el primer commit:

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

18.4.2 Segunda revisión

En esta revisión, vamos a limpiar desde la primera creación, agregar el paquete *vlc* a nuestra configuración, crear de nuevo, probar y enviar los cambios al git.

El comando `lb clean` limpiará todos los ficheros generados en las primeras creaciones a excepción del caché, lo cual ahorra tener que volver a descargar de nuevo los paquetes. Esto asegura que el siguiente `lb build` vuelva a ejecutar todas las fases para regenerar los ficheros de nuestra nueva configuración.

```
# lb clean
```

Añadir ahora el paquete *vlc* a nuestra lista de paquetes local en `config/package-lists/my.list.chroot`:

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

Crear de nuevo:

```
# lb build
```

Probar, y cuando se esté satisfecho, enviar la próxima revisión

al git:

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

Por supuesto, es posible hacer cambios más complicados en la configuración, tal vez añadiendo ficheros en los subdirectorios de `config/`. Cuando se envían nuevas revisiones, hay que tener cuidado de no editar a mano o enviar los ficheros del nivel superior en `config` que contienen variables `LB_*` ya que estos son productos de creación también y son siempre limpiados por `lb clean` y recreados con `lb config` a través de sus respectivos scripts `auto`.

Hemos llegado al final de nuestra serie de tutoriales. Si bien son posibles muchos más tipos de personalización, aunque sólo sea con las pocas características explicadas en estos sencillos ejemplos, se puede crear una variedad casi infinita de imágenes diferentes. Los ejemplos que quedan en esta sección abarcan varios casos de usos diferentes procedentes de las experiencias recogidas de los usuarios de sistemas en vivo.

18.5 Un cliente VNC kiosk

Caso Práctico: Crear una imagen con *live-build* para que se conecte directamente a un servidor VNC al arrancar.

Crear un directorio de construcción y lanzar una configuración de esqueleto en su interior, desactivando «recommends» para conseguir un sistema mínimo. Y a continuación, crear dos listas iniciales de paquetes: La primera generada con un script proporcionado por *live-build* llamado `Packages` (ver «Generar listas de paquetes»), y la segunda lista una que incluya *xorg*, *gdm3*, *metacity* y *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
```

Como se explica en «Ajuste de APT para ahorrar espacio» puede ser necesario volver a agregar algunos paquetes recomendados para que la imagen funcione correctamente.

Una manera fácil de conocer todos los «recommends» es utilizar *apt-cache*. Por ejemplo:

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

En este ejemplo, descubrimos que teníamos que volver a incluir varios paquetes recomendados por *live-config* y *live-boot*: *user-setup* para hacer funcionar el inicio automático de sesión y *sudo* programa esencial para apagar el sistema. Además, podría ser útil añadir *live-tools* para poder copiar la imagen en la memoria RAM y *eject* para finalmente poder expulsar el medio en vivo. Por eso:

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

Después, crear el directorio `/etc/skel` en `config/includes.chroot` y poner dentro un fichero `.xsession` personalizado para el usuario que por defecto ejecutará *metacity* e iniciará el *xvncviewer*, conectando al puerto 5901 de un servidor en `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
```

Crear la imagen:

```
# lb build
```

Disfrutarlo.

18.6 Una imagen básica para un pendrive USB de 128MB

Caso Práctico: Crear una imagen quitando algunos componentes para que quepa en un pendrive USB de 128MB dejando un poco de espacio libre para poder usarlo para lo que se quiera.

Al optimizar una imagen para adaptarla al tamaño de algunos medios de almacenamiento, es necesario comprender el equilibrio que se está haciendo entre tamaño y funcionalidad. En este ejemplo, se recorta tanto sólo para dar cabida a material adicional dentro de un tamaño de 128MB, pero sin hacer nada para destruir la integridad de los paquetes que contiene, tales como la depuración de las variantes locales a través del paquete *localepurge* u otro tipo de optimizaciones «intrusivas». Cabe destacar que se utiliza

--debootstrap-options para crear un sistema mínimo desde el principio.

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

Para hacer que la imagen funcione correctamente, tenemos que volver a añadir, al menos, dos paquetes recomendados, que son excluidos por la opción --apt-recommends false. Ver [«Ajuste de APT para ahorrar espacio»](#)

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

Ahora, crear la imagen de forma habitual:

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

En el sistema del autor, en el momento de escribir esto, la configuración anterior produjo una imagen de 77MB. Esto se compara favorablemente en tamaño con la imagen de 177MB producida por la configuración por defecto en el [«Tutorial 1»](#).

