Linux From Scratch

Versión 6.0

Gerard Beekmans

Linux From Scratch: Versión 6.0

por Gerard Beekmans

Traducido por el proyecto *LFS-ES*

Versión de la traducción: FINAL del 8 de Diciembre de 2004

Tabla de contenidos

Prólogo

1. Prefacio

Mis aventuras con Linux empezaron hace 6 años cuando descargué e instalé mi primera distribución. Tras trabajar cierto tiempo con ella descubrí algunos aspectos que definitivamente quería ver mejorados. Por ejemplo, no me gustaba la forma en la que estaban organizados los guiones de arranque. Intenté con otras distribuciones para solventar estos detalles, pero todas tenían sus ventajas e inconvenientes. Llegué a darme cuenta de que si quería estar completamente satisfecho con el sistema Linux, tenía que construir el mío propio desde cero.

¿Qué significaba esto? Decidí no utilizar paquetes precompilados de ningún tipo, ni CD-ROMs o discos de arranque que instalasen las utilidades básicas. Quería usar mi actual sistema Linux para desarrollar mi propio sistema personalizado. Este sistema Linux "perfecto" debería tener toda la potencia de los otros sistemas sin sus debilidades. Al principio, la idea fue bastante desalentadora, pero me mantuve aferrado a la idea de que podía construir un sistema que tuviese en consideración mis necesidades y deseos en vez de usar un estándar que no se ajustaba a lo que andaba buscando.

Después de sortear todos los problemas de dependencias circulares y errores de compilación, creé un sistema Linux personalizado hecho a medida y completamente funcional. Este proceso me permitió además crear un sistema compacto y ajustado que era más rápido y ocupaba menos espacio que cualquier sistema operativo tradicional. Llamé a este sistema Linux From Scratch (Linux Desde Cero), o sistema LFS para acortar.

Cuando compartí mis metas y experiencias con otros miembros de la comunidad Linux se hizo palpable que había un amplio interés en las ideas que surgieron de mis aventuras con Linux. No sólo porque dicho sistema LFS de construcción personalizada podía cubrir las especificaciones y requerimientos del usuario, sino también porque ofrecía una gran oportunidad para el aprendizaje a los programadores y administradores de sistemas y ampliar su conocimiento sobre Linux. Con este creciente interés nació el Proyecto Linux From Scratch.

El libro *Linux From Scratch* otorga a los lectores el conocimento y las instrucciones para diseñar y construir un sistema Linux a medida. Este libro resalta el proyecto Linux From Scratch y los beneficios que conlleva el uso de este sistema. Los usuarios pueden definir todos los aspectos de su sistema, incluida la jerarquía de directorios, los guiones de arranque y la seguridad. El sistema resultante se compilará a partir del código fuente y el usuario podrá especificar dónde, por qué y cómo se instalarán los programas. Este libro permite a sus lectores adaptar por completo sus sistemas Linux según sus propias necesidades y ofrece a los usuarios un mayor control sobre el sistema.

Espero que paséis buenos momentos trabajando en vuestro sistema LFS y que disfrutéis de los numerosos beneficios de tener un sistema que es realmente *vuestro*.

--
Gerard Beekmans
gerard@linuxfromscratch.org

2. Audiencia

Existen muchas razones por las que alguien podría querer leer este libro. La principal razón es instalar un sistema Linux a partir del código fuente. La pregunta que mucha gente se hace es "¿Por qué pasar por todo el embrollo de instalar manualmente un sistema Linux desde cero cuando te puedes limitar a descargar e instalar uno ya existente?". Es una buena pregunta y es el motivo de esta sección del libro.

Una importante razón para la existencia de LFS es enseñar a la gente cómo trabaja internamente un sistema Linux. Construir un sistema LFS ayuda a demostrar lo que hace que Linux funcione, cómo trabajan juntas las distintas partes y cómo unas dependen de otras. Una de las mejores cosas que este proceso de aprendizaje proporciona es la habilidad para adaptar Linux a tus propios gustos y necesidades.

Uno de los beneficios claves de LFS es que tienes el control de tu sistema sin tener que confiar en la implementación de Linux de nadie. Con LFS *tu* estás en el asiento del conductor y puedes dictar cada aspecto de tu sistema, como la estructura de directorios y la configuración de los guiones de arranque. También podrás decidir dónde, por qué y cómo se instalan los programas.

Otro beneficio de LFS es que puedes crear un sistema Linux verdaderamente compacto. Cuando instalas una distribución normal acabas instalando muchos programas que probablemente nunca usarás. Tan sólo están ahí ocupando precioso espacio de disco (o peor aún, ciclos de CPU). No es muy difícil conseguir un sistema LFS instalado en menos de 100 MB. ¿Todavía te parece demasiado? Algunos de nosotros hemos estado trabajando para crear un sistema LFS embebido realmente pequeño. Hemos instalado un sistema que contiene lo suficiente para ejecutar un servidor web Apache utilizando tan sólo 8 MB de espacio en disco. Con un repaso adicional para reducirlo, se podría llegar a 5 MB o menos. Intenta eso con una distribución normal. Esta es una de las muchas ventajas que te ofrece diseñar tu propio sistema Linux.

Podríamos comparar una distribución de Linux con una hamburguesa que compras en un restaurante de comida rápida. No tienes idea de lo que te estás comiendo. En cambio, LFS no te da una hamburguesa, sino la receta para hacer la hamburguesa. Te permite revisarla, eliminar los ingredientes no deseados y añadir tus propios ingredientes para mejorar el sabor de tu hamburguesa. Cuando estés satisfecho con la receta entonces empiezas a prepararla. Tu la cocinas de la forma que prefieres: asada, cocida, frita o a la barbacoa.

Otra analogía que podemos usar es comparar a LFS con una casa terminada. LFS te dará los planos de la casa, pero tú debes construirla. Tienes libertad para adaptar los planos durante el proceso, para adaptarlos a tus necesidades y preferencias.

Una última ventaja de un sistema Linux hecho a la medida es la seguridad. Compilando el sistema entero a partir del código fuente tienes la posibilidad de supervisar todo y aplicar todos los parches de seguridad que creas que son necesarios. No tienes que esperar a que alguien te proporcione un nuevo paquete binario que corrija un problema de seguridad. A no ser que examines el nuevo parche y lo implantes por ti mismo no tienes garantía de que ese nuevo paquete se haya construido correctamente y realmente solucione el problema.

El obejtivo de LFS es construir un sistema basado en niveles completo y utilizable. Los lectores que no deseen construir su propio sistema LFS no se podrán beneficiar de la información que hay en este libro. Si sólo quieres saber lo que sucede mientras arranca tu ordenador, entonces te recomendamos el "From Power Up To Bash Prompt" HOWTO (De La Puesta En Marcha Al Indicador De Bash CÓMO) que podrás encontrar en *http://axiom.anu.edu.au/~okeefe/p2b/* o en el sitio web The Linux Documentation Project (TLDP) *http://www.tldp.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html*. Este CÓMO construye un sistema que es similar al de este libro, pero lo enfoca estrictamente hacia la creación de un sistema capaz de iniciar el símbolo del sistema de BASH. Considera tu objetivo. Si lo que quieres es construirte tu propio sistema Linux y aprender mientras lo haces, este libro es la mejor opción.

Hay muy buenas razones para construir tu propio sistema LFS aparte de las aquí listadas. Esta sección es sólo la punta del iceberg. A medida que avances en tu experiencia con LFS encontrarás por ti mismo el poder que la información y el conocimiento realmente brindan.

3. Prerrequisitos

Este libro asume que el lector tienen un conocimiento razonable de la utilización e instalación de software en Linux. Antes de construir un sistema LFS, recomendamos leer los siguientes CÓMOs:

- Software-Building-HOWTO (Construcción de Software CÓMO):
  http://www.tldp.org/HOWTO/Software-Building-HOWTO.html

  Esta es una guía asequible sobre cómo construir e instalar las distribuciones de software Unix "genéricas" bajo Linux.

- The Linux Users' Guide (La Guía del Usuario de Linux).
  Versión en castellano:
  http://es.tldp.org/Manuales-LuCAS/GLUP/glup_0.6-1.1-html-1.1
  Versión en inglés:
  http://espc22.murdoch.edu.au/~stewart/guide/guide.html

  Esta guía cubre el uso de una amplia gama de software Linux.

- The Essential Pre-Reading Hint (Receta de las lecturas previas esenciales):
  http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt

  Esta es una receta del LFS escrita específicamente para los nuevos usuarios de Linux. Incluye un listado de enlaces a excelentes fuentes de información sobre un amplio rango de tópicos. Cualquier persona que intente instalar LFS debería comprender muchos de los tópicos mencionados en esta receta.

4. Tipografía

Para facilitar la comprensión se utilizan ciertas convenciones tipográficas a lo largo del libro. Esta sección contiene algunos ejemplos del formato tipográfico que encontrarás en Linux From Scratch:

```
./configure --prefix=/usr
```

Este tipo de texto está diseñado para teclearse exactamente como aparece, a menos que se indique lo contrario en el texto subyacente. También se utiliza en las secciones explicativas para identificar el comando al que se hace referencia.

```
install-info: unknown option '--dir-file=/mnt/lfs/usr/info/dir'
```

Este tipo de texto (texto de ancho fijo) representa salida por pantalla, probablemente como resultado de la ejecución de comandos. También se usa para especificar nombres de ficheros, como `/etc/ld.so.conf`.

Enfasis

Este tipo de texto se utiliza con varios fines en el libro, principalmente para poner de relieve puntos importantes.

http://www.linuxfromscratch.org/

Este tipo de texto se usa para hipervínculos, tanto dentro de la comunidad LFS como a páginas exteriores, direcciones de descarga, CÓMOs o sitios web.

```
cat > $LFS/etc/group << "EOF"
root:x:0:
bin:x:1:
......
EOF
```

Este formato se usa para la creación de ficheros de configuración. El primer comando solicita al sistema que cree el fichero `$LFS/etc/group` a partir de lo que se teclee en las líneas siguientes, hasta encontrar la secuencia de fín de fichero (EOF). Por lo tanto, la sección entera debe teclearse tal cual.

`[TEXTO A REEMPLAZAR]`

Este formato se utiliza para encapsular texto que no debe ser escrito tal y como aparece.

5. Estructura

Este libro se divide en las siguientes partes:

5.1. Parte I - Introducción

En la Parte I se explican algunas cosas importantes sobre cómo hacer la instalación de LFS. También ofrece información general sobre el libro.

5.2. Parte II - Preparativos para la construcción

La Parte II describe cómo preparar el proceso de construcción: crear una partición, descargar los paquetes y compilar las herramientas temporales.

5.3. Parte III - Construcción del sistema LFS

La Parte III te guía a través de la construcción del sistema LFS: compilar e instalar todos los paquetes uno por uno, activar los guiones de arranque e instalar el núcleo. El sistema Linux obtenido es la base sobre la que podrás construir más software, ampliando tu sistema del modo que prefieras. Al final del libro encontrarás un listado de todos los programas, librerías y ficheros importantes que se han instalado, a modo de referencia rápida.

Parte I. Introducción

Capítulo 1. Introducción

1.1. Cómo construir un sistema LFS

El sistema LFS se construirá utilizando una distribución Linux ya instalada (como Debian, Mandrake, RedHat o SuSE). Este sistema Linux existente (el anfitrión) se utilizará como punto de inicio para suministrar los programas necesarios, como un compilador, un enlazador y un intérprete de comandos, para construir el nuevo sistema. Selecciona la opción “desarrollo” durante la instalación de la distribución para poder acceder a estas herramientas.

El Capítulo 2 de este libro describe cómo crear una nueva partición nativa Linux y un sistema de ficheros, el sitio donde se compilará e instalará el nuevo sistema LFS. El Capítulo 3 explica qué paquetes y parches deben descargarse para construir un sistema LFS y cómo guardarlos en el nuevo sistema de ficheros. El Capítulo 4 muestra cómo configurar un entorno de trabajo adecuado. Por favor, lee con detenimiento el Capítulo 4, pues explica diversos temas importantes a tener en cuenta antes de empezar a trabajar en el Capítulo 5 y posteriores.

En el Capítulo 5 se describe la instalación de una serie de paquetes que formarán el entorno básico de desarrollo (o herramientas principales) utilizado para construir el sistema real en el Capítulo 6. Varios de estos paquetes son necesarios para resolver dependencias circulares. Por ejemplo, para compilar un compilador necesitas un compilador.

El Capítulo 5 muestra también al usuario cómo construir en una primera fase las herramientas principales, compuestas por Binutils y GCC (primera fase significa, básicamente, que estos dos paquetes centrales se instalarán una segunda vez). Los programas de estos paquetes se enlazarán estáticamente para poder utilizarlos independientemente del sistema anfitrión. El siguiente paso es construir Glibc, la librería C. Esta será compilada con los programas de las herramientas principales construidas en la primera fase. Entonces se construirá una segunda fase de las herramientas principales. Esta vez se enlazarán dinámicamente contra la recién construida Glibc. Todos los restantes paquetes del Capítulo 5 se construirán usando esta segunda fase de las herramientas principales. Cuando esto esté hecho, el proceso de instalación de LFS ya no dependerá de la distribución anfitriona, con la excepción del núcleo en ejecución.

Aunque esto puede parecer mucho trabajo para aislarse de la distribución anfitriona, al comienzo del Capítulo 5 se da una explicación técnica completa, incluyendo algunas notas sobre las diferencias entre programas enlazados estática y dinámicamente.

En el Capítulo 6 se construye el auténtico sistema LFS. Se utiliza el programa **chroot** (change root, cambio de raíz) para entrar en un entorno virtual y ejecutar un nuevo intérprete de comandos cuyo directorio raíz será la partición LFS. Esto es muy similar a reiniciar e indicarle al núcleo que monte la partición LFS como partición raíz. El sistema no es realmente reiniciado, si no que se cambia la raíz, porque crear un sistema arrancable requiere un trabajo adicional que no es necesario aún. La mayor ventaja es que “cambiar la raíz” permite seguir usando el sistema anfitrión mientras se construye el LFS. Mientras espera que se complete la compilación de un paquete, el usuario puede cambiar a otra consola virtual (VC) o escritorio X y continuar usando el ordenador normalmente.

Para terminar la instalación, en el Capítulo 7 se configuran los guiones de arranque, y el núcleo y el gestor de arranque se configuran en el Capítulo 8. El Capítulo 9 contiene información para profundizar en la experiencia LFS después de este libro. Tras completar los pasos de este libro, el ordenador estará preparado para reiniciarse dentro del nuevo sistema LFS.

Este es el proceso en pocas palabras. La información detallada sobre cada paso a dar se expone en los siguientes capítulos y descripciones de los paquetes. Los temas que pueden parecer complicados se aclararán y todo estará en su sitio a medida que te embarques en la aventura del LFS.

1.2. Recursos

1.2.1. FAQ

Si durante la construcción del sistema LFS encuentras algún fallo, tienes preguntas, o encuentras un error tipográfico en el libro, consulta primero las FAQ (Preguntas Hechas Frecuentemente) que se encuentran en *http://www.linuxfromscratch.org/faq/*.

En *http://www.lfs-es.com/lfs-es/faq* tienes una versión en castellano, aunque actualmente está muy desfasada.

1.2.2. Listas de correo

El servidor `linuxfromscratch.org` hospeda una serie de listas de correo utilizadas para el desarrollo del proyecto LFS. Estas incluyen, entre otras, las listas principales de desarrollo y soporte.

Para obtener información relacionada con las listas disponibles, cómo suscribirse a ellas, localización de los archivos, etc, visita *http://www.linuxfromscratch.org/mail.html*.

La comunidad hispanoparlante dispone de dos listas de correo que no pertenecen al servidor `linuxfromscratch.org`:

- Soporte, ayuda y noticias sobre LFS:
  http://www.linuxauen.net/mailman/listinfo/linux-desde-cero
- Coordinación de la traducción de LFS al castellano:
  http://listas.escomposlinux.org/mailman/listinfo/lfs-es

1.2.3. IRC

Varios miembros de la comunidad LFS ofrecen asistencia técnica en nuestro servidor IRC. Antes de utilizar este método de ayuda te pedimos que compruebes si en las FAQ de LFS o en los archivos de las listas de correo se encuentra la respuesta a tu problema. Puedes entrar al servidor IRC a través de `irc.linuxfromscratch.org` o `irc.linux-phreak.net`. El canal de soporte se llama #LFS-support.

1.2.4. Servidor de noticias

Las listas de correo hospedadas en `linuxfromscratch.org` también son accesibles a través de un servidor NNTP. Todos los mensajes publicados en una lista de correo son copiados en el grupo de noticias correspondiente y viceversa.

El servidor de noticias es `news.linuxfromscratch.org`.

1.2.5. Wiki

Para obtener más información sobre un paquete, actualización de versiones, trucos, experiencias personales y más cosas, mira el Wiki de LFS en *http://wiki.linuxfromscratch.org/*. Los usuarios también pueden añadir información para ayudar a otros con sus actividades futuras en el LFS.

1.2.6. Referencias

En *http://www.linuxfromscratch.org/~matthew/LFS-references.html* tienes a tu disposición unos apuntes útiles con información adicional sobre los paquetes.

1.2.7. Servidores alternativos

El proyecto LFS tiene por todo el mundo varios servidores alternativos para facilitar el acceso a las páginas web y la descarga de los paquetes requeridos. Por favor, visita el sitio web *http://www.linuxfromscratch.org/* para consultar la lista de los servidores alternativos actuales.

El proyecto LFS-ES, que se ocupa de la traducción al castellano de los textos del LFS, dispone de los siguientes servidores:

- EcolNet, España [Varios servidores] - *http://www.escomposlinux.org/lfs-es/*
- Balaguer, España [300 Kbits] - *http://www.macana-es.com/*
- Dattatec.com, Argentina [100 Mbits] - *http://www.lfs-es.com/*

1.2.8. Información de contacto

Por favor, envía todas tus preguntas y comentarios a una de las listas de correo de LFS o LFS-ES (ver arriba).

1.3. Ayuda

Si mientras estás usando este libro te surge algún problema o duda, consulta primero las FAQ que hay en *http://www.linuxfromscratch.org/faq/#generalfaq* (hay una versión anticuada en castellano en *http://www.lfs-es.com/lfs-es/faq*). Probablemente tu pregunta esté contestada aquí. Si no es así, prueba a encontrar la fuente del problema. La siguiente receta puede darte algunas ideas para encontrar la solución: *http://www.linuxfromscratch.org/hints/downloads/files/errors.txt*.

También tenemos una maravillosa comunidad LFS que está encantada de ofrecer ayuda a través del canal IRC y las listas de correo (mira el Capítulo 1 - Listas de correo). Para ayudarles a diagnosticar y resolver el problema, incluye toda la información relevante que sea posible en tu petición de ayuda.

1.3.1. Cosas a mencionar

Además de una breve explicación del problema experimentado, las cosas esenciales que se deben incluir en la petición de ayuda son:

- La versión del libro que se está usando (en este caso, 6.0).
- La distribución anfitriona (y su versión) usada como base para crear el LFS.
- El paquete o sección en el que se encontró el problema.
- El mensaje de error exacto o los síntomas que aparecen.
- Si te has desviado o no del libro.

Nota

Desviarse del libro *no* implica que no vayamos a ayudarte. Después de todo, LFS se basa en la elección. Avisarnos sobre cualquier cambio en el procedimiento establecido nos ayudará a detectar las posibles causas de tu problema.

1.3.2. Problemas de configuración

Cuando algo va mal en la fase en que la se ejecuta el guión configure, consulta el fichero `config.log`. Este fichero puede contener errores encontrados durante la configuración que no se muestran en pantalla. Incluye esas líneas relevantes si necesitas pedir ayuda.

1.3.3. Problemas de compilación

Tanto la salida por pantalla como el contenido de varios ficheros son útiles para determinar la causa de los problemas de compilación. La salida por pantalla del guión **./configure** y del comando **make** pueden ser útiles. No es necesario incluir toda la salida, pero incluye suficiente información relevante. A continuación hay un ejemplo del tipo de información a incluir de una salída por pantalla de **make**:

```
gcc -DALIASPATH=\"/mnt/lfs/usr/share/locale:.\"
-DLOCALEDIR=\"/mnt/lfs/usr/share/locale\"
-DLIBDIR=\"/mnt/lfs/usr/lib\"
-DINCLUDEDIR=\"/mnt/lfs/usr/include\" -DHAVE_CONFIG_H -I. -I.
-g -O2 -c getopt1.c
gcc -g -O2 -static -o make ar.o arscan.o commands.o dir.o
expand.o file.o function.o getopt.o implicit.o job.o main.o
misc.o read.o remake.o rule.o signame.o variable.o vpath.o
default.o remote-stub.o version.o opt1.o
-lutil job.o: In function `load_too_high':
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference
to `getloadavg'
collect2: ld returned 1 exit status
make[2]: *** [make] Error 1
make[2]: Leaving directory `/lfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/lfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2
```

En este caso, mucha gente simplemente incluye la sección final a partir de:

```
make [2]: *** [make] Error 1
```

Esto no es suficiente información para diagnosticar el problema porque sólo nos dice que algo fue mal, no *qué* fue mal. Lo que se debería incluir para resultar útil es la sección completa tal y como aparece en el ejemplo anterior, ya que incluye el comando que se estaba ejecutando y sus mensajes de error.

En *http://catb.org/~esr/faqs/smart-questions.html* hay disponible un artículo excelente sobre cómo buscar ayuda en Internet. Lee y sigue los consejos de este documento para aumentar las posibilidades de obtener la ayuda que necesitas.

1.3.4. Problemas en los bancos de pruebas

Muchos paquetes proporcionan un banco de pruebas que, dependiendo de la importancia del paquete, debería ejecutarse. En ocasiones los paquetes generarán fallos falsos o esperados. Si te encuentras con ellos, puedes comprobar la página Wiki de LFS en *http://wiki.linuxfromscratch.org/* para ver si nosotros ya los hemos investigado y anotado. Si ya están anotados y localizados, no hay necesidad de preocuparse.

1.4. Sobre el CD incluido

Para tu conveniencia, a este libro le acompaña un CD que contiene las fuentes de los paquetes necesarios para crear un sistema Linux From Scratch. El CD es autoarrancable y proporciona un entorno de trabajo estable para construir el LFS. Este libro se refiere a dicho sistema como el “sistema anfitrión”.

En complemento a las herramientas requeridas para construir el LFS, el sistema anfitrión del CD incluye una serie de otras herramientas útiles instaladas:

- Una versión de este libro en HTML (en inglés)
- El entorno X Window System
- Herramientas web
  - Wget (descargador de ficheros en línea de comandos)
  - Lynx (navegador web en modo texto)
  - Irssi (cliente IRC para la consola)
  - Firefox (navegador web en modo gráfico)
  - Xchat (cliente IRC basado en las X)
- Editores de texto
  - Vim
  - Nano
- Herramientas de red
  - SSH, servidor y cliente
  - NFS, servidor y cliente
  - Smbmount (mount.cifs) para comparticiones Windows
  - Subversion
  - Dhcpcd (cliente DHCP)
- Programas de sistemas de ficheros
  - Reiserfsprogs
  - Xfsprogs
- nALFS - Una herramienta para automatizar la construcción de LFS

La imagen ISO del CD puede descargarse de estas direcciones:
ftp://ftp.lfs-matrix.de/pub/lfs-boot-cd/ (Alemania)
http://cis.sac.accd.edu/lfs/ (Texas, USA)

Capítulo 2. Preparar una nueva partición

2.1. Introducción

En este capítulo se preparará la partición que contendrá el sistema LFS. Se creará la propia partición, se creará un sistema de ficheros en ella y se montará.

2.2. Crear una nueva partición

Para construir un nuevo sistema Linux se requiere espacio en forma de una partición de disco vacía. Si el ordenador no tiene una partición libre o sitio en ninguno de los discos duros para crear una, LFS puede construirse en la misma partición en la que está instalada la distribución actual.

Nota
Este procedimiento avanzado no es recomendable para tu primera instalación de LFS, pero si andas escaso de espacio en disco el siguiente documento puede ayudarte: *http://www.lfs-es.com/recetas/lfs_next_to_existing_systems.html* (la versión original en inglés se encuentra en *http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing_systems.txt*).

Un sistema mínimo necesita una partición de 1,3 GB más o menos. Esto es suficiente para almacenar todos los archivos de código fuente y compilar los paquetes. Sin embargo, si se piensa usar el sistema LFS como sistema Linux principal probablemente se instalará software adicional, necesitando más espacio (2 o 3 GB). El propio sistema LFS no ocupa mucho espacio. Una gran parte de este espacio es requerido para proporcionar suficiente espacio libre temporal. Compilar paquetes puede necesitar mucho espacio en disco que será liberado tras instalar el paquete.

Como casi nunca hay suficiente memoria RAM disponible para los procesos de compilación, es buena idea utilizar una pequeña partición como espacio de intercambio (swap). Este espacio lo usa el núcleo para almacenar los datos menos usados y hacer sitio en memoria para los procesos activos. La partición de intercambio para el sistema LFS puede ser la misma del sistema anfitrión, por lo que no hace falta crear otra si el sistema anfitrión tiene una activada.

Inicia un programa de particionado como **cfdisk** o **fdisk** pasándole como argumento el nombre del disco duro en el que debe crearse la nueva partición, por ejemplo `/dev/hda` para el disco IDE primario. Crea una partición Linux nativa y, si hace falta, una partición de intercambio. Por favor, consulta la página de manual de **cfdisk** o de **fdisk** si todavía no sabes cómo usar estos programas.

Recuerda la denominación de tu nueva partición (por ejemplo, `hda5`). Este libro se referirá a ella como la partición LFS. Recuerda también la denominación de la partición de intercambio. Estos nombres se necesitarán posteriormente para el fichero `/etc/fstab`.

2.3. Crear un sistema de ficheros en la partición

Ahora que hay preparada una partición en blanco ya puede crearse el sistema de ficheros. El más usado en el mundo de Linux es el llamado “second extended file system” (segundo sistema de ficheros extendido, o ext2), pero con la gran capacidad de los discos duros actuales los llamados sistemas de ficheros con registro de transacciones (journaling) se están haciendo muy populares. Aquí se creará un sistema de ficheros ext2, pero en *http://www.lfs-es.com/blfs-es-CVS/postlfs/filesystems.html* podrás encontrar la instrucciones de instalación para otros sistemas de ficheros (la versión original en inglés se encuentra en *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/filesystems.html*).

Para crear un sistema de ficheros ext2 en la partición LFS, ejecuta lo siguiente:

```
mke2fs /dev/[xxx]
```

Sustituye *[xxx]* por el nombre de la partición LFS (`hda5` en nuestro ejemplo anterior).

Si se creó una partición de intercambio (swap), también será necesario inicializarla (también conocido como formatearla, como hiciste anteriormente con **mke2fs**) ejecutando lo siguiente. Si utilizas una partición de intercambio ya existente, no es necesario formatearla:

```
mkswap /dev/[yyy]
```

Sustituye *[yyy]* por el nombre de la partición de intercambio.

2.4. Montar la nueva partición

Ahora que se ha creado un sistema de ficheros es necesario hacer accesible la partición. Para esto debe montarse en el punto de montaje elegido. Para los propósitos de este libro se asume que el sistema de ficheros se monta en `/mnt/lfs`, pero la elección del directorio se deja para tí.

Elige un punto de montaje y asígnalo a la variable de entorno `LFS` ejecutando:

```
export LFS=/mnt/lfs
```

Crea el punto de montaje y monta el sistema de ficheros LFS ejecutando:

```
mkdir -p $LFS
mount /dev/[xxx] $LFS
```

Sustituye *`[xxx]`* por el nombre de la partición LFS.

Si utilizas múltiples particiones para LFS (digamos que una para `/` y otra para `/usr`) móntalas usando:

```
mkdir -p $LFS
mount /dev/[xxx] $LFS
mkdir $LFS/usr
mount /dev/[yyy] $LFS/usr
```

Sustituye *`[xxx]`* e *`[yyy]`* por los nombres de partición apropiados.

Asegúrate de que esta nueva partición no se monte con permisos muy restrictivos (como las opciones *`nosuid`*, *`nodev`* o *`noatime`*). Ejecuta el comando **mount** sin parámetros para ver con qué opciones está montada la partición LFS. Si ves *`nosuid`*, *`nodev`* o *`noatime`*, necesitarás remontarla.

Ahora que se ha establecido un lugar en el que trabajar, es hora de descargar los paquetes.

Parte II. Preparativos para la construcción

Capítulo 3. Paquetes y parches

3.1. Introducción

Este capítulo incluye una lista con los paquetes que se han de descargar para construir un sistema Linux básico. Los números de versión listados corresponden a versiones de los programas que se sabe que funcionan y este libro se basa en ellos. Recomendamos encarecidamente que no uses versiones más nuevas, pues los comandos de construcción para una versión puede que no funcionen con la nueva. Los paquetes más nuevos pueden también tener problemas para los cuales no se han desarrollado aún soluciones.

Todas las URLs, cuando es posible, apuntan a la página de información del paquete que hay en *http://www.freshmeat.net/*. Las páginas de Freshmeat proporcionan un acceso fácil a los sitios oficiales de descarga, así como a los sitios web del proyecto, listas de correo, FAQs, historiales de modificaciones y más cosas.

Las localizaciones de descarga puede que no estén siempre disponibles. En el caso de que una localización de descarga haya cambiado desde la publicación de este libro, Google (*http://www.google.com*) es una útil herramienta de búsqueda para muchos paquetes. Si la búsqueda no da resultados, prueba uno de los métodos alternativos de descarga listados en *http://www.linuxfromscratch.org/lfs/packages.html*.

Será necesario guardar todos los paquetes y parches descargados en algún sitio que esté disponible durante toda la construcción. También se necesita un directorio de trabajo en el que desempaquetar las fuentes y construirlas. Puede usarse `$LFS/sources` tanto para almacenar los paquetes y parches como directorio de trabajo. Al usar este directorio, los elementos requeridos se encontrarán en la partición LFS y estarán disponibles durante todas las fases del proceso de construcción.

Para crear este directorio, ejecuta el siguiente comando como usuario *root* antes de comenzar la sesión de descarga:

```
mkdir $LFS/sources
```

Haz este directorio escribible y pegajoso (sticky). “Pegajoso” significa que aunque diversos usuarios tengan permisos de escritura en un mismo directorio, sólo el propietario de un fichero puede borrarlo. El siguiente comando activará los modos de escritura y pegajoso:

```
chmod a+wt $LFS/sources
```

3.2. Todos los paquetes

Descarga u obtén por otros métodos los siguientes paquetes:

- Autoconf (2.59) - 903 kilobytes (KB):
  http://freshmeat.net/projects/autoconf/
- Automake (1.9.1) - 681 KB:
  http://freshmeat.net/projects/automake/
- Bash (3.0) - 1,910 KB:
  http://freshmeat.net/projects/gnubash/
- Binutils (2.15.91.0.2) - 10,666 KB:
  http://freshmeat.net/projects/binutils/?branch_id=12688
- Bison (1.875a) - 796 KB:
  ftp://ftp.linuxfromscratch.org/pub/lfs/lfs-packages/conglomeration/bison/
- Bzip2 (1.0.2) - 650 KB:
  http://freshmeat.net/projects/bzip2/
- Coreutils (5.2.1) - 3,860 KB:
  http://freshmeat.net/projects/coreutils/
- DejaGNU (1.4.4) - 1,055 KB:
  http://freshmeat.net/projects/dejagnu/
- Diffutils (2.8.1) - 762 KB:
  http://freshmeat.net/projects/diffutils/
- E2fsprogs (1.35) - 3,003 KB:
  http://freshmeat.net/projects/e2fsprogs/
- Expect (5.42.1) - 510 KB:
  http://freshmeat.net/projects/expect/
- File (4.10) - 356 KB:
  http://freshmeat.net/projects/file/

Nota

File (4.10) puede que no esté disponible en la localización indicada. En ocasiones los administradores de la localización principal de descarga eliminan las versiones antiguas cuando se libera una nueva. Una localización de descarga alternativa que puede tener disponible la versión correcta es *ftp://ftp.linuxfromscratch.org/pub/lfs/*.

- Findutils (4.1.20) - 760 KB:
  http://freshmeat.net/projects/findutils/
- Flex (2.5.31) - 372 KB:
  http://freshmeat.net/projects/flex/
- Gawk (3.1.4) - 1,692 KB:
  http://freshmeat.net/projects/gnuawk/

- GCC (3.4.1) - 27,000 KB:
  http://freshmeat.net/projects/gcc/
- Gettext (0.14.1) - 6,397 KB:
  http://freshmeat.net/projects/gettext/
- Glibc (2.3.4-20040701) - 13,101 KB:
  http://freshmeat.net/projects/glibc/

Nota

Los paquetes liberados de Glibc no son lo suficientemente nuevos para nuestro propósito, así que crea el paquete apropiado a partir de una captura CVS con los siguientes comandos:

```
cvs -z 3 -d \
    :pserver:anoncvs@sources.redhat.com:/cvs/glibc \
    export -d glibc-2.3.4-20040701 \
    -D "2004-07-01 17:30 UTC" libc
sed -i -e "s/stable/2004-07-01/" \
    -e "s/2\.3\.3/2.3.4/" \
    glibc-2.3.4-20040701/version.h
tar jcvf glibc-2.3.4-20040701.tar.bz2 \
    glibc-2.3.4-20040701
```

Alternativamente, el equipo del LFS ha creado un paquete que puede descargarse de cualquiera de los sitios réplica FTP listados en el sitio web de LFS en *http://www.linuxfromscratch.org/lfs/packages.html#http*. Puede encontrarse bajo el directorio `/pub/lfs/packages/conglomeration/glibc`. El paquete está firmado usando GPG y recomendamos encarecidamente que se verifique su autenticidad antes de usarlo. Las instrucciones para instalar GnuPG, que permite hacer la verificación, se muestran en el libro BLFS en *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/gnupg.html*.