El mayor ahorro de espacio aquí, en comparación con la creación de una imagen por defecto en un sistema de arquitectura i386 es seleccionar sólo la versión del kernel 486 en lugar de la de por defecto -k "486 686-pae". Dejar fuera los índices de APT con --apt-indices false también ahorra una cantidad importante de espacio, la desventaja es que será necesario hacer un apt-get update antes de usar apt en

el sistema en vivo. Excluyendo los paquetes recomendados con `--apt-recommends false` se ahorra un poco de espacio adicional a costa de omitir algunos paquetes que de otro modo podría esperarse que estuvieran allí. `--debootstrap-options "--variant=minbase"` preinstala un sistema mínimo desde el principio. El hecho de no incluir automáticamente paquetes de firmware con `--firmware-chroot false` también ahorra un poco de espacio. Y por último, `--memtest none` evita la instalación de un comprobador de memoria.

Nota: También se puede conseguir un sistema mínimo utilizando scripts gancho como por ejemplo el script `stripped.hook.chroot` que se encuentra en `/usr/share/doc/live-build/examples/hooks`, que puede reducir aún más el tamaño de la imagen hasta 62MB. Sin embargo, el script elimina documentación y otros ficheros de los paquetes instalados en el sistema. Esto viola la integridad de los paquetes y como se comenta en el encabezado del script, puede tener consecuencias imprevistas. Es por eso por lo que el uso de *debootstrap* es el método recomendado para conseguir este objetivo.

18.7 Un escritorio GNOME con variante local e instalador

Caso práctico: Crear una imagen que contenga el escritorio gráfico GNOME, la variante local Suiza y un instalador.

Se desea crear una imagen iso-hybrid para la arquitectura i386 con un escritorio preferido, en este caso el GNOME, que contiene todos los mismos paquetes que serían instalados por el programa de instalación estándar de Debian para GNOME.

El primer problema es descubrir los nombres de las tareas adecuadas. En la actualidad, *live-build* no puede ayudar en

esto. Aunque podríamos tener suerte y encontrarlos a base de pruebas, hay una herramienta, `grep-dctrl`, para extraerlos de las descripciones de tareas en `tasksel-data`, para proceder, asegurarse de tener ambas cosas:

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

Ahora podemos buscar las tareas apropiadas, primero con:

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

Con este comando, se descubre que la tarea se llama, sencillamente, `german`. Ahora, para encontrar las tareas relacionadas:

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

En el momento del arranque se va a generar la variante local **de_CH.UTF-8** y seleccionar la distribución del teclado **ch**. Ahora vamos a poner las piezas juntas. Recordando de **Utilizar metapaquetes** que los metapaquetes tienen el prefijo `task-`, especificamos estos parámetros del lenguaje en el arranque y a continuación añadimos los paquetes de prioridad estándar y los metapaquetes que hemos descubierto a la lista de paquetes de la siguiente manera:

```
$ 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

Tener en cuenta que se ha incluido el paquete *debian-installer-launcher* para lanzar el instalador desde el escritorio en vivo, y que también se ha especificado el kernel 486, ya que actualmente es necesario que el instalador y el kernel del sistema en vivo coincidan para que el lanzador funcione correctamente.

Apéndice

915 Style guide

916 19. Guía de estilo

917 19.1 Instrucciones para los autores

918 Esta sección se ocupa de algunas consideraciones generales a tener en cuenta al escribir documentación técnica para *live-manual*. Se dividen en aspectos lingüísticos y procedimientos recomendados.

919 **Nota:** Los autores deberían leer primero <contribuir a este documento>

920 19.1.1 Aspectos lingüísticos

921 • Utilizar un inglés llano

922 Tener en cuenta que un alto porcentaje de lectores no son hablantes nativos de inglés. Así que, como regla general, se debe utilizar frases cortas y significativas, seguidas de un punto y aparte.

923 Esto no significa que se tenga que utilizar un estilo simplista. Es una sugerencia para tratar de evitar, en la medida de lo posible, las oraciones subordinadas complejas que hacen que el texto sea difícil de entender para los hablantes no nativos de inglés.

924 • Variedad de inglés

925 Las variedades más extendidas del inglés son la británica y la americana, así que es muy probable que la mayoría de los autores utilicen una u otra. En un entorno de colaboración, la variedad ideal sería el “inglés internacional”, pero es muy difícil, por no decir imposible, decidir qué variedad entre todas las existentes, es la mejor.

926

Esperamos que las diferentes variedades puedan mezclarse sin crear malentendidos, pero en términos generales se debe tratar de ser coherente y antes de decidir sobre el uso de las variantes británica o americana, o cualquier otro tipo de inglés a su discreción, por favor, dar un vistazo a cómo escriben otras personas y tratar de imitarlas.

• Ser equilibrado

927

Hay que ser imparcial. Evitar incluir referencias a ideologías totalmente ajenas a *live-manual*. La escritura técnica debe ser lo más neutral posible. Está en la naturaleza misma de la escritura científica.

928

• Ser políticamente correcto

929

Evitar el lenguaje sexista tanto como sea posible. Si se necesita hacer referencia a la tercera persona del singular, utilizar preferiblemente “they” en lugar de “he” o “she” o inventos extraños como “s/he” o “s(he)”.

930

• Ser concisos

931

Ir directamente al grano y no dar vueltas a las cosas. Dar toda la información necesaria, pero no dar más información de la necesaria, es decir, no explicar detalles innecesarios. Los lectores son inteligentes. Se presume algún conocimiento previo de su parte.

932

• Minimizar la labor de traducción

933

Tener en cuenta que cualquier cosa que se escriba tendrá que ser traducido a otros idiomas. Esto implica que un número de personas tendrán que hacer un trabajo extra si se agrega información innecesaria o redundante.

934

• Ser coherente

935

Como se ha sugerido anteriormente, es casi imposible estandarizar un documento creado en colaboración en un

936

todo perfectamente unificado. Sin embargo, se aprecia todo esfuerzo por escribir de manera coherente con el resto de los autores.

• *Cohesión*

Utilizar conectores de discurso para conseguir un texto coherente y sin ambigüedades. (Normalmente se llaman connectors).

• *Ser descriptivo*

Es preferible describir el asunto en uno o varios párrafos que la mera utilización de una serie de oraciones en un típico estilo de “changelog”. Hay que describirlo! Los lectores lo agradecerán.

• *Diccionario*

Buscar el significado de las palabras en un diccionario o una enciclopedia si no se sabe cómo expresar ciertos conceptos en inglés. Pero hay que tener en cuenta que un diccionario puede ser el mejor amigo o puede convertirse en el peor enemigo si no se utiliza correctamente.

El inglés tiene el mayor vocabulario que existe (con más de un millón de palabras). Muchas de estas palabras son préstamos de otras lenguas. Al buscar el significado de las palabras en un diccionario bilingüe la tendencia de un hablante no nativo de inglés es elegir las que suenan más similares en su lengua materna. A menudo, esto se convierte en un discurso excesivamente formal que no suena muy natural en inglés.

Como regla general, si un concepto se puede expresar utilizando diferentes sinónimos, es un buen consejo elegir la primera palabra propuesta por el diccionario. En caso de duda, la elección de palabras de origen germánico (Normalmente palabras monosílabas) es a menudo lo correcto. Tener en