- Grep (2.5.1) - 545 KB:
  http://freshmeat.net/projects/grep/
- Groff (1.19.1) - 2,360 KB:
  http://freshmeat.net/projects/groff/
- Grub (0.95) - 902 KB:
  ftp://alpha.gnu.org/pub/gnu/grub/
- Gzip (1.3.5) - 324 KB:
  ftp://alpha.gnu.org/gnu/gzip/
- Iana-Etc (1.01) - 161 KB:
  http://freshmeat.net/projects/iana-etc/
- Inetutils (1.4.2) - 1,019 KB:
  http://freshmeat.net/projects/inetutils/
- IPRoute2 (2.6.8-040823) - 264 KB:
  http://developer.osdl.org/dev/iproute2/download/
- Kbd (1.12) - 617 KB:
  http://freshmeat.net/projects/kbd/
- Less (382) - 259 KB:
  http://freshmeat.net/projects/less/

- LFS-Bootscripts (2.2.2) - 16 KB:
  http://downloads.linuxfromscratch.org/
- Libtool (1.5.8) - 2,602 KB:
  http://freshmeat.net/projects/libtool/
- Linux (2.6.8.1) - 34,793 KB:
  http://freshmeat.net/projects/linux/?branch_id=46339
- Linux-Libc-Headers (2.6.8.1) - 2,602 KB:
  http://ep09.pld-linux.org/~mmazur/linux-libc-headers/
- M4 (1.4.2) - 337 KB:
  http://freshmeat.net/projects/gnum4/
- Make (3.80) - 899 KB:
  http://freshmeat.net/projects/gnumake/
- Man (1.5o) - 223 KB:
  http://freshmeat.net/projects/man/
- Man-pages (1.67) - 1,586 KB:
  http://freshmeat.net/projects/man-pages/
- Mktemp (1.5) - 69 KB:
  http://freshmeat.net/projects/mktemp/
- Module-Init-Tools (3.0) - 118 KB:
  ftp://ftp.kernel.org/pub/linux/utils/kernel/module-init-tools/
- Ncurses (5.4) - 2,019 KB:
  http://freshmeat.net/projects/ncurses/
- Patch (2.5.4) - 182 KB:
  http://freshmeat.net/projects/patch/
- Perl (5.8.5) - 9,373 KB:
  http://freshmeat.net/projects/perl/
- Procps (3.2.3) - 265 KB:
  http://freshmeat.net/projects/procps/
- Psmisc (21.5) - 375 KB:
  http://freshmeat.net/projects/psmisc/
- Readline (5.0) - 940 KB:
  http://freshmeat.net/projects/gnureadline/
- Sed (4.1.2) - 749 KB:
  http://freshmeat.net/projects/sed/
- Shadow (4.0.4.1) - 795 KB:
  http://freshmeat.net/projects/shadow/
- Sysklogd (1.4.1) - 80 KB:
  http://freshmeat.net/projects/sysklogd/
- Sysvinit (2.85) - 91 KB:
  http://freshmeat.net/projects/sysvinit/

- Tar (1.14) - 1,025 KB:
  http://freshmeat.net/projects/tar/
- Tcl (8.4.7) - 3,363 KB:
  http://freshmeat.net/projects/tcltk/
- Texinfo (4.7) - 1,385 KB:
  http://freshmeat.net/projects/texinfo/
- Udev (030) - 374 KB:
  ftp://ftp.kernel.org/pub/linux/utils/kernel/hotplug/
- Configuración de permisos de Udev - 2 KB:
  http://downloads.linuxfromscratch.org/udev-config-2.permissions
- Configuración de reglas de Udev - 1 KB:
  http://downloads.linuxfromscratch.org/udev-config-1.rules
- Util-linux (2.12b) - 1,921 KB:
  http://freshmeat.net/projects/util-linux/
- Vim (6.3) - 3,612 KB:
  http://freshmeat.net/projects/vim/
- Ficheros de lenguaje de Vim (6.3) (opcional) - 1,033 KB:
  http://freshmeat.net/projects/vim/
- Zlib (1.2.1) - 277 KB:
  http://freshmeat.net/projects/zlib/

Tamaño total de estos paquetes: 135 MB

3.3. Parches necesarios

Aparte de los paquetes, también se necesitan varios parches. Estos parches corrigen pequeños errores en los paquetes que debería solucionar su desarrollador. Los parches también hacen pequeñas modificaciones para facilitar el trabajo con el paquete. Los siguientes parches son necesarios para construir un sistema LFS:

- Bash Display Wrap Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/bash-3.0-display_wrap-1.patch
- Coreutils Suppress Uptime, Kill, Su Patch - 16 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/coreutils-5.2.1-suppress_uptime_kill_su-1.patch
- Coreutils Uname Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/coreutils-5.2.1-uname-2.patch
- Expect Spawn Patch - 6 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/expect-5.42.1-spawn-1.patch
- Flex Brokenness Patch - 8 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/flex-2.5.31-debian_fixes-2.patch
- GCC Linkonce Patch - 12 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-linkonce-1.patch
- GCC No-Fixincludes Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-no_fixincludes-1.patch
- GCC Specs Patch - 11 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-specs-1.patch
- Inetutils Kernel Headers Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/inetutils-1.4.2-kernel_headers-1.patch
- Inetutils No-Server-Man-Pages Patch - 4 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/inetutils-1.4.2-no_server_man_pages-1.patch
- IPRoute2 Disable DB Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/iproute2-2.6.8_040823-remove_db-1.patch
- Man 80-Columns Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/man-1.5o-80cols-1.patch
- Mktemp Tempfile Patch - 3 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/mktemp-1.5-add_tempfile-1.patch
- Perl Libc Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/perl-5.8.5-libc-1.patch
- Readline Display Wrap Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/readline-5.0-display_wrap-1.patch
- Sysklogd Kernel Headers Patch - 3 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysklogd-1.4.1-kernel_headers-1.patch
- Sysklogd Signal Handling Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysklogd-1.4.1-signal-1.patch
- Sysvinit /proc Title Length Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysvinit-2.85-proclen-1.patch

- Texinfo Segfault Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/texinfo-4.7-segfault-1.patch
- Util-Linux Sfdisk Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/util-linux-2.12b-sfdisk-2.patch
- Zlib Security Patch - 1KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/zlib-1.2.1-security-1.patch

Aparte de los anteriores parches necesarios, hay una serie de parches opcionales creados por la comunidad LFS. Estos parches opcionales solucionan pequeños problemas, o activan alguna funcionalidad que no lo está por defecto. Eres libre de examinar la base de datos de parches que se encuentra en *http://www.linuxfromscratch.org/patches* y elegir cualquier parche adicional que cubra las necesidades del sistema.

Capítulo 4. Últimos preparativos

4.1. Sobre $LFS

Durante este libro la variable de entorno LFS se usará frecuentemente. Es importante que esta variable esté siempre definida. Debería establecerse al punto de montaje que elegiste para tu partición LFS. Comprueba que tu variable LFS está correctamente establecida con:

```
echo $LFS
```

Asegúrate de que la salida muestra la ruta a tu punto de montaje de la partición LFS, que es /mnt/lfs si seguiste el ejemplo aquí usado. Si la salida es errónea, puedes establecer la variable con:

```
export LFS=/mnt/lfs
```

Tener establecida esta variable significa que si se indica que ejecutes un comando como **mkdir $LFS/tools**, puede teclearse literalmente. El intérprete de comandos sustituirá “$LFS” con “/mnt/lfs” (o aquello a lo que esté establecida la variable) cuando procese la línea de comandos.

No olvides comprobar que $LFS esté establecida cada vez que salgas y vuelvas al entorno (o cuando hagas “su” a *root* o a otro usuario).

4.2. Creación del directorio $LFS/tools

Todos los programas compilados en el Capítulo 5 se instalarán bajo `$LFS/tools` para mantenerlos separados de los programas compilados en el Capítulo 6. Los programas compilados aquí son sólo herramientas temporales y no formarán parte del sistema LFS final. Al mantener estos programas en un directorio aparte podrán eliminarse fácilmente tras su uso. Esto también evita que acaben en los directorios de producción de tu anfitrión (que es fácil que ocurra por accidente en el Capítulo 5).

Crea el directorio necesario ejecutando lo siguiente como *root*:

```
mkdir $LFS/tools
```

El próximo paso es crear un enlace `/tools` en el sistema anfitrión. Este apuntará al directorio que acabamos de crear en la partición LFS. Ejecuta este comando también como *root*:

```
ln -s $LFS/tools /
```

Nota

El comando anterior es correcto. El comando **ln** tiene bastantes variaciones de sintaxis, por lo que asegúrate de comprobar su página de manual e info antes de informar de lo que puedes pensar que es un error.

El enlace simbólico creado posibilita que el conjunto de herramientas se compile siempre en referencia a `/tools`, de forma que el compilador, ensamblador y enlazador funcionarán en este capítulo (en el que todavía estamos utilizando algunas herramientas del sistema anfitrión) y en el siguiente (cuando “cambiemos la raíz” a la partición LFS).

4.3. Añadir el usuario lfs

Si se trabaja como *root*, un simple error puede dañar o destruir un sistema. Por tanto recomendamos construir los paquetes del siguiente capítulo como un usuario sin privilegios. Puedes usar tu propio nombre de usuario, pero para facilitar la creación de un entorno de trabajo limpio, crea un nuevo usuario llamado *lfs* como miembro de un nuevo grupo (llamado también *lfs*) y utilízalo para el proceso de construcción. Como *root*, ejecuta el siguiente comando para añadir el nuevo usuario:

```
groupadd lfs
useradd -s /bin/bash -g lfs -m -k /dev/null lfs
```

Significado de las opciones:

`-s /bin/bash`
Esto hace de **bash** el intérprete de comandos por defecto para el usuario *lfs*.

`-g lfs`
Esta opción añade el usuario *lfs* al grupo *lfs*.

`-m`
Esto crea el directorio personal para *lfs*.

`-k /dev/null`
Este parámetro evita que se copien ficheros procedentes de un posible esqueleto de directorio (por defecto es `/etc/skel`), cambiando la localización de entrada al dispositivo especial nulo.

`lfs`
Este es el nombre real del usuario y grupo creados.

Para ingresar como *lfs* (en vez de cambiar al usuario *lfs* cuando se está como *root*, que no precisa que el usuario *lfs* tenga una contraseña), asígnale una contraseña a *lfs*:

```
passwd lfs
```

Concede a *lfs* acceso completo a `$LFS/tools` dándole la propiedad del directorio:

```
chown lfs $LFS/tools
```

Si creaste un directorio de trabajo como te sugerimos, haz que el usuario *lfs* sea también el propietario de este directorio:

```
chown lfs $LFS/sources
```

A continuación, entra como usuario *lfs*. Esto se puede hacer mediante una consola virtual, con un administrador de sesión gráfico o con el siguiente comando de sustitución de usuario:

```
su - lfs
```

El "`-`" le indica a **su** que inicie un intérprete de comandos de ingreso, en lugar de uno de no ingreso. La diferencia entre estos dos tipos de intérpretes de comandos se encuentra detallada en las páginas de manual e info de Bash.

4.4. Configuración del entorno

Establece un buen entorno de trabajo mediante la creación de dos nuevos ficheros de inicio para el intérprete de comandos **bash**. Estando en el sistema como usuario *lfs*, ejecuta los siguientes comandos para crear un `.bash_profile` nuevo:

```
cat > ~/.bash_profile << "EOF"
exec env -i HOME=$HOME TERM=$TERM PS1='\u:\w\$ ' /bin/bash
EOF
```

Cuando entras como usuario *lfs* el intérprete de comandos inicial es un intérprete *de ingreso* que lee el `/etc/profile` de tu anfitrión (que posiblemente contenga algunos ajustes de variables de entorno) y luego lee `.bash_profile`. El comando **exec env -i ... /bin/bash** del fichero `.bash_profile` sustituye el intérprete de comandos en ejecución por uno nuevo con un entorno completamente vacío, excepto por las variables `HOME`, `TERM` y `PS1`. Esto asegura que en el entorno de construcción no aparezcan variables de entorno indeseadas o dañinas procedentes del sistema anfitrión. La técnica aquí usada consigue el objetivo de asegurar un entorno limpio.

La nueva instancia del intérprete comandos es un intérprete de *no ingreso* que no lee los ficheros `/etc/profile` o `.bash_profile`, pero en su lugar lee el fichero `.bashrc`. Crea ahora el fichero `.bashrc`:

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL PATH
EOF
```

El comando **set +h** desactiva la función de tablas de dispersión (hash) de **bash**. Normalmente, esta función es muy útil: **bash** usa una tabla de dispersión para recordar la ruta completa de los ejecutables, evitando búsquedas reiteradas en el `PATH` para encontrar el mismo binario. Sin embargo, las nuevas herramientas deberían utilizarse a medida que son instaladas. Al desactivar esta característica, el intérprete de comandos siempre buscará en el `PATH` cuando deba ejecutarse un programa. Por tanto, el intérprete de comandos encontrará las herramientas recién compiladas en `$LFS/tools` tan pronto como estén disponibles, sin recordar una anterior versión del mismo programa en una ubicación diferente.

Establecer la máscara de creación de ficheros (umask) a 022 asegura que los ficheros y directorios de nueva creación sólo pueden ser escritos por su propietario, pero son legibles y ejecutables por cualquiera (asumiendo que los modos por defecto son usados por la llamada open(2) del sistema, los nuevos ficheros tendrán permisos 644 y los directorios 755).

La variable `LFS` debe establecerse al punto de montaje elegido.

La variable `LC_ALL` controla la localización de ciertos programas, haciendo que sus mensajes sigan las convenciones para un determinado país. Si el sistema anfitrión utiliza una versión de Glibc anterior a la 2.2.4, tener `LC_ALL` establecida a algo diferente a “POSIX” o “C” (durante el siguiente capítulo) puede causar problemas si sales del entorno chroot e intentas regresar más tarde. Establecer `LC_ALL` a “POSIX” o “C” (ambos son equivalentes) asegura que todo funcionará como se espera dentro del entorno chroot.

Al añadir `/tools/bin` al principio del `PATH`, todos los programas instalados en el Capítulo 5 son inmediatamente detectados por el intérprete de comandos tras su instalación. Esto, combinado con la desactivación de las tablas de dispersión, limita el riesgo de utilizar los antiguos programas del anfitrión cuando ya no son necesarios.

Finalmente, para tener el entorno preparado por completo para construir las herramientas temporales, carga el perfil de usuario recién creado:

```
source ~/.bash_profile
```

4.5. Sobre los SBUs

Bastante gente desea saber de antemano cuanto tiempo, aproximadamente, le llevará compilar e instalar cada paquete. Pero Linux From Scratch puede construirse sobre muchos sistemas diferentes, siendo imposible dar tiempos reales y precisos. El paquete más grande (Glibc) tardará unos 20 minutos en un sistema rápido, ¡pero puede tardar hasta tres días en uno lento! Así que en vez de proporcionar tiempos reales se usará la unidad de medida Static Build Unit (SBU, Unidad de Construcción Estática).

Funciona de la siguiente forma. El primer paquete que se compila en este libro es, en el Capítulo 5, Binutils enlazado estáticamente. El tiempo que tarde en compilar este paquete es lo que llamamos Unidad de Construcción Estática o SBU. Todos los demás tiempos de compilación se expresarán con relación a este tiempo.

Por ejemplo, considera un paquete cuyo tiempo de compilación es de 4,5 SBUs. Esto significa que si un sistema tarda en compilar e instalar el Binutils estático 10 minutos, tardará *aproximadamente* 45 minutos en construir dicho paquete. Por suerte, bastantes de los tiempos de construcción son mucho más cortos que el de Binutils.

Ten en cuenta que si el compilador de tu anfitrión está basado en GCC-2.x, los SBUs listados pueden ser algo bajos. Esto es debido a que el SBU está basado en el primer paquete, compilado con el antiguo GCC, mientras que el resto del sistema se compila con el nuevo GCC-3.4.1 (que se sabe que es aproximadamente un 30% más lento). Los SBU tampoco son exactos para máquinas basadas en Multi-Procesadores Simétricos (SMP).

Para ver los tiempos reales de un cierto número de máquinas concretas, recomendamos visitar *http://www.linuxfromscratch.org/~bdubbs/*.

En general, los SBU no son muy exactos debido a que dependen de muchos factores, no sólo la versión de GCC. Se han suministrado aquí para mostrar una aproximación de cuanto podría tardar la instalación de un paquete, pero los números pueden variar en docenas de minutos en algunos casos.

4.6. Sobre los bancos de pruebas

Muchos paquetes proporcionan un banco de pruebas. Ejecutar el banco de pruebas para un paquete recién construido es una buena idea, pues puede proporcionar una “verificación de calidad” indicando que todo se ha compilado correctamente. Un banco de pruebas que supere sus comprobaciones normalmente confirma que el paquete está funcionando tal y como el desarrollador espera. Pero esto, sin embargo, no garantiza que el paquete está totalmente libre de errores.

Algunos bancos de pruebas son más importantes que otros. Por ejemplo, los bancos de pruebas de los paquetes de las herramientas principales (GCC, Binutils y Glibc) son de la mayor importancia debido a su papel central en el correcto funcionamiento del sistema. Los bancos de pruebas para GCC y Glibc pueden tardar bastante tiempo en completarse, sobre todo en hardware lento, pero son muy recomendables.

Nota

La experiencia ha mostrado que se gana poco ejecutando los bancos de pruebas en el Capítulo 5. No se puede escapar del hecho de que el sistema anfitrión siempre ejerce cierta influencia sobre las pruebas en dicho capítulo, causando con frecuencia fallos inexplicables. Debido a que las herramientas construidas en el Capítulo 5 son temporales y descartables, no recomendamos el lector medio ejecutar los bancos de pruebas en el Capítulo 5. Las instrucciones para ejecutarlos se suministran en benficio de los verificadores y desarrolladores, pero son estrictamente opcionales.

Un problema común al ejecutar los bancos de pruebas de Binutils y GCC es quedarse sin seudo-terminales (PTYs). El síntoma es un número inusualmente alto de pruebas fallidas. Esto puede suceder por diferentes razones, pero lo más probable es que el sistema anfitrión no tenga el sistema de ficheros `devpts` configurado correctamente. En el Capítulo 5 se tratará este tema con mayor detalle.

En ocasiones los bancos de pruebas de los paquetes muestran falsos fallos. Consulta el Wiki de LFS en *http://wiki.linuxfromscratch.org/* para consultar si estos fallos son normales. Este sitio es válido para todas las pruebas que aparecen en el libro.

Capítulo 5. Construir un sistema temporal

5.1. Introducción

Este capítulo muestra cómo compilar e instalar un sistema Linux mínimo. Este sistema contendrá sólo las herramientas necesarias para poder iniciar la construcción del sistema LFS definitivo en el Capítulo 6, permitiendo un entorno de trabajo algo más amigable para el usuario que el que un entorno mínimo ofrecería.

La construcción de este sistema minimalista se hará en dos etapas. La primera es construir un conjunto de herramientas independiente del sistema anfitrión (compilador, ensamblador, enlazador, librerías y unas pocas herramientas útiles). La segunda etapa utiliza estas herramientas para construir el resto de herramientas esenciales.

Los ficheros compilados en este capítulo se instalarán bajo el directorio `$LFS/tools` para mantenerlos separados de los ficheros que se instalen en el siguiente capítulo y de los directorios de producción de tu anfitrión. Puesto que los paquetes compilados aquí son puramente temporales, no queremos que estos ficheros contaminen el futuro sistema LFS.

Antes de ejecutar las instrucciones de construcción para un paquete, debes desempaquetarlo como usuario *lfs* y hacer un **`cd`** para entrar al directorio creado. Las instrucciones de construcción asumen que estás usando el intérprete de comandos **bash**.

Varios de los paquetes deben parchearse antes de compilarlos, pero sólo cuando el parche es necesario para solucionar un problema. Con frecuencia el parche es necesario tanto en éste como en el siguiente capítulo, pero a veces sólo es necesario en uno de ellos. Por lo tanto, no te preocupes si parece que faltan las instrucciones para uno de los parches descargados. Igualmente, cuando se aplique un parche ocasionalmente verás un mensaje de aviso sobre *offset* o *fuzz*. No debes preocuparte por estos avisos, pues el parche se aplicará correctamente.

Durante la compilación de muchos paquetes verás aparecer en pantalla diversos avisos (warnings). Esto es normal y puedes ignorarlos con tranquilidad. No son más que eso, avisos; la mayoría debidos a un uso inapropiado, pero no inválido, de la sintaxis de C o C++. Se debe a que los estándares de C cambian con frecuencia y algunos paquetes todavía usan un estándar antiguo. Esto no es un problema, pero hace que se muestre el aviso.

Tras instalar cada paquete debes borrar sus directorios de fuentes y de construcción, excepto si se indica lo contrario. Borrar las fuentes ahorra espacio, pero también previene de fallos de configuración cuando el mismo paquete se reinstale más adelante. Sólo necesitarás guardar los directorios de fuentes y construcción de tres paquetes durante un tiempo, para que su contenido pueda ser usado por posteriores comandos. Estate atento a dichos recordatorios.

Comprueba de nuevo que la variable de entorno `LFS` está correctamente establecida:

```
echo $LFS
```

Asegúrate de que la salida muestra la ruta al punto de montaje de tu partición LFS, que es `/mnt/lfs` si seguiste nuestro ejemplo.

5.2. Requisitos del sistema anfitrión

El anfitrión debe estar ejecutando, al menos, un núcleo 2.6.2 compilado con GCC-3.0 o superior. Hay dos razones principales para estos altos requisitos. Primero, hacemos uso de la Librería Nativa de Hilos POSIX (Native Posix Threading Library, NPTL) cuyo banco de pruebas fallará si el núcleo del anfitrión no ha sido compilado con GCC-3.0 o superior. En segundo lugar, se necesita un núcleo 2.6.2 o posterior para utilizar Udev. Udev crea dispositivos dinámicamente a partir de lo que lee en el sistema de ficheros `sysfs`. Sin embargo, sólo muy recientemente se ha implementado el soporte para este sistema de ficheros en muchos de los controladores del núcleo. Debemos asegurarnos de que todos los dispositivos críticos del sistema son correctamente creados.

Para comprobar que el núcleo de tu anfitrión cumple los requisitos arriba mencionados, puedes ejecutar el siguiente comando:

```
cat /proc/version
```

Esto generará una salida similar a:

```
Linux version 2.6.2 (user@host) (gcc version 3.4.0) #1
    Tue Apr 20 21:22:18 GMT 2004
```

Si el resultado del comando anterior muestra que el núcleo de tu anfitrión no fue compilado usando GCC-3.0 (o superior), necesitarás compilar uno tu mismo y reiniciar tu anfitrión para usar el núcleo recién compilado. Las instrucciones para compilar el núcleo y configurar el gestor de arranque (asumiendo que tu anfitrión utiliza GRUB) se muestran en el Capítulo 8.

5.3. Notas técnicas sobre las herramientas

Esta sección explica algunos de los razonamientos y detalles técnicos que hay detrás del sistema de construcción. No es esencial que entiendas todo esto inmediatamente. La mayor parte tendrá sentido cuando hayas hecho una construcción real. Puedes consultar esta sección en cualquier momento durante la construcción.

El principal objetivo del Capítulo 5 es proporcionar un entorno temporal al que podamos entrar con chroot y a partir del cual podamos generar una construcción limpia y libre de problemas del sistema LFS en el Capítulo 6. Por el camino intentaremos independizarnos todo lo posible del sistema anfitrión, y para eso construimos unas herramientas principales autocontenidas y autohospedadas. Debería tenerse en cuenta que el proceso de construcción ha sido diseñado de forma que se minimice el riesgo para los nuevos lectores y, al mismo tiempo, se proporcione el máximo valor educacional. En otras palabras, se pueden usar técnicas más avanzadas para construir el sistema.

Importante

Antes de continuar, deberías informarte del nombre de tu plataforma de trabajo, conocido con frecuencia como *target triplet* (triplete del objetivo). Para muchos el "target triplet" posiblemente sea *i686-pc-linux-gnu*. Una forma simple de determinar tu "target triplet" es ejecutar el guión **config.guess** que se incluye con las fuentes de muchos paquetes. Desempaqueta las fuentes de Binutils, ejecuta el guión **`./config.guess`** y anota el resultado.

Igualmente necesitarás saber el nombre del enlazador dinámico de tu plataforma, también conocido como cargador dinámico (no debe confundirse con el enlazador estándar **ld**, que es parte de Binutils). El enlazador dinámico suministrado por Glibc encuentra y carga las librerías compartidas necesarias para un programa, prepara el programa y lo ejecuta. Usualmente el nombre del enlazador dinámico es `ld-linux.so.2`. En plataformas menos conocidas puede ser `ld.so.1` y en las nuevas plataformas de 64 bits puede que incluso sea algo totalmente diferente. El nombre del enlazador dinámico de tu plataforma puede determinarse mirando en el directorio `/lib` de tu sistema anfitrión. Un modo seguro es inspeccionar un binario cualquiera de tu sistema anfitrión ejecutando: **`readelf -l <nombre del binario> | grep interpreter`** y anotar la salida. La referencia autorizada que cubre todas las plataformas está en el fichero `shlib-versions` en la raíz del árbol de las fuentes de Glibc.

Algunas claves técnicas sobre cómo funciona el método de construcción del Capítulo 5:

- Similar en principio a la compilación cruzada, donde las herramientas instaladas dentro del mismo prefijo trabajan en cooperación y utilizan una pequeña "magia" de GNU.
- Cuidada manipulación de la ruta de búsqueda de librerías del enlazador estándar para asegurar que los programas se enlazan sólo contra las librerías que elegimos.
- Cuidada manipulación del fichero `specs` de **gcc** para indicarle al compilador cuál es el enlazador dinámico a usar.

Se instala primero Binutils debido a que, tanto en GCC como en Glibc, la ejecución de **./configure** realiza varias pruebas sobre el ensamblador y el enlazador para determinar qué características del software deben activarse o desactivarse. Esto es más importante de lo que uno podría pensar. Un GCC o una Glibc incorrectamente configurados puede provocar unas herramientas sutilmente rotas cuyo impacto podría no notarse hasta casi finalizada la construcción de una distribución completa. Por suerte, un fallo en el banco de pruebas normalmente nos avisará antes de perder demasiado tiempo.

Binutils instala tanto su ensamblador como su enlazador en dos ubicaciones, `/tools/bin` y `/tools/$TARGET_TRIPLET/bin`. Las herramientas de una ubicación son enlaces duros a la otra. Un aspecto importante del enlazador es su orden de búsqueda de librerías. Puede obtenerse información detallada de **ld** pasándole la opción *`--verbose`*. Por ejemplo, un **ld --verbose | grep SEARCH** mostrará las rutas de búsqueda actuales y su orden. Puedes ver qué ficheros son realmente enlazados por **ld** compilando un programa simulado y pasándole la opción *`--verbose`*. Por ejemplo, **`gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded`** te mostrará todos los ficheros abiertos con éxito durante el enlazado.

El siguiente paquete instalado es GCC y durante su fase **./configure** verás, por ejemplo:

```
checking what assembler to use...
    /tools/i686-pc-linux-gnu/bin/as
checking what linker to use...
    /tools/i686-pc-linux-gnu/bin/ld

comprobando qué ensamblador usar...
    /tools/i686-pc-linux-gnu/bin/as
comprobando qué enlazador usar...
    /tools/i686-pc-linux-gnu/bin/ld
```

Esto es importante por la razón mencionada antes. También demuestra que el guión configure de GCC no explora los directorios del PATH para encontrar las herramientas a usar. Sin embargo, durante la operación real del propio **gcc**, no se utilizan necesariamente las mismas rutas de búsqueda. Para saber cuál es el enlazador estándar que utilizará **gcc**, ejecuta: **`gcc -print-prog-name=ld`**.

Puedes obtener información detallada a partir de **gcc** pasándole la opción *`-v`* mientras compilas un programa simulado. Por ejemplo: **`gcc -v dummy.c`** te mostrará los detalles sobre las fases de preprocesamiento, compilación y ensamblado, incluidas las rutas de búsqueda de **gcc** y su orden.

A continuación se instala Glibc. Las consideraciones más importantes para la construcción de Glibc son el compilador, las herramientas de binarios y las cabeceras del núcleo. Normalmente el compilador no es problema, pues Glibc siempre utilizará el **gcc** que se encuentre en un directorio del `PATH`. Las herramientas de binarios y las cabeceras del núcleo pueden ser algo mas problemáticas, así que no nos arriesgaremos y haremos uso de las opciones disponibles de configure para forzar las opciones correctas. Después de ejecutar **./configure** puedes revisar el contenido del fichero `config.make` en el directorio `glibc-build` para ver todos los detalles importantes. Encontrarás algunas cosas interesantes, como el uso de *`CC="gcc -B/tools/bin/"`* para controlar qué herramientas de binarios son usadas, y también el uso de las opciones *`-nostdinc`* y *`-isystem`* para controlar la ruta de búsqueda de cabeceras del compilador. Estos detalles ayudan a resaltar un aspecto importante del paquete Glibc: es muy autosuficiente en cuanto a su maquinaria de construcción y generalmente no se apoya en las opciones por defecto de las herramientas.

Después de la instalación de Glibc, haremos algunos ajustes para asegurar que la búsqueda y el enlazado tengan lugar solamente dentro de nuestro directorio `/tools`. Instalaremos un **ld** ajustado, que tiene limitada su ruta de búsqueda interna a `/tools/lib`. Entonces retocaremos el fichero specs de **gcc** para que apunte a nuestro nuevo enlazador dinámico en `/tools/lib`. Este último paso es vital para el proceso completo. Como se mencionó antes, dentro de cada ejecutable compartido ELF se fija la ruta a un enlazador dinámico. Puedes verificar esto mediante: **`readelf -l <nombre del binario> | grep interpreter`**. Retocando el ficheros specs de **gcc** estaremos seguros de que todo binario compilado desde aquí hasta el final de este capítulo usará nuestro nuevo enlazador dinámico en `/tools/lib`.

La necesidad de utilizar el nuevo enlazador dinámico es también la razón por la que aplicamos el parche Specs en la segunda fase de GCC. De no hacer esto los propios programas de GCC incluirían dentro suyo el nombre del enlazador dinámico del directorio `/lib` del sistema anfitrión, lo que arruinaría nuestro objetivo de librarnos del anfitrión.

Durante la segunda fase de Binutils podremos usar la opción *`--with-lib-path`* de configure para controlar la ruta de búsqueda de librerías de **ld**. A partir de este punto el corazón de las herramientas está autocontenido y autohospedado. El resto de los paquetes del Capítulo 5 se construirán todos contra la nueva Glibc en `/tools`.

Tras entrar en el entorno chroot en el Capítulo 6, el primer gran paquete a instalar es Glibc, debido a su naturaleza autosuficiente. Una vez que esta Glibc se instale dentro de `/usr`, haremos un rápido cambio en las opciones por defecto de las herramientas y entonces procederemos a la construcción real del sistema LFS.

5.3.1. Notas sobre el enlazado estático

Muchos programas han de realizar, dentro de sus tareas específicas, muchas operaciones comunes y, en ocasiones, triviales. Esto incluye reservar memoria, explorar directorios, leer y escribir ficheros, manejar cadenas, emparejar patrones, realizar cálculos y muchas otras tareas. En vez de obligar a cada programa a reinventar la rueda, el sistema GNU facilita todas estas funciones básicas dentro de librerías listas para usar. La principal librería en cualquier sistema Linux es *Glibc*.

Hay dos formas de enlazar las funciones de una librería en un programa que las utilice: estática o dinámicamente. Cuando un programa se enlaza estáticamente el código de las funciones usadas se incluye dentro del ejecutable, resultando un programa algo abultado. Cuando un programa se enlaza dinámicamente lo que se incluye es una referencia al enlazador dinámico, el nombre de la librería y el nombre de la función, resultando un ejecutable mucho más pequeño. Un tercer método es utilizar la interfaz de programación del enlazador dinámico (mira la página de manual de *dlopen* para obtener más información).

En Linux se usa por defecto enlazado dinámico y tiene tres ventajas fundamentales sobre el enlazado estático. Primero, sólo necesitas en tu disco duro una copia del código de la librería ejecutable, en vez de tener muchas copias del mismo código dentro de un montón de programas, ahorrando así espacio en disco. La segunda es que cuando varios programas usan la misma función de una librería al mismo tiempo sólo hace falta cargar una copia del código de la función, ahorrando espacio en memoria. Por último, cuando se corrige un error o se mejora una función de una librería sólo necesitas recompilar esta librería, en vez de tener que recompilar todos los programas que hacen uso de esta función mejorada.

Si el enlace dinámico tiene varias ventajas, ¿por qué enlazamos estáticamente los dos primeros paquetes en este capítulo? La razón es triple: histórica, educacional y técnica. Histórica porque las anteriores versiones de LFS enlazaban estáticamente cada paquete en este capítulo. Educacional porque conocer la diferencia es útil. Técnica porque ganamos un elemento de independencia sobre el anfitrión al hacerlo. Por ejemplo, estos programas pueden usarse independientemente del sistema anfitrión. Sin embargo, hay que mencionar que se puede conseguir una construcción correcta del sistema LFS al completo cuando los dos primeros paquetes se construyen dinámicamente.

5.4. Binutils-2.15.91.0.2 - Fase 1

El paquete Binutils contiene un enlazador, un ensamblador y otras utilidades para trabajar con ficheros objeto.

Tiempo estimado de construcción: 1.0 SBU
Espacio requerido en disco: 194 MB

La instalación de Binutils depende de: Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed y Texinfo

5.4.1. Instalación de Binutils

Es importante que Binutils sea el primer paquete que compiles, pues tanto Glibc como GCC llevan a cabo varias comprobaciones sobre el enlazador y el ensamblador disponibles para determinar qué características activar.

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como CFLAGS o CXXFLAGS, desactívala cuando construyas Binutils.

La documentación de Binutils recomienda construirlo en un directorio dedicado, fuera del árbol de las fuentes:

```
mkdir ../binutils-build
cd ../binutils-build
```

Nota

Si quieres que los valores de los SBUs mostrados en el resto del libro sean de utilidad, mide el tiempo que se tarda en construir este paquete desde la compilación hasta la primera instalación. Para ello, envuelve los comandos dentro de un comando **time** de esta forma: **time { ./configure ... && ... && ... && make install; }**.

Prepara Binutils para su compilación:

```
../binutils-2.15.91.0.2/configure --prefix=/tools \
    --disable-nls
```

Significado de las opciones de configure:

--prefix=/tools
Esto le indica al guión configure que los programas de Binutils se instalarán en el directorio /tools.

--disable-nls
Esta opción desactiva la internacionalización. No es necesaria para nuestros programas estáticos y NLS suele causar problemas con el enlazado estático.

Compila el paquete:

```
make configure-host
make LDFLAGS="-all-static"
```

Significado de los parámetros de make:

`configure-host`
: Esto fuerza que todos los subdirectorios se configuren inmediatamente. Una construcción enlazada estáticamente fallará sin esto. Por lo tanto, usamos esta opción para evitar el problema.

`LDFLAGS="-all-static"`
: Esto le indica al enlazador que todos los programas de Binutils deben enlazarse estáticamente. Sin embargo, y estrictamente hablando, *`"-all-static"`* se le pasa al programa **libtool**, el cual luego le pasa *`"-static"`* al enlazador.

La compilación se ha completado. Normalmente deberíamos ejecutar ahora el banco de pruebas, pero en esta temprana fase el entorno de trabajo para los bancos de pruebas (Tcl, Expect y DejaGnu) todavía no está en su sitio. Los beneficios de ejecutar las pruebas ahora son mínimos, pues los programas de esta primera fase pronto serán sustituidos por los de la segunda.

Instala el paquete:

```
make install
```

Prepara el enlazador para la posterior fase de “ajuste”:

```
make -C ld clean
make -C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib
```

Significado de los parámetros de make:

`-C ld clean`
: Esto le indica al programa make que elimine todos los ficheros compilados que haya en el subdirectorio `ld`.

`-C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib`
: Esta opción vuelve a construir todo dentro del subdirectorio `ld`. Especificar la variable `LIB_PATH` del Makefile en la línea de comandos nos permite obviar su valor por defecto y apuntar a nuestro directorio de herramientas temporales. El valor de esta variable especifica la ruta de búsqueda de librerías por defecto del enlazador. Estos preparativos se utilizan más tarde en este capítulo.

Aviso

No borres los directorios de fuentes y de construcción de Binutils. Los necesitarás un poco más adelante en este capítulo en el estado en que se encuentran ahora.

Los detalles sobre este paquete se encuentran en la Sección 6.13.2, “Contenido de Binutils”.

5.5. GCC-3.4.1 - Fase 1

El paquete GCC contiene la colección de compiladores GNU, que incluye los compiladores C y C++.

Tiempo estimado de construcción: 4.4 SBU
Espacio requerido en disco: 300 MB

La instalación de GCC depende de: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed y Texinfo

5.5.1. Instalación de GCC

Desempaqueta sólo el paquete gcc-core, pues por el momento no vamos a necesitar ni el compilador de C++ ni el banco de pruebas.

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como CFLAGS o CXXFLAGS, desactívala cuando construyas GCC.

La documentación de GCC recomienda construirlo en un directorio dedicado, fuera del árbol de las fuentes:

```
mkdir ../gcc-build
cd ../gcc-build
```

Prepara GCC para su compilación:

```
../gcc-3.4.1/configure --prefix=/tools \
    --libexecdir=/tools/lib --with-local-prefix=/tools \
    --disable-nls --enable-shared --enable-languages=c
```

Significado de las opciones de configure:

--with-local-prefix=/tools
Esta opción es para eliminar /usr/local/include de las rutas de búsqueda por defecto de **gcc**. Esto no es esencial, sin embargo ayuda a minimizar la influencia del sistema anfitrión.

--enable-shared
Esta opción no parece intuitiva al principio, pero nos permite construir libgcc_s.so.1 y libgcc_eh.a, y tener a libgcc_eh.a disponible nos asegura que el guión configure de Glibc (el siguiente paquete por compilar) produzca los resultados apropiados. Ten en cuenta que los binarios de GCC se compilarán estáticamente de todas formas, ya que esto lo controla el valor *-static* que asumirá la variable BOOT_LDFLAGS en el siguiente paso.

--enable-languages=c
Esta opción nos asegura que sólo se construya el compilador de C. Es necesaria únicamente en caso de que hayas descargado y desempaquetado el paquete completo de GCC.

Compila el paquete:

```
make BOOT_LDFLAGS="-static" bootstrap
```

Significado de los parámetros de make:

`BOOT_LDFLAGS="-static"`
Esto le indica a GCC que sus programas se enlacen estáticamente.

`bootstrap`
Este objetivo no sólo compila GCC, sino que lo compila varias veces. Usa los programas compilados la primera vez para compilarse a sí mismo una segunda vez y luego una tercera. Después compara la segunda compilación con la tercera para asegurarse que puede reproducirse a sí mismo sin errores. Esto también implica que se ha compilado correctamente.

La compilación se ha completado. En este punto normalmente ejecutaríamos el banco de pruebas, pero, como se mencionó antes, el entorno de trabajo para los bancos de pruebas no se encuentra todavía en su lugar. Los beneficios de ejecutar ahora los bancos de pruebas son mínimos, pues los programas de esta primera fase pronto serán sustituidos.

Instala el paquete:

```
make install
```

Como toque final, crea un enlace simbólico. Muchos programas y guiones ejecutan **cc** en vez de **gcc**. Esto es una forma de hacer que los programas sean genéricos y por tanto utilizables en toda clase de sistemas Unix. No todos tienen instalado el compilador de C de GNU. Ejecutar **cc** deja al administrador del sistema libre de decidir qué compilador de C instalar, mientras haya un enlace simbólico que apunte a él.

```
ln -s gcc /tools/bin/cc
```

Los detalles sobre este paquete se encuentran en la Sección 6.14.2, “Contenido de GCC”.

5.6. Linux-Libc-Headers-2.6.8.1

El paquete Linux-Libc-Headers contiene las cabeceras “saneadas” del núcleo.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 22 MB

La instalación de Linux-Libc-Headers depende de: Coreutils

5.6.1. Instalación de Linux-Libc-Headers

Durante años ha sido una práctica común utilizar las llamadas cabeceras “crudas” del núcleo (extraidas de un paquete del núcleo) en `/usr/include`, pero en el transcurso de los últimos años los desarrolladores del núcleo han tomado la firme postura de que esto no debe suceder. Así nació el proyecto Linux-Libc-Headers, destinado a mantener una versión estable de la API de las cabeceras Linux.

Instala los ficheros de cabecera:

```
cp -R include/asm-i386 /tools/include/asm
cp -R include/linux /tools/include
```

Si tu arquitectura no es i386 (o compatible), modifica adecuadamente el primer comando.

Los detalles sobre este paquete se encuentran en la Sección 6.9.2, “Contenido de Linux-Libc-Headers”.

5.7. Cabeceras de Linux-2.6.8.1

El paquete del núcleo Linux contiene las fuentes del núcleo y los ficheros de cabecera utilizados por Glibc.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 186 MB

La instalación de las cabeceras de Linux depende de: Coreutils y Make

5.7.1. Instalación de las cabeceras del núcleo

Como ciertos paquetes necesitan usar los ficheros de cabecera (headers) del núcleo, vamos a desempaquetar el archivo del núcleo ahora, configurarlo, y copiar los ficheros necesarios a un lugar donde **gcc** pueda encontrarlos.

Prepara la instalación de las cabeceras:

```
make mrproper
```

Esto asegurará que el árbol del núcleo está absolutamente limpio. El equipo de desarrollo recomienda usar este comando antes de *cada* compilación del núcleo, y en realidad no debes confiar en que el árbol de las fuentes esté limpio tras desempaquetarlo.

Crea el fichero `include/linux/version.h`:

```
make include/linux/version.h
```

Crea el enlace simbólico `include/asm` específico de la plataforma:

```
make include/asm
```

Instala los ficheros de cabecera específicos de la plataforma:

```
mkdir /tools/glibc-kernheaders
cp -HR include/asm /tools/glibc-kernheaders
cp -R include/asm-generic /tools/glibc-kernheaders
```

Por último, instala los ficheros de cabecera del núcleo independientes de la plataforma:

```
cp -R include/linux /tools/glibc-kernheaders
```

Los detalles sobre este paquete se encuentran en la Sección 8.3.2, “Contenido de Linux”.

5.8. Glibc-2.3.4-20040701

El paquete Glibc contiene la librería C principal. Esta librería proporciona todas las rutinas básicas para la ubicación de memoria, búsqueda de directorios, abrir y cerrar ficheros, leerlos y escribirlos, manejo de cadenas, coincidencia de patrones, aritmética, etc...

Tiempo estimado de construcción: 11.8 SBU
Espacio requerido en disco: 800 MB

La instalación de Glibc depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed y Texinfo

5.8.1. Instalación de Glibc

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como CFLAGS o CXXFLAGS, desactívala cuando construyas Glibc.

Hay que resaltar que compilar Glibc de forma diferente a como el libro sugiere pone la estabilidad de tu sistema en grave riesgo.

La documentación de Glibc recomienda construirlo fuera del árbol de las fuentes, en un directorio de construcción dedicado:

```
mkdir ../glibc-build
cd ../glibc-build
```

Prepara Glibc para su compilación:

```
../glibc-2.3.4-20040701/configure --prefix=/tools \
    --disable-profile --enable-add-ons=nptl --with-tls \
    --with-__thread --enable-kernel=2.6.0 \
    --with-binutils=/tools/bin --without-gd --without-cvs \
    --with-headers=/tools/glibc-kernheaders
```

Significado de las opciones de configure:

--disable-profile
Esto construye las librerías sin información de perfiles. Omite esta opción si planeas usar perfiles en las herramientas temporales.

--enable-add-ons=nptl
Esto le indica a Glibc que utilice el añadido NPTL como su librería de hilos.

--with-tls
Esto le indica a Glibc que incluya el soporte para TLS (almacenamiento de hilos locales). Esto es necesario para que funcione NPTL.

--with-__thread
Esta opción le indica a Glibc que incluya el soporte para hilos. Es necesario para conseguir que TLS se compile correctamente.

--enable-kernel=2.6.0
Esto le indica a Glibc que compile la librería con soporte para núcleos Linux 2.6.x.

--with-binutils=/tools/bin
Aunque no es necesario, esta opción nos asegura que no haya equívocos sobre qué programas de Binutils se utilizarán durante la construcción de Glibc.

--without-gd
Esto evita la construcción del programa **memusagestat**, el cual insiste en enlazarse contra las librerías del sistema anfitrión (libgd, libpng, libz y demás).

--without-cvs
Esto se indica para evitar que los Makefiles intenten hacer automáticamente un **cvs checkout** cuando se utiliza una imagen CVS. Aunque no es realmente necesario, es recomendable porque silencia un molesto pero inofensivo aviso sobre la ausencia del programa **autoconf**.

--with-headers=/tools/glibc-kernheaders
Esto le indica a Glibc que se compile contra las cabeceras "crudas" del núcleo, para que así sepa exactamente las características que tiene el núcleo y pueda optimizarse de acuerdo a esto.

Durante esta fase puede que veas el siguiente mensaje de aviso:

```
configure: WARNING:
*** These auxiliary programs are missing or
*** incompatible versions: msgfmt
*** some features will be disabled.
*** Check the INSTALL file for required versions.

configure: AVISO:
*** Versión incompatible o ausente de estos
*** programas auxiliares: msgfmt
*** algunas características serán desactivadas.
*** Comprueba en el fichero INSTALL las versiones requeridas.
```

Normalmente, la ausencia o incompatibilidad del programa **msgfmt** es inofensiva, pero se cree que en ocasiones puede causar problemas al ejecutar el banco de pruebas. El programa **msgfmt** es parte del paquete Gettext y debería proporcionarlo el sistema anfitrión. Si **msgfmt** está presente pero es incompatible, actualiza el paquete Gettext del sistema anfitrión o continúa sin él y observa si los bancos de pruebas se ejecutan sin problemas.

Compila el paquete:

```
make
```

La compilación está completa. Como se mencionó antes, no es obligatorio ejecutar los bancos de pruebas de las herramientas temporales en este capítulo. Si de todas formas deseas ejecutar el banco de pruebas de Glibc, hazlo con el siguiente comando:

```
make check
```

Consulta en la Sección 6.11, "Glibc-2.3.4-20040701", la explicación de los fallos de las pruebas que tienen una particular importancia.

En este capítulo algunas pruebas pueden verse afectadas adversamente por las herramientas existentes o el entorno del sistema anfitrión. En resumen, no te preocupes demasiado si ves fallos en el banco de pruebas de Glibc en este capítulo. La Glibc del Capítulo 6 es la que acabaremos usando al final, por lo que es la que necesitamos que pase la mayoría de las pruebas (incluso en el Capítulo 6 es posible que todavía ocurran algunos fallos, la prueba *math* por ejemplo).

Cuando aparezca un fallo, anótalo y continua ejecutando de nuevo **make check**. El banco de pruebas debería continuar a partir de donde se quedó. Puedes evitar esta secuencia de inicio-parada ejecutando **make -k check**. Si utilizas esta opción, asegúrate de registrar la salida para que más tarde puedas revisar el fichero de registro en búsqueda de errores.

La fase de instalación de Glibc mostrará un aviso inofensivo sobre la ausencia del fichero `/tools/etc/ld.so.conf`. Evita este confuso aviso con:

```
mkdir /tools/etc
touch /tools/etc/ld.so.conf
```

Instala el paquete:

```
make install
```

Diferentes países y culturas tienen diferentes convenciones sobre cómo comunicarse. Estas convenciones van desde las más simples, como el formato para representar fechas y horas, a las más complejas, como el lenguaje hablado. La “internacionalización” de los programas GNU funciona mediante el uso de *locales*.

Nota

Si no estás ejecutando los bancos de pruebas en este capítulo, como recomendamos, no hay razón para instalar ahora las locales. Las instalaremos en el siguiente capítulo.

Si de todas formas quieres instalar las locales de Glibc, hazlo con el siguiente comando:

```
make localedata/install-locales
```

Para ahorrar tiempo, una alternativa al comando anterior (que genera e instala todas las locale que Glibc conoce) es instalar solamente aquellas locales que necesites o desees. Esto puede hacerse usando el comando **localedef**. Puedes encontrar más información sobre esto en el fichero `INSTALL` de las fuentes de Glibc. Sin embargo, hay un número de locales que son esenciales para que las comprobaciones de paquetes posteriores se realicen. En particular, la prueba de *libstdc++* en GCC. Las siguientes instrucciones, en vez del objetivo anterior *`install-locales`*, instalarán el conjunto mínimo de locales necesario para que las pruebas se ejecuten correctamente:

```
mkdir -p /tools/lib/locale
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i en_HK -f ISO-8859-1 en_HK
localedef -i en_PH -f ISO-8859-1 en_PH
localedef -i en_US -f ISO-8859-1 en_US
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

Los detalles sobre este paquete se encuentran en la Sección 6.11.4, “Contenido de Glibc”.

5.9. Ajustar las herramientas

Ahora que se han instalado las librerías de C temporales, todas las herramientas que compilemos en el resto de este capítulo deberían enlazarse contra ellas. Para conseguirlo, deben ajustarse el enlazador y el fichero specs del compilador.

El enlazador, que se ajustó al final del primer paso de Binutils, se instala ejecutando el siguiente comando desde el directorio `binutils-build`:

```
make -C ld install
```

Desde ahora todo se enlazará solamente contra las librerías que hay en `/tools/lib`.

Nota

Si por alguna razón olvidaste el aviso sobre conservar los directorios de las fuentes y de construcción del primer paso de Binutils, ignora el comando anterior. El resultado es la pequeña pega de que los siguientes programas de pruebas se enlazarán contra las librerías del anfitrión. Esto no es lo ideal, pero no es un gran problema. La situación se corregirá cuando instalemos un poco más adelante la segunda fase de Binutils.

Ahora que se ha instalado el enlazador ajustado, debes eliminar los directorios de las fuentes y de construcción de Binutils.

Lo siguiente es corregir el fichero de especificaciones de GCC para que apunte al nuevo enlazador dinámico. Un simple comando sed lo hará:

```
SPECFILE=`gcc --print-file specs` &&
sed 's@ /lib/ld-linux.so.2@ /tools/lib/ld-linux.so.2@g' \
    $SPECFILE > tempspecfile &&
mv -f tempspecfile $SPECFILE &&
unset SPECFILE
```

Alternativamente, puedes editar el fichero specs a mano si quieres. Esto se hace reemplazando cada aparición de “/lib/ld-linux.so.2” con “/tools/lib/ld-linux.so.2”.

Asegúrate de revisar visualmente el fichero specs para verificar que se han hecho los cambios deseados.

Importante

Si estás trabajando sobre una plataforma en la que el nombre del enlazador dinámico no es `ld-linux.so.2`, en el anterior comando debes sustituir `ld-linux.so.2` con el nombre del enlazador dinámico de tu plataforma. En caso necesario consulta la Sección 5.3, “Notas técnicas sobre las herramientas”.

Existe la posibilidad de que algunos ficheros de cabecera de nuestro sistema anfitrión se hayan colado dentro del directorio privado de cabeceras de GCC. Esto puede suceder debido al proceso “fixincludes” de GCC que se ejecuta como parte de su proceso de construcción. Explicaremos esto con más detalle dentro de este capítulo. Por ahora, ejecuta este comando para eliminar dicha posibilidad:

```
rm -f /tools/lib/gcc/*/*/include/{pthread.h,bits/sigthread.h}
```

Atención

En este punto es obligatorio parar y asegurarse de que las operaciones básicas (compilación y enlazado) de las nuevas herramientas funcionan como se espera. Para esto vamos a hacer una simple comprobación:

```
echo 'main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /tools'
```

Si todo funciona correctamente, no debe haber errores y la salida del último comando debe ser:

```
[Requesting program interpreter:
    /tools/lib/ld-linux.so.2]

[Intérprete de programa solicitado:
    /tools/lib/ld-linux.so.2]
```

Confirma que `/tools/lib` aparezca como el prefijo de tu enlazador dinámico.

Si no recibes una salida como la mostrada arriba, o no hay salida alguna, algo está seriamente mal. Investiga y revisa tus pasos para encontrar el problema y corregirlo. El problema debe resolverse antes de continuar. Primero, repite la comprobación de sanidad usando **gcc** en vez de **cc**. Si esto funciona significa que falta el enlace simbólico `/tools/bin/cc`. Vuelve a la Sección 5.5, “GCC-3.4.1 - Fase 1” e instala el enlace simbólico. Seguidamente, asegúrate de que tu `PATH` es correcto. Puedes comprobarlo ejecutando **echo $PATH** y verificando que `/tools/bin` está en cabeza de la lista. Si el `PATH` está mal puede significar que no has ingresado como usuario *lfs* o que algo salió mal en la Sección 4.4, “Configuración del entorno”. Otra opción es que algo pudo ir mal en el anterior arreglo del fichero specs. En este caso, repite el arreglo del fichero.

Cuando todo esté bien, borra los ficheros de prueba:

```
rm dummy.c a.out
```

5.10. Tcl-8.4.7

El paquete Tcl contiene el Tool Command Language (Lenguaje para Herramientas de Comandos).

Tiempo estimado de construcción: 0.9 SBU
Espacio requerido en disco: 23 MB

La instalación de Tcl depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make y Sed

5.10.1. Instalación de Tcl

Este paquete y los dos siguientes (Expect y DejaGNU) se instalan con el único propósito de poder ejecutar los bancos de pruebas de GCC y Binutils. Instalar tres paquetes sólo para realizar comprobaciones puede parecer excesivo, pero es muy tranquilizador, si no esencial, saber que las herramientas más importantes funcionan adecuadamente. Aunque los bancos de pruebas no se ejecuten en este capítulo (no son obligatorios), estos paquetes son todavía necesarios para los bancos de pruebas en el Capítulo 6.

Prepara Tcl para su compilación:

```
cd unix
./configure --prefix=/tools
```

Construye el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`TZ=UTC make test`**. Se sabe que el banco de pruebas de Tcl experimenta fallos bajo ciertas condiciones del anfitrión que aún no se comprenden por completo. Sin embargo, estos fallos no son una sorpresa y no se consideran críticos. El parámetro *`TZ=UTC`* establece la zona horaria al Tiempo Universal Coordinado (UTC), también conocido como Hora del Meridiano de Greenwich (GMT), pero sólo mientras se ejecuta el banco de pruebas. Esto asegura que las pruebas de reloj se ejecuten correctamente. En el Capítulo 7 se proporcionan detalles sobre la variable de entorno `TZ`.

Instala el paquete:

```
make install
```

Aviso

No borres todavía el directorio de fuentes de `tcl8.4.7`, ya que el próximo paquete necesitará sus ficheros de cabecera internos.

Crea un enlace simbólico necesario:

```
ln -s tclsh8.4 /tools/bin/tclsh
```

5.10.2. Contenido de Tcl

Programas instalados: tclsh (enlace a tclsh8.4) y tclsh8.4
Librería instalada: libtcl8.4.so

Descripciones cortas

tclsh8.4	Es el intérprete de comandos de Tcl.
tclsh	Enlace a tclsh8.4
`libtcl8.4.so`	La librería Tcl.

5.11. Expect-5.42.1

El paquete Expect suministra un programa que mantiene diálogos programados con otros programas interactivos.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 3.9 MB

La instalación de Expect depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed y Tcl

5.11.1. Instalación de Expect

Corrige un error que puede causar falsos fallos durante la ejecución del banco de pruebas de GCC:

```
patch -Np1 -i ../expect-5.42.1-spawn-1.patch
```

Prepara Expect para su compilación:

```
./configure --prefix=/tools --with-tcl=/tools/lib --with-x=no
```

Significado de las opciones de configure:

`--with-tcl=/tools/lib`
Esto asegura que el guión configure encuentre la instalación de Tcl en nuestra ubicación temporal de herramientas. No queremos que encuentre una que pudiese residir en el sistema anfitrión.

`--with-x=no`
Esto le indica al guión configure que no busque Tk (el componente GUI de Tcl) o las librerías del sistema X Window, las cuales posiblemente se encuentren en el sistema anfitrión.

Construye el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make test`**. Sin embargo, se sabe que el banco de pruebas para Expect a veces experimenta fallos bajo ciertas condiciones del anfitrión que no están bajo nuestro control. Por tanto, estos fallos del banco de pruebas no son una sorpresa y no se consideran críticos.

Instala el paquete:

```
make SCRIPTS="" install
```

Significado del parámetro de make:

`SCRIPTS=""`
Esto evita la instalación de los guiones suplementarios de expect, que no son necesarios.

Ya puedes borrar los directorios de fuentes de Tcl y Expect.

5.11.2. Contenido de Expect

Programa instalado: expect
Librería instalada: libexpect-5.42.a

Descripciones cortas

expect	Se comunica con otros programas interactivos según un guión.
`libexpect-5.42.a`	Contiene funciones que permiten a Expect usarse como una extensión de Tcl o usarse directamente en C o C++ (sin Tcl)."

5.12. DejaGNU-1.4.4

El paquete DejaGNU contiene un entorno de trabajo para comprobar otros programas.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 8.6 MB

La instalación DejaGNU depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make y Sed

5.12.1. Instalación de DejaGNU

Prepara DejaGNU para su compilación:

```
./configure --prefix=/tools
```

Construye e instala el paquete:

```
make install
```

5.12.2. Contenido de DejaGNU

Programa instalado: runtest

Descripción corta

runtest Un guión envoltorio que encuentra el intérprete de comandos de **expect** adecuado y entonces ejecuta DejaGNU.

5.13. GCC-3.4.1 - Fase 2

Tiempo estimado de construcción: 11.0 SBU
Espacio requerido en disco: 274 MB

La instalación de GCC depende de: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed y Texinfo

5.13.1. Reinstalación de GCC

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como CFLAGS o CXXFLAGS, desactívala cuando construyas GCC.

Ahora están instaladas las herramientas necesarias para comprobar GCC y Binutils: Tcl, Expect y DejaGNU. Por lo que ahora pueden reconstruirse GCC y Binutils enlazándolos con la nueva Glibc y comprobarlos adecuadamente (si llevas a cabo los bancos de pruebas en este capítulo). Sin embargo, una cosa a tener en cuenta es que estos bancos de pruebas son altamente dependientes del correcto funcionamiento de las pseudo-terminales (PTYs) suministradas por tu distribución anfitrión. Las PTYs se implementan normalmente mediante el sistema de ficheros devpts. Comprueba si tu sistema anfitrión está configurado correctamente en este aspecto ejecutando una simple prueba:

```
expect -c "spawn ls"
```

La respuesta podría ser:

```
The system has no more ptys.
Ask your system administrator to create more.

El sistema no tiene más ptys.
Pídele al administrador del sistema que cree más.
```

Si recibes el mensaje anterior, tu sistema anfitrión no está configurado para operar correctamente con PTYs. En este caso no hay razón para ejecutar los bancos de pruebas de GCC y Binutils hasta resolver este asunto. Puedes consultar el Wiki de LFS en *http://wiki.linuxfromscratch.org/* para obtener información sobre cómo conseguir que funcionen las PTYs.

Puesto que se construirán los compiladores de C y C++, desempaqueta los paquetes core y g++ (y también el testsuite, si quieres ejecutar las pruebas). Al desempaquetarlos en el directorio de trabajo, todos ellos se ubicarán en un único subdirectorio gcc-3.4.1/.

Corrige un problema conocido y haz un ajuste esencial:

```
patch -Np1 -i ../gcc-3.4.1-no_fixincludes-1.patch
patch -Np1 -i ../gcc-3.4.1-specs-1.patch
```

El primer parche desactiva el guión **fixincludes** de GCC. Antes lo mencionamos brevemente, pero ahora queremos brindarte una explicación un poco más profunda del proceso de corrección de las cabeceras que realiza dicho guión. En circunstancias normales, el guión **fixincludes** de GCC busca en tu sistema los ficheros de cabecera que necesita corregir. Puede encontrar que algún fichero de cabecera de Glibc de tu sistema anfitrión necesite ser corregido, en cuyo caso lo corrige y lo pone en un directorio privado de GCC. Más adelante, en el Capítulo 6, después de instalar la nueva Glibc, se buscará en el directorio privado antes que en el directorio del sistema, por lo que GCC encontrará las cabeceras corregidas del sistema anfitrión, que muy probablemente no se corresponderán con la versión de Glibc utilizada para el sistema LFS.

El segundo parche cambia la localización por defecto para GCC del enlazador dinámico (normalmente

`ld-linux.so.2`). También elimina `/usr/include` de la ruta de búsqueda de GCC. Parchear ahora en lugar de ajustar el fichero specs tras la instalación asegura que nuestro nuevo enlazador dinámico sea utilizado durante la construcción actual de GCC. Esto es, todos los binarios finales (y temporales) creados durante la construcción se enlazarán contra la nueva Glibc.

Importante

Los parches anteriores son críticos para asegurar una correcta construcción. No olvides aplicarlos.

Vuelve a crear un directorio de construcción dedicado:

```
mkdir ../gcc-build
cd ../gcc-build
```

Prepara GCC para su compilación:

```
../gcc-3.4.1/configure --prefix=/tools \
    --libexecdir=/tools/lib --with-local-prefix=/tools \
    --enable-clocale=gnu --enable-shared \
    --enable-threads=posix --enable-__cxa_atexit \
    --enable-languages=c,c++ --disable-libstdcxx-pch
```

Significado de las nuevas opciones de configure:

`--enable-clocale=gnu`
Esta opción asegura que se seleccione el modelo de locale correcto para las librerías de C++ en todos los casos. Si el guión configure encuentra instalada la locale *de_DE*, seleccionará el modelo correcto de *gnu*. Sin embargo, las personas que no instalan la locale *de_DE* pueden correr el riesgo de construir librerías de C++ incompatibles en la ABI debido a que se selecciona el modelo de locale genérico, que es incorrecto.

`--enable-threads=posix`
Esto activa el manejo de excepciones C++ para código multihilo.

`--enable-__cxa_atexit`
Esta opción permite el uso de *__cxa_atexit*, en vez de *atexit*, para registrar destructores C++ para objetos estáticos locales y objetos globales. Es esencial para un manejo de destructores completamente compatible con los estándares. También afecta al ABI de C++ obteniendo librerías compartidas y programas C++ interoperables con otras distribuciones Linux.

`--enable-languages=c,c++`
Esta opción asegura que se construyan tanto el compilador de C como el de C++.

`--disable-libstdcxx-pch`
No construye la cabecera precompilada (PCH) para `libstdc++`. Necesita mucho espacio y nosotros no la utilizamos.

Compila el paquete:

```
make
```

Aquí no hace falta usar el objetivo *`bootstrap`*, ya que el compilador que estamos utilizando para construir GCC ha sido construido a partir de la misma versión de las fuentes de GCC que usamos antes.

La compilación está completa. Como se mencionó antes, no es obligatorio ejecutar los bancos de pruebas de

las herramientas temporales en este capítulo. Si de todas formas deseas ejecutar el banco de pruebas de GCC, hazlo con el siguiente comando:

```
make -k check
```

La opción *-k* se usa para que el banco de pruebas se ejecute por completo y sin detenerse ante el primer error. El banco de pruebas de GCC es muy exhaustivo y es casi seguro que generará algunos fallos. Para ver un resumen de los resultados ejecuta:

```
../gcc-3.4.1/contrib/test_summary
```

Para ver sólo el resumen, redirige la salida a través de **grep -A7 Summ**.

Puedes comparar tus resultados con los publicados en la lista de correo gcc-testresults para configuraciones similares a la tuya. Hay un ejemplo de cómo debería comportarse GCC-3.4.1 en sistemas i686-pc-linux-gnu en *http://gcc.gnu.org/ml/gcc-testresults/2004-07/msg00179.html*.

No siempre se pueden evitar unos cuantos fallos inesperados. Los desarrolladores de GCC normalmente están al tanto de dichos problemas, pero no los han resulelto aún. A no ser que tus resultados difieran en gran medida de los mostrados en la anterior URL, es seguro continuar.

Instala el paquete:

```
make install
```

En este punto se recomienda encarecidamente que se repitan las comprobaciones que realizamos anteriormente en este capítulo. Regresa a la Sección 5.9, “Ajustar las herramientas”, y repite la pequeña prueba de compilación. Si los resultados son malos muy posiblemente se deba a que no se aplicó correctamente el parche Specs de GCC.

Los detalles sobre este paquete se encuentran en la Sección 6.14.2, “Contenido de GCC”.

5.14. Binutils-2.15.91.0.2 - Fase 2

El paquete Binutils contiene un enlazador, un ensamblador y otras utilidades para trabajar con ficheros objeto.

Tiempo estimado de construcción: 1.5 SBU
Espacio requerido en disco: 108 MB

La instalación de Binutils depende de: Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed y Texinfo

5.14.1. Reinstalación de Binutils

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como CFLAGS o CXXFLAGS, desactívala cuando construyas Binutils.

Vuelve a crear un directorio dedicado para la construcción:

```
mkdir ../binutils-build
cd ../binutils-build
```

Prepara Binutils para su compilación:

```
../binutils-2.15.91.0.2/configure --prefix=/tools \
    --enable-shared --with-lib-path=/tools/lib
```

Significado de la nueva opción de configure:

--with-lib-path=/tools/lib
Esto le indica al guión configure que especifique la ruta de búsqueda de librerías por defecto durante la compilación de Binutils, resultando en que se le pase /tools/lib al enlazador. Esto evita que el enlazador busque en los directorios de librerías del anfitrión.

Compila el paquete:

```
make
```

La compilación está completa. Como se explicó antes, no recomendamos ejecutar los bancos de pruebas de las herramientas temporales en este capítulo. Si de todas formas deseas ejecutar el banco de pruebas de Binutils, hazlo con el siguiente comando:

```
make check
```

Instala el paquete:

```
make install
```

Prepara el enlazador para la fase de “Reajuste” del próximo capítulo:

```
make -C ld clean
make -C ld LIB_PATH=/usr/lib:/lib
```

Aviso

No borres todavía los directorios de fuentes y de construcción de Binutils. Se necesitarán durante el siguiente capítulo en el estado en que se encuentran ahora.

Los detalles sobre este paquete se encuentran en la Sección 6.13.2, “Contenido de Binutils”.

5.15. Gawk-3.1.4

El paquete Gawk contiene programas para manipular ficheros de texto.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 17 MB

La instalación de Gawk depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

5.15.1. Instalación de Gawk

Prepara Gawk para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.20.2, “Contenido de Gawk”.

5.16. Coreutils-5.2.1

El paquete Coreutils contiene utilidades para mostrar y establecer las características básicas del sistema.

Tiempo estimado de construcción: 0.9 SBU
Espacio requerido en disco: 69 MB

La instalación de Coreutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl y Sed

5.16.1. Instalación de Coreutils

Prepara Coreutils para su compilación:

```
DEFAULT_POSIX2_VERSION=199209 ./configure --prefix=/tools
```

Este paquete tiene un problema cuando se compila contra una versión de Glibc posterior a 2.3.2. Algunas de las utilidades de Coreutils (como **head**, **tail** y **sort**) rechazarán su sintaxis tradicional, la cual se ha usado desde hace aproximadamente unos 30 años. Esta vieja sintaxis está tan arraigada que debería preservarse la compatibilidad hasta que puedan actualizarse los múltiples sitios en la que se usa. La compatibilidad hacia atrás se consigue estableciendo en el anterior comando el valor de la variable de entorno `DEFAULT_POSIX2_VERSION` a “199209”. Si no deseas que Coreutils sea compatible con la sintaxis tradicional, simplemente omite dicha variable de entorno. Pero ten en cuenta que hacer esto tiene consecuencias, incluida la necesidad de parchear los múltiples paquetes que todavía utilizan la vieja sintaxis. Por tanto, nosotros recomendamos seguir las instrucciones mostradas.

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make RUN_EXPENSIVE_TESTS=yes check`**. El parámetro *`RUN_EXPENSIVE_TESTS=yes`* le indica al banco de pruebas que realice varias comprobaciones adicionales que se consideran relativamente costosas (en términos de uso de CPU y memoria) en ciertas plataformas, aunque normalmente no hay problemas en Linux.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.15.2, “Contenido de Coreutils”.

5.17. Bzip2-1.0.2

El paquete Bzip2 contiene programas para comprimir y descomprimir ficheros. En ficheros de texto consigue una mejor compresión que el tradicional **gzip**.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 2.5 MB

La instalación de Bzip2 depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc y Make

5.17.1. Instalación de Bzip2

El paquete Bzip2 no tiene un guión **configure**. Compílalo con:

```
make
```

Instala el paquete:

```
make PREFIX=/tools install
```

Los detalles sobre este paquete se encuentran en la Sección 6.40.2, “Contenido de Bzip2”.

5.18. Gzip-1.3.5

El paquete Gzip contiene programas para comprimir y descomprimir ficheros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 2.6 MB

La instalación de Gzip depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make y Sed

5.18.1. Instalación de Gzip

Prepara Gzip para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.46.2, “Contenido de Gzip”.

5.19. Diffutils-2.8.1

El paquete Diffutils contiene programas que muestran las diferencias entre ficheros o directorios.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 7.5 MB

La instalación de Diffutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

5.19.1. Instalación de Diffutils

Prepara Diffutils para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.41.2, “Contenido de Diffutils”.

5.20. Findutils-4.1.20

El paquete Findutils contiene programas para encontrar ficheros. Se suministran procesos para hacer búsquedas recursivas en un árbol de directorios, y para crear, mantener y consultar una base de datos (más rápida que la búsqueda recursiva, pero imprecisa si la base de datos no se ha actualizado recientemente).

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 7.6 MB

La instalación de Findutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

5.20.1. Instalación de Findutils

Prepara Findutils para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.19.2, “Contenido de Findutils”.

5.21. Make-3.80

El paquete Make contiene un programa para compilar paquetes grandes.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 8.8 MB

La instalación de Make depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep y Sed

5.21.1. Instalación de Make

Prepara Make para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.48.2, “Contenido de Make”.

5.22. Grep-2.5.1

El paquete Grep contiene programas para buscar dentro de ficheros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 5.8 MB

La instalación de Grep depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed y Texinfo

5.22.1. Instalación de Grep

Prepara Grep para su compilación:

```
./configure --prefix=/tools \
    --disable-perl-regexp --with-included-regex
```

Significado de las opciones de configure:

`--disable-perl-regexp`
Esto asegura que **grep** no se enlaza contra alguna librería PCRE que pudiese estar presente en el anfitrión y que no estaría disponible una vez que entremos en el entorno chroot.

`--with-included-regex`
Esto asegura que Grep utilice su código interno de expresiones regulares. Sin esta opción Grep usaría el código de Glibc, que se sabe que tiene algunos fallos.

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.44.2, “Contenido de Grep”.

5.23. Sed-4.1.2

El paquete Sed contiene un editor de flujos.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 5.2 MB

La instalación de Sed depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Texinfo

5.23.1. Instalación de Sed

Prepara Sed para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.28.2, “Contenido de Sed”.

5.24. Gettext-0.14.1

El paquete Gettext contiene utilidades para la internacionalización y localización. Esto permite a los programas compilarse con Soporte de Lenguaje Nativo (NLS), lo que les permite mostrar mensajes en el idioma nativo del usuario.

Tiempo estimado de construcción: 0.5 SBU
Espacio requerido en disco: 55 MB

La instalación de Gettext depende de: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make y Sed

5.24.1. Instalación de Gettext

Prepara Gettext para su compilación:

```
./configure --prefix=/tools --disable-libasprintf \
    --disable-csharp
```

Significado de las opciones de configure:

`--disable-libasprintf`
Esta opción le indica a Gettext que no construya la librería `asprintf`. Puesto que nada en este capítulo o el siguiente requiere dicha librería y Gettext se reconstruirá más adelante, la excluimos para salvar tiempo y espacio.

`--disable-csharp`
Esto le indica a Gettext que no utilice un compilador C#, aunque haya un compilador C# instalado en el anfitrión. Esto es necesario debido a que dentro del chroot C# no estará disponible.

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**. Esto tarda mucho tiempo, unos 7 SBUs. Más aún, se sabe que el banco de pruebas de Gettext falla bajo ciertas condiciones del anfitrión, por ejemplo si encuentra un compilador Java. En el proyecto Patches, en *http://www.linuxfromscratch.org/patches/*, hay disponible un parche experimental para desactivar Java.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.30.2, “Contenido de Gettext”.

5.25. Ncurses-5.4

El paquete Ncurses contiene librerías para el manejo de pantallas de caracteres independiente del terminal.

Tiempo estimado de construcción: 0.7 SBU
Espacio requerido en disco: 26 MB

La instalación de Ncurses depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make y Sed

5.25.1. Instalación de Ncurses

Prepara Ncurses para su compilación:

```
./configure --prefix=/tools --with-shared \
    --without-debug --without-ada --enable-overwrite
```

Significado de las opciones de configure:

`--without-ada`
Esto le indica a Ncurses que no construya su soporte para Ada, aunque haya un compilador Ada instalado en el anfitrión. Esto debe hacerse porque una vez dentro del chroot Ada no estará disponible.

`--enable-overwrite`
Esto le indica a Ncurses que instale sus ficheros de cabecera en `/tools/include` en vez de en `/tools/include/ncurses` para asegurar que otros paquetes puedan encontrar sin problemas las cabeceras de Ncurses.

Compila el paquete:

```
make
```

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.21.2, “Contenido de Ncurses”.

5.26. Patch-2.5.4

El paquete Patch contiene un programa para modificar ficheros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 1.9 MB

La instalación de Patch depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make y Sed

5.26.1. Instalación de Patch

Prepara Patch para su compilación:

```
CPPFLAGS=-D_GNU_SOURCE ./configure --prefix=/tools
```

La opción del preprocesador *-D_GNU_SOURCE* sólo es necesaria en la plataforma PowerPC. Puedes omitirla para otras arquitecturas.

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.50.2, “Contenido de Patch”.

5.27. Tar-1.14

El paquete Tar contiene un programa de archivado.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 10 MB

La instalación de Tar depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

5.27.1. Instalación de Tar

Prepara Tar para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.56.2, “Contenido de Tar”.

5.28. Texinfo-4.7

El paquete Texinfo contiene programas usados para leer, escribir y convertir documentos Info.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 16 MB

La instalación de Texinfo depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses y Sed

5.28.1. Instalación de Texinfo

Prepara Texinfo para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.34.2, “Contenido de Texinfo”.

5.29. Bash-3.0

El paquete Bash contiene la “Bourne-Again SHell”.

Tiempo estimado de construcción: 1.2 SBU
Espacio requerido en disco: 27 MB

La instalación de Bash depende de: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurse y Sed

5.29.1. Instalación de Bash

Prepara Bash para su compilación:

```
./configure --prefix=/tools --without-bash-malloc
```

Significado de la opción de configure:

`--without-bash-malloc`
Esta opción desactiva el uso de la función de ubicación de memoria (malloc) de Bash, que se sabe que provoca violaciones de segmento. Al desactivar esta opción Bash utilizará la función malloc de Glibc, que es más estable.

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make tests`**.

Instala el paquete:

```
make install
```

Crea un enlace para los programas que usan **sh** como intérprete de comandos:

```
ln -s bash /tools/bin/sh
```

Los detalles sobre este paquete se encuentran en la Sección 6.37.2, “Contenido de Bash”.

5.30. M4-1.4.2

El paquete M4 contiene un procesador de macros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 3.0 MB

La instalación de M4 depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl y Sed

5.30.1. Instalación de M4

Prepara M4 para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.24.2, “Contenido de M4”.

5.31. Bison-1.875a

El paquete Bison contiene un generador de analizadores sintácticos.

Tiempo estimado de construcción: 0.6 SBU
Espacio requerido en disco: 10.6 MB

La instalación de Bison depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make y Sed

5.31.1. Instalación de Bison

Prepara Bison para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre ste paquete se encuentran en la Sección 6.25.2, “Contenido de Bison”.

5.32. Flex-2.5.31

El paquete Flex contiene una utilidad para generar programas que reconocen patrones de texto.

Tiempo estimado de construcción: 0.6 SBU
Espacio requerido en disco: 10.6 MB

La instalación de Flex depende de: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make y Sed

5.32.1. Instalación de Flex

Flex contiene varios errores conocidos. Pueden corregirse con el siguiente parche:

```
patch -Np1 -i ../flex-2.5.31-debian_fixes-2.patch
```

Las autotools de GNU detectan que el código fuente de Flex fue modificado por dicho parche e intentan actualizar la página de manual. Esto no funciona correctamente en muchos sistemas y la página original es correcta, así que asegúrate que no sea regenerada:

```
touch doc/flex.1
```

Prepara Flex para su compilación:

```
./configure --prefix=/tools
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en la Sección 6.29.2, “Contenido de Flex”.

5.33. Util-linux-2.12b

El paquete Util-linux contiene una miscelánea de utilidades. Entre otras hay utilidades para manejar sistemas de ficheros, consolas, particiones y mensajes.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 16 MB

La instalación de Util-linux depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed y Zlib

5.33.1. Instalación de Util-linux

Util-linux no utiliza las cabeceras y librerías recién instaladas del directorio `/tools`. Esto se corrige modificando el guión configure:

```
sed -i 's@/usr/include@/tools/include@g' configure
```

Prepara Util-linux para su compilación:

```
./configure
```

Construye algunas rutinas de soporte:

```
make -C lib
```

Puesto que sólo necesitamos algunas de las utilidades incluidas en este paquete, construimos estas:

```
make -C mount mount umount
make -C text-utils more
```

Este paquete no incluye un banco de pruebas.

Copia estos programas al directorio de herramientas temporales:

```
cp mount/{,u}mount text-utils/more /tools/bin
```

Los detalles sobre este paquete se encuentran en la Sección 6.58.3, “Contenido de Util-linux”.

5.34. Perl-5.8.5

El paquete Perl contiene el Lenguaje Práctico de Extracción e Informe.

Tiempo estimado de construcción: 0.8 SBU
Espacio requerido en disco: 74 MB

La instalación de Perl depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make y Sed

5.34.1. Instalación de Perl

Aplica el siguiente parche para corregir algunas rutas a la librería C fijadas en el código:

```
patch -Np1 -i ../perl-5.8.5-libc-1.patch
```

Prepara Perl para su compilación (asegúrate de poner correctamente 'IO Fcntl POSIX', todo son letras):

```
./configure.gnu --prefix=/tools -Dstatic_ext='IO Fcntl POSIX'
```

Significado de la opción de configure:

`-Dstatic_ext='IO Fcntl POSIX'`
Esto le indica a Perl que construya el conjunto mínimo de extensiones estáticas necesarias para ejecutar el banco de pruebas de Coreutils en el siguiente capítulo.

Compila sólo las herramientas necesarias:

```
make perl utilities
```

Aunque Perl incluye un banco de pruebas, no es recomendable ejecutarlo ahora. Sólo se ha construido una parte de Perl y la ejecución de **`make test`** provocaría que también se compilase el resto de Perl, que es innecesario en este momento. El banco de pruebas puede ejecutarse en el siguiente capítulo, si se desea.

Copia estas herramientas y sus librerías:

```
cp perl pod/pod2man /tools/bin
mkdir -p /tools/lib/perl5/5.8.5
cp -R lib/* /tools/lib/perl5/5.8.5
```

Los detalles sobre este paquete se encuentran en la Sección 6.33.2, “Contenido de Perl”.

5.35. Udev-030

El paquete Udev contiene programas para la creación dinámica de nodos de dispositivos.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 5.2 MB

La instalación de Udev depende de: Coreutils y Make

5.35.1. Instalación de Udev

El programa **udevstart** tiene fijada la ruta al programa **udev** dentro suyo, lo que podría causar problemas pues **udev** se instalará en una ubicación no estándar. Corrige esto ejecutando lo siguiente:

```
sed -i 's@/sbin/udev@/tools/sbin/udev@g' udevstart.c
```

Del mismo modo, asegúrate que **udev** conoce la localización correcta de sus ficheros de configuración:

```
sed -i 's@/etc@/tools/etc@g' etc/udev/udev.conf.in
```

Compila Udev:

```
make prefix=/tools etcdir=/tools/etc
```

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make DESTDIR=/tools udevdir=/dev install
```

La configuración por defecto de Udev no es la ideal, así que instala aquí los ficheros de configuración especificos de LFS:

```
cp ../udev-config-2.permissions \
    /tools/etc/udev/permissions.d/00-lfs.permissions
cp ../udev-config-1.rules /tools/etc/udev/rules.d/00-lfs.rules
```

Lo detalles sobre este paquete se encuentran en la Sección 6.57.2, “Contenido de Udev”.

5.36. Eliminación de Símbolos

Los pasos de esta sección son opcionales, pero si la partición LFS es pequeña es bueno saber que se pueden eliminar algunas cosas innecesarias. Los binarios y librerías que se han construido contienen unos 130 MB de símbolos de depuración innecesarios. Elimina esos símbolos con:

```
strip --strip-debug /tools/lib/*
strip --strip-unneeded /tools/{,s}bin/*
```

El último de los comandos anteriores se saltará una veintena de ficheros, avisando que no reconoce su formato. Muchos de ellos son guiones en vez de binarios.

Ten cuidado de *no* utilizar *--strip-unneeded* con las librerías. Las estáticas se destruirían y tendrías que construir de nuevo los tres paquetes de las herramientas principales.

Para recuperar otros 30 MB, elimina la documentación:

```
rm -rf /tools/{doc,info,man}
```

Se necesitan como mínimo 850 MB de espacio libre en el sistema de ficheros LFS para poder construir e instalar Glibc en el siguiente capítulo. Si puedes construir e instalar Glibc, podrás construir e instalar el resto.

Parte III. Construcción del sistema LFS

Capítulo 6. Instalación de los programas del sistema base

6.1. Introducción

En este capítulo entramos en la zona de edificación y comenzamos a construir de verdad nuestro sistema LFS. Es decir, cambiamos la raíz a nuestro mini sistema Linux temporal, hacemos unos cuantos preparativos finales, y entonces comenzamos a instalar los paquetes.

La instalación de estos programas es bastante sencilla. Aunque en muchos casos las instrucciones podrían acortarse y ser más genéricas, hemos optado por suministrar las instrucciones completas para cada paquete para minimizar la posibilidad de errores. La clave para aprender qué hace que un sistema Linux funcione es conocer para qué se utiliza cada paquete y por qué el usuario (o el sistema) lo necesita. Para cada paquete instalado se incluye un sumario con su contenido, seguido de una concisa descripción de cada programa y librería instalados por el paquete.

Si piensas usar optimizaciones para la compilación durante este capítulo, mírate la receta de optimización en *http://www.lfs-es.com/recetas/optimization.html* (la versión original en inglés se encuentra en *http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt*). Las optimizaciones del compilador pueden hacer que un programa funcione más rápido, pero también pueden dificultar la compilación e incluso dar problemas al ejecutar el programa. Si un paquete rehusa compilar cuando se usan optimizaciones, prueba a compilarlo sin ellas y mira si el problema desaparece. Incluso si el paquete se compila usando optimización, existe el riesgo de que pueda haberse compilado incorrectamente debido a las complejas interacciones entre el código y las herramientas de construcción. La pequeña ganancia que se consigue usando optimizaciones en la compilación generalmente queda ensombrecida por los riesgos. Aconsejamos a los constructores primerizos de LFS que construyan sin optimizaciones personalizadas. Tu sistema aún será muy rápido y, al mismo tiempo, muy estable.

El orden en el que se instalan los paquetes en este capítulo debe respetarse estrictamente para asegurar que ningún programa inserte en su código una ruta referente a `/tools`. Por la misma razón, no compiles paquetes en paralelo. La compilación en paralelo puede ahorrarte algo de tiempo (sobre todo en máquinas con CPUs duales), pero puede generar un programa que contenga referencias a `/tools`, lo que provocaría que el programa dejase de funcionar cuando se elimine dicho directorio.

Antes de las instrucciones de instalación de cada paquete se muestra algo de información sobre el mismo: una breve descripción de lo que contiene, cuánto tardará aproximadamente en construirse, cuánto espacio en disco necesita durante el proceso de construcción, y qué otros paquetes necesita para construirse correctamente. A las instrucciones de instalación le sigue una lista de los programas y librerías que instala el paquete, junto con sus descripciones cortas.

Si quieres mantener un registro de qué ficheros instala cada paquete, puede que quieras utilizar un administrador de paquetes. Para tener una idea general de los administradores de paquetes consulta *http://www.lfs-es.com/blfs-es-CVS/introduction/important.html* (la versión original en inglés se encuentra en *http://www.linuxfromscratch.org/blfs/view/cvs/introduction/important.html*). Recomendamos que leas la receta *http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt*, pues es un método diseñado especialmente para LFS.

Nota

El resto de este libro debe realizarse como usuario *root*, no como usuario *lfs*.

6.2. Montar los sistemas de ficheros virtuales del núcleo

Varios sistemas de ficheros exportados por el núcleo no existen en el disco duro, pero son usados para comunicarse hacia y desde el propio núcleo.

Comienza creando los directorios sobre los que se montarán dichos sistemas de ficheros:

```
mkdir -p $LFS/{proc,sys}
```

Ahora monta los sistemas de ficheros:

```
mount -t proc proc $LFS/proc
mount -t sysfs sysfs $LFS/sys
```

Recuerda que si por alguna razón detienes tu trabajo con el sistema LFS y lo reinicias más tarde, es importante comprobar que estos sistemas de ficheros sean montados de nuevo antes de entrar en el entorno chroot.

Pronto se montarán sistemas de ficheros adicionales desde dentro del entorno chroot. Para mantener el anfitrión actualizado, realiza ahora un “falso montaje” para cada uno de ellos:

```
mount -f -t ramfs ramfs $LFS/dev
mount -f -t tmpfs tmpfs $LFS/dev/shm
mount -f -t devpts -o gid=4,mode=620 devpts $LFS/dev/pts
```

6.3. Entrar al entorno chroot

Es hora de entrar en el entorno chroot para iniciar la construcción e instalar tu sistema LFS final. Como usuario *root*, ejecuta el siguiente comando para entrar a un mundo que está, en este momento, poblado sólo por las herramientas temporales.

```
chroot "$LFS" /tools/bin/env -i \
    HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \
    /tools/bin/bash --login +h
```

La opción *-i* pasada al comando **env** limpiará todas las variables del chroot. Después de esto solamente se establecen de nuevo las variables HOME, TERM, PS1 y PATH. La construcción *TERM=$TERM* establece la variable TERM dentro del chroot al mismo valor que tiene fuera del chroot. Dicha variable es necesaria para que funcionen correctamente programas como **vim** y **less**. Si necesitas tener presentes otras variables, como CFLAGS o CXXFLAGS, este es un buen sitio para establecerlas.

Desde este punto ya no es necesario utilizar la variable LFS porque todo lo que hagas estará restringido al sistema de ficheros LFS. Esto se debe a que al intérprete de comandos se le dice que $LFS es ahora el directorio raíz (/).

Advierte que /tools/bin queda último en el PATH. Esto significa que una herramienta temporal no volverá a usarse a partir de que se instale su versión final. Esto ocurre cuando el intérprete de comandos no "recuerda" la localización de los binarios ejecutados; por esta razón se desactiva la tabla interna de rutas pasándole la opción *+h* a **bash**.

Debes asegurarte de que todos los comandos que aparecen en el resto de este y los siguientes capítulos son ejecutados dentro del entorno chroot. Si por alguna razón abandonas este entorno (tras un reinicio, por ejemplo), debes recordar montar primero los sistemas de ficheros proc y devpts (como explicamos en la sección anterior) y entrar de nuevo en el chroot antes de seguir con las instalaciones.

Ten en cuenta que en la línea de entrada de comandos de bash pondrá: "I have no name!" (¡No tengo nombre!). Esto es normal pues el fichero /etc/passwd aún no ha sido creado.

6.4. Cambio del propietario

En estos momentos el directorio `/tools` pertenece al usuario *lfs*, que sólo existe en el sistema anfitrión. Aunque probablemente quieras borrar el directorio `/tools` una vez que hayas terminado tu sistema LFS, también es posible que quieras conservarlo para, por ejemplo, construir más sistemas LFS. Pero si guardas el directorio `/tools` en el estado actual, acabarás con ficheros que pertenecen a un identificador de usuario sin cuenta correspondiente. Esto es peligroso porque una cuenta de usuario creada posteriormente podría tener esta identidad de usuario y poseería repentinamente los derechos sobre el directorio `/tools` y todos los ficheros que contiene, exponiéndolos a una posible manipulación.

Para evitar este problema, puedes añadir el usuario *lfs* al nuevo sistema LFS cuando creemos el fichero `/etc/passwd`, teniendo cuidado de asignarle los mismos identificadores de usuario y grupo que en el sistema anfitrión. Alternativamente, puedes asignar el contenido del directorio `/tools` al usuario *root* ejecutando el siguiente comando:

```
chown -R 0:0 /tools
```

Este comando utiliza *`0:0`* en lugar de *`root:root`*, pues **chown** no es capaz de resolver el nombre “root” hasta que el fichero de contraseñas sea creado. El libro asume que has ejecutado el anterior comando.

6.5. Creación de los directorios

Es hora de crear cierta estructura en el sistema de ficheros LFS. Crearemos un árbol de directorios. Usando los siguientes comandos se creará un árbol estándar:

```
install -d /{bin,boot,dev,etc/opt,home,lib,mnt}
install -d /{sbin,srv,usr/local,var,opt}
install -d /root -m 0750
install -d /tmp /var/tmp -m 1777
install -d /media/{floppy,cdrom}
install -d /usr/{bin,include,lib,sbin,share,src}
ln -s share/{man,doc,info} /usr
install -d /usr/share/{doc,info,locale,man}
install -d /usr/share/{misc,terminfo,zoneinfo}
install -d /usr/share/man/man{1,2,3,4,5,6,7,8}
install -d /usr/local/{bin,etc,include,lib,sbin,share,src}
ln -s share/{man,doc,info} /usr/local
install -d /usr/local/share/{doc,info,locale,man}
install -d /usr/local/share/{misc,terminfo,zoneinfo}
install -d /usr/local/share/man/man{1,2,3,4,5,6,7,8}
install -d /var/{lock,log,mail,run,spool}
install -d /var/{opt,cache,lib/{misc,locate},local}
install -d /opt/{bin,doc,include,info}
install -d /opt/{lib,man/man{1,2,3,4,5,6,7,8}}
```

Los directorios se crean por defecto con los permisos 755, pero esto no es deseable para todos los directorios. En los comandos anteriores se hacen dos cambios: uno para el directorio personal de *root* y otro para los directorios de los ficheros temporales.

El primer cambio nos asegura que nadie aparte de root pueda entrar en el directorio `/root`, lo mismo que debería hacer un usuario normal con su directorio personal. El segundo cambio nos asegura que cualquier usuario pueda escribir en los directorios `/tmp` y `/var/tmp`, pero no pueda borrar los ficheros de otros usuarios. Esto último lo prohibe el llamado “bit pegajoso” (sticky bit), el bit más alto (1) en la máscara de permisos 1777.

6.5.1. Nota de conformidad con FHS

El árbol de directorios está basado en el estándar FHS (disponible en *http://www.pathname.com/fhs/*). Además del árbol arriba creado, dicho estándar estipula la existencia de `/usr/local/games` y `/usr/share/games`. No los recomendamos para un sistem base, sin embargo, eres libre de hacer que tu sistema cumpla el FHS. Como sobre la estructura del subdirectorio `/usr/local/share` el FHS no es preciso, creamos aquí sólo los directorios que son necesarios.

6.6. Creación de enlaces simbólicos esenciales

Algunos programas tienen fijadas en su código rutas a programas que aún no existen. Para satisfacer a estos programas creamos unos cuantos enlaces simbólicos que serán sustituidos por ficheros reales durante el transcurso de este capítulo a medida que vayamos instalando todos los programas.

```
ln -s /tools/bin/{bash,cat,pwd,stty} /bin
ln -s /tools/bin/perl /usr/bin
ln -s /tools/lib/libgcc_s.so.1 /usr/lib
ln -s bash /bin/sh
```

6.7. Creación de los ficheros de contraseñas, grupos y registro

Para que *root* pueda entrar al sistema y para que el nombre “root” sea reconocido, es necesario tener las entradas apropiadas en los ficheros `/etc/passwd` y `/etc/group`.

Crea el fichero `/etc/passwd` ejecutando el siguiente comando:

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
EOF
```

La contraseña real para *root* (la “x” es sólo un sustituto) se establecerá más adelante.

Crea el fichero `/etc/group` ejecutando el siguiente comando:

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
EOF
```

Los grupos creados no son parte de ningún estándar, son los grupos que la configuración de Udev utilizará en la siguiente sección. El LSB (Linux Standard Base, *http://www.linuxbase.org/*) sólo recomienda que, aparte del grupo “root” con GID 0, esté presente un grupo “bin” con GID 1. Todos los demás nombres de grupos y sus GID pueden ser elegidos libremente por el usuario, pues los paquetes correctamente escritos no dependen del número GID, sino que utilizan el nombre del grupo.

Para eliminar el “I have no name!” del símbolo del sistema, iniciaremos un nuevo intérprete de comandos. Puesto que instalamos una Glibc completa en el Capítulo 5 y acabamos de crear los ficheros `/etc/passwd` y `/etc/group`, la resolución de nombres de usuarios y grupos funcionará ahora.

```
exec /tools/bin/bash --login +h
```

Advierte el uso de la directiva *+h*. Esto le indica a **bash** que no utilice su tabla interna de rutas. Sin esta directiva, **bash** recordaría la ruta a los binarios que ha ejecutado. Para poder usar los binarios recién compilados tan pronto como sean instalados, se desactiva esta función durante el resto de este capítulo.

Los programas **login**, **getty** e **init** (entre otros) mantienen una serie de ficheros de registro con información sobre quienes están y estaban dentro del sistema. Sin embargo, estos programas no crean dichos ficheros si no existen. Crea los ficheros de registro con sus permisos correctos:

```
touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}
chgrp utmp /var/run/utmp /var/log/lastlog
chmod 664 /var/run/utmp /var/log/lastlog
```

El fichero `/var/run/utmp` lista los usuarios que están actualmente dentro del sistema, `/var/log/wtmp` registra todos los ingresos y salidas. El fichero `/var/log/lastlog` muestra, para cada usuario, cuando fue la última vez que ingresó, y el fichero `/var/log/btmp` lista los intentos de ingreso fallidos.

6.8. Poblar /dev

6.8.1. Crear los nodos de dispositivo iniciales

Cuando el núcleo arranca el sistema necesita la presencia de ciertos nodos de dispositivo, en concreto los dispositivos `console` y `null`. Créalos ejecutando los siguientes comandos:

```
mknod -m 600 /dev/console c 5 1
mknod -m 666 /dev/null c 1 3
```

6.8.2. Montar ramfs y poblar /dev

El método ideal para poblar `/dev` es montar un `ramfs` en `/dev`, similar a `tmpfs`, y crear en él los dispositivos en cada arranque. Puesto que el sistema no ha sido arrancado, es necesario hacer lo que los guiones de arranque habrían hecho y poblar `/dev`. Comienza montando `/dev`:

```
mount -n -t ramfs none /dev
```

Ejecuta el programa **udevstart** instalado para crear los dispositivos iniciales basándose en la información que hay en `/sys`:

```
/tools/sbin/udevstart
```

Ciertos enlaces simbólicos y directorios requeridos por LFS no son creados por Udev, así que crealos:

```
ln -s /proc/self/fd /dev/fd
ln -s /proc/self/fd/0 /dev/stdin
ln -s /proc/self/fd/1 /dev/stdout
ln -s /proc/self/fd/2 /dev/stderr
ln -s /proc/kcore /dev/core
mkdir /dev/pts
mkdir /dev/shm
```

Finalmente, monta los sistemas de ficheros virtuales (del núcleo) adecuados en los directorios recién creados:

```
mount -t devpts -o gid=4,mode=620 none /dev/pts
mount -t tmpfs none /dev/shm
```

El comando **mount** ejecutado arriba puede mostrar el siguiente aviso:

```
can't open /etc/fstab: No such file or directory

no puedo abrir /etc/fstab: No existe el fichero o directorio.
```

El fichero `/etc/fstab` no ha sido creado todavía, pero tampoco es necesario para que los sistemas de ficheros se monten correctamente. Por tanto, puedes ignorar el aviso con tranquilidad.

6.9. Linux-Libc-Headers-2.6.8.1

El paquete Linux-Libc-Headers contiene las cabeceras “saneadas” del núcleo.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 22 MB

La instalación de Linux-Libc-Headers depende de: Coreutils

6.9.1. Instalación de Linux-Libc-Headers

Durante años ha sido una práctica común utilizar las cabeceras “crudas” del núcleo (procedentes de un paquete del núcleo) en `/usr/include`, pero en los últimos años los desarrolladores del núcleo han expresado su firme opinión de que eso no debe hacerse. Esto dió lugar al nacimiento del proyecto Linux-Libc-Headers, que fue diseñado para mantener una versión estable de la API de la cabeceras Linux.

Instala los ficheros de cabecera:

```
cp -R include/asm-i386 /usr/include/asm
cp -R include/linux /usr/include
```

Asegúrate de que todas las cabeceras son propiedad de root:

```
chown -R root:root /usr/include/{asm,linux}
```

Asegúrate de que los usuarios pueden leer las cabeceras:

```
find /usr/include/{asm,linux} -type d -exec chmod 755 {} \;
find /usr/include/{asm,linux} -type f -exec chmod 644 {} \;
```

6.9.2. Contenido de Linux-Libc-Headers

Cabeceras instaladas: /usr/include/{asm,linux}/*.h

Descripción corta

`/usr/include/{asm,linux}/*.h` La API de las cabeceras de Linux.

6.10. Man-pages-1.67

El paquete Man-pages contiene alrededor de 1.200 páginas de manual.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 15 MB

La instalación de Man-pages depende de: Bash, Coreutils y Make

6.10.1. Instalación de Man-pages

Instala Man-pages ejecutando:

```
make install
```

6.10.2. Contenido de Man-pages

Ficheros instalados: diversas páginas de manual

Descripción corta

`páginas de manual`	Describen las funciones C y C++, los ficheros de dispositivo importantes y los ficheros de configuración más significativos.

6.11. Glibc-2.3.4-20040701

El paquete Glibc contiene la librería C principal. Esta librería proporciona todas las rutinas básicas para la ubicación de memoria, búsqueda de directorios, abrir y cerrar ficheros, leerlos y escribirlos, manejo de cadenas, coincidencia de patrones, aritmética, etc...

Tiempo estimado de construcción: 12.3 SBU
Espacio requerido en disco: 784 MB

La instalación de Glibc depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed y Texinfo

6.11.1. Instalación de Glibc

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como `CFLAGS` o `CXXFLAGS`, desactívala cuando construyas Glibc.

El sistema de construcción de Glibc está muy bien autocontenido y se instalará perfectamente, incluso aunque nuestro fichero de especificaciones del compilador y los guiones del enlazador todavía apunten a `/tools`. No podemos ajustar las especificaciones y el enlazador antes de instalar Glibc, porque entonces las comprobaciones del autoconf de Glibc darían resultados erróneos y esto arruinaría nuestro objetivo de conseguir una construcción limpia.

La documentación de Glibc recomienda construirlo fuera del árbol de las fuentes, en un directorio de construcción dedicado:

```
mkdir ../glibc-build
cd ../glibc-build
```

Prepara Glibc para su compilación:

```
../glibc-2.3.4-20040701/configure --prefix=/usr \
    --disable-profile --enable-add-ons=nptl --with-tls \
    --with-__thread --enable-kernel=2.6.0 --without-cvs \
    --libexecdir=/usr/lib/glibc \
    --with-headers=/tools/glibc-kernheaders
```

Significado de la nueva opción de configure:

--libexecdir=/usr/lib/glibc
: Esto cambia la localización del programa **pt_chown** de su ubicación por defecto `/usr/libexec` a `/usr/lib/glibc`.

Compila el paquete:

```
make
```

Importante

En esta sección, el banco de pruebas para Glibc se considera crítico. No te lo saltes bajo ninguna circunstancia.

Comprueba los resultados:

```
make check
```

El banco de pruebas de Glibc depende en gran medida de ciertas funciones de tu sistema anfitrión, en particular del núcleo. En general, se espera que el banco de pruebas de Glibc pase siempre con éxito. Sin embargo, bajo ciertas circunstancias algunos fallos son inevitables. Aquí hay una lista con los problemas más comunes:

- La prueba *math* falla en ocasiones cuando se ejecuta en sistemas donde la CPU no es una Intel genuina o una AMD genuina relativamente nueva. Es sabido que ciertos ajustes de optimización también afectan.
- La prueba *gettext* falla en ocasiones debido a problemas del sistema anfitrión. La razón exacta aún no está clara.
- La prueba *atime* falla en ocasiones cuando la partición LFS está montada con la opción *`noatime`*.
- La prueba *shm* puede fallar en el caso de que el sistema anfitrión utilice el sistema de ficheros `devfs` pero no tenga un sistema de ficheros `tmpfs` montado en `/dev/shm`. Esto sucede debido a la falta de soporte para `tmpfs` en el núcleo.
- Cuando se ejecutan en hardware antiguo y lento, varias pruebas pueden fallar debido a que se excede el tiempo estimado.

Aunque se trata de un mensaje inofensivo, la fase de instalación de Glibc se quejará de la ausencia de `/etc/ld.so.conf`. Evita este molesto aviso con:

```
touch /etc/ld.so.conf
```

Instala el paquete:

```
make install
```

Las locales que hacen que tu sistema responda en un idioma diferente no se instalaron con el comando anterior. Hazlo con este:

```
make localedata/install-locales
```

Para ahorrar tiempo, una alternativa al comando anterior (que genera e instala todas las locales que Glibc conoce) es instalar solamente aquellas locales que necesites o desees. Esto puede hacerse usando el comando **localedef**. Puedes encontrar más información sobre esto en el fichero `INSTALL` de las fuentes de Glibc. Sin embargo, hay un número de locales que son esenciales para que las comprobaciones de paquetes posteriores se realicen. En particular, la prueba de *libstdc++* en GCC. Las siguientes