cuenta que estas dos técnicas puede producir un discurso más bien informal, pero al menos la elección de palabras será de amplio uso y aceptación general.

Se recomienda el uso de un diccionario de frases hechas. Son muy útiles cuando se trata de saber qué palabras suelen aparecer juntas.

Una vez más, es una buena práctica aprender del trabajo de los otros. El uso de un motor de búsqueda para comprobar cómo otros autores utilizan ciertas expresiones puede ayudar mucho.

• *Falsos amigos, modismos y otras expresiones idiomáticas*

Cuidado con los falsos amigos. No importa lo competente que se sea en un idioma extranjero, se puede caer de vez en cuando en la trampa de los llamados “falsos amigos”, palabras que se parecen en dos idiomas pero cuyos significados o usos pueden ser completamente diferentes.

Intentar evitar los modismos tanto como sea posible. Los “modismos” son expresiones que tienen un significado completamente diferente de lo que sus palabras parecen decir por separado. A veces, los modismos pueden resultar difíciles de entender incluso para los hablantes nativos de inglés!

• *Evitar el argot, las abreviaturas, las contracciones...*

A pesar de que se anime a utilizar un inglés sencillo, del día a día, la escritura técnica pertenece al registro formal de la lengua.

Intentar evitar el argot, las abreviaturas inusuales que son difíciles de entender y por encima de todo, las contracciones que tratan de imitar el lenguaje hablado. Por no hablar de las expresiones familiares o típicas del irc.

19.1.2 Procedimientos

• Probar antes de escribir

Es importante que los autores prueben sus ejemplos antes de añadirlos a *live-manual* para asegurarse de que todo funciona como se describe. Hacer las pruebas en un entorno chroot limpio o una máquina virtual puede ser un buen punto de partida. Además, sería ideal si las pruebas fueran llevadas a cabo en diferentes máquinas con un hardware diferente para detectar los posibles problemas que puedan surgir.

• Ejemplos

Cuando se pone un ejemplo hay que tratar de ser lo más específico posible. Un ejemplo es, después de todo, tan sólo un ejemplo.

A menudo es mejor utilizar un ejemplo que sólo se aplica a un caso concreto que el uso de abstracciones que puedan confundir a los lectores. En este caso se puede dar una breve explicación de los efectos del ejemplo propuesto.

Puede haber algunas excepciones cuando el ejemplo sugiera el uso de comandos potencialmente peligrosos que, si se utilizan incorrectamente, pueden provocar la pérdida de datos u otros efectos indeseables similares. En este caso se debe proporcionar una explicación detallada acerca de los posibles efectos secundarios.

• Enlaces externos

Los enlaces a sitios externos sólo se deben utilizar cuando la información en esos sitios es crucial para comprender un punto concreto. A pesar de eso, tratar de utilizar enlaces a sitios externos lo menos posible. Los enlaces de Internet pueden cambiar de vez en cuando, dando lugar a enlaces rotos y dejando los argumentos en un estado incompleto.

Además, las personas que leen el manual sin conexión no

953

tendrán la oportunidad de seguir los enlaces.

• Evitar las marcas y las cosas que violan la licencia bajo la que se publica el manual

Tratar de evitar las marcas tanto como sea posible. Tener en cuenta que otros proyectos derivados pueden hacer uso de la documentación que escribimos. Así que estamos complicando las cosas para ellos si se añade determinado material específico.

live-manual se publica bajo la GNU GPL. Esto tiene una serie de implicaciones que se aplican a la distribución de los materiales (de cualquier tipo, incluyendo gráficos o logos con derechos de autor) que se publican en él.

• Escribir un primer borrador, revisar, editar, mejorar, rehacer de nuevo si es necesario

- Lluvia de ideas!. Es necesario organizar las ideas primero en una secuencia lógica de eventos.

- Una vez que, de alguna manera, se han organizado esas ideas en la mente, escribir un primer borrador.

- Revisar la gramática, la sintaxis y la ortografía. Tener en cuenta que los nombres propios de las versiones, tales como **jessie** o **sid**, no se deben escribir en mayúscula cuando se refieren a los nombres en clave. Para comprobar la ortografía se puede ejecutar el target "spell". Es decir, `make spell`

- Mejorar las frases y rehacer cualquier parte si es necesario.

• Capítulos

Utilizar el sistema de numeración convencional de los capítulos y subtítulos. Por ejemplo 1, 1.1, 1.1.1, 1.1.2 ... 1.2, 1.2.1, 1.2.2 ... 2, 2.1 ... y así sucesivamente. Ver marcado a continuación.

Si es necesario enumerar una serie de pasos o etapas en la descripción, también se puede utilizar los números ordinales: primero, segundo, tercero ... o en primer lugar, entonces, después, por fin ... Alternativamente, se pueden utilizar puntos.

- *Marcado*

Y por último, pero no menos importante, *live-manual* utiliza `<SiSU>` para procesar los ficheros de texto y producir múltiples formatos. Se recomienda echar un vistazo al `<manual de SiSU>` para familiarizarme con su marcado, o bien escribir:

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

Estos son algunos ejemplos de marcado que pueden ser útiles:

- Para el énfasis/negrita:

```
*{foo}* o !{foo}!
```

producen: **foo** o **foo** . Se usan para enfatizar ciertas palabras clave.

- Para la cursiva:

```
/ {foo} /
```

produce: *foo*. Se usa, por ejemplo, para los nombres de paquetes Debian.

- Para monospace:

```
# {foo} #
```

produce: `foo`. Se usa, por ejemplo, para los nombres de los comandos. Y también para resaltar algunas palabras clave o cosas como las rutas.

- Para los bloques de código:

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

produce:

```
$ foo
# bar
```

Se utiliza `code{` para abrir y `}code` para cerrar los bloques. Es importante recordar dejar un espacio al principio de cada línea de código.

19.2 Directrices para los traductores

Esta sección se ocupa de algunas consideraciones generales a tener en cuenta al traducir el contenido de *live-manual*.

Como recomendación general, los traductores deberían haber leído y entendido las reglas de traducción que se aplican a sus lenguas específicas. Por lo general, los grupos de traducción y las listas de correo proporcionan información sobre cómo hacer

traducciones que cumplan con los estándares de calidad de Debian.

995 **Nota:** Los traductores también deberían leer <Cómo contribuir a este documento>. En particular, la sección <Traducción>

996 19.2.1 Consejos de traducción

997 • Comentarios

998 El papel del traductor es transmitir lo más fielmente posible el significado de las palabras, oraciones, párrafos y textos de como fueron escritos por los autores originales a su idioma.

999 Así que ellos deben abstenerse de añadir comentarios personales o informaciones adicionales por su cuenta. Si se desea agregar un comentario para los traductores que trabajan en los mismos documentos, se pueden dejar en el espacio reservado para ello. Es decir, el encabezado de las cadenas de los ficheros **po** precedidos por el signo # . La mayoría de los programas gráficos de traducción pueden manejar ese tipo de comentarios automáticamente.

1000 • NT, Nota del Traductor

1001 Es perfectamente aceptable, sin embargo, incluir una palabra o una expresión entre paréntesis en el texto traducido si, y sólo si, hace que el significado de una palabra o expresión difícil sea más clara para el lector. Dentro de los paréntesis, el traductor debe poner de manifiesto que la adición es suya utilizando la abreviatura “NT” o “Nota del traductor”.

1002 • Frases impersonales

1003 Los documentos escritos en Inglés utilizan muchísimo la forma impersonal “you”. En algunos otros idiomas que no comparten esta característica, esto podría dar la falsa impresión de que

los textos originales se dirigen directamente al lector cuando en realidad no lo hacen. Los traductores deben ser conscientes de este hecho y reflejarlo en su idioma con la mayor precisión posible.

• Falsos amigos

1004

La trampa de los “falsos amigos” explicada anteriormente se aplica especialmente a los traductores. Volver a comprobar el significado de los falsos amigos sospechosos en caso de duda.

1005

• Marcado

1006

Los traductores que trabajen inicialmente con los ficheros **pot** y posteriormente con los ficheros **po** encontrarán muchas características de marcado en las cadenas. Se puede traducir el texto, siempre que sea traducible, pero es extremadamente importante que se utilice exactamente el mismo marcado que la versión original en inglés.

1007

• Bloques de código

1008

A pesar de que los bloques de código son generalmente intraducibles, incluirlos en la traducción es la única manera de conseguir una traducción completa al 100%. Y aunque eso signifique más trabajo al principio, ya que puede ser necesaria la intervención de los traductores cuando hay cambios en el código, es la mejor manera, a la larga, de identificar lo que se ha traducido y lo que no al comprobar la integridad de los ficheros .po.

1009

• Saltos de línea

1010

Los textos traducidos deben tener exactamente los mismos saltos de línea que los textos originales. Tener cuidado de presionar la tecla “Enter” o escribir si aparece en los ficheros originales. Estas nuevas líneas aparecen a menudo, por ejemplo, en los bloques de código.

1011

- 1012 No hay que confundirse, esto no significa que el texto traducido
tenga que tener la misma longitud que la versión en inglés. Eso
es prácticamente imposible.
- 1013 • *Cadenas intraducibles*
- 1014 Los traductores nunca deben traducir:
- 1015 - Los nombres en clave de las versiones (que debe ser escrito
en minúsculas)
- 1016 - Los nombres de los programas
- 1017 - Los comandos que se ponen como ejemplos
- 1018 - Los metadatos (aparecen a menudo entre dos puntos
:**metadata:**)
- 1019 - Los enlaces
- 1020 - Las rutas

SiSU Metadata, document information

Título: Manual de Live Systems

Creador: Proyecto Live Systems <debian-live@lists.debian.org>

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

License: Este programa es software libre: puede ser redistribuido y/o modificado
bajo los términos de la GNU General Public License publicada por la Free Software
Foundation, bien de la versión 3 de la Licencia, o (a su elección) cualquier versión
posterior.

Este programa se distribuye con la esperanza de que sea útil, pero SIN NINGUNA
GARANTÍA, incluso sin la garantía implícita de COMERCIALIZACIÓN o IDONEIDAD
PARA UN PROPÓSITO PARTICULAR. Consulte la GNU General Public License para
más detalles.

Debería haber recibido una copia de la General Public License GNU junto con
este programa. Si no, vea <<http://www.gnu.org/licenses/>>.

El texto completo de la GNU Licencia Pública General se pueden encontrar
en /usr/share/common-licenses/GPL-3

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

Fecha: 2014-10-25

Version Information

Fichero fuente: live-manual.ssm.sst

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

Source Digest: SHA256(live-manual.ssm.sst)=a40f0265eca8fdb4a19038-
fc5db629b9a58b4c9950cf25c4acb5e71dedbc8f

Generated

Última generación (metaverse) del documento: 2014-10-25 13:12:50 +0000

Generado por: SiSU 5.7.1 of 2014w41/7 (2014-10-19)

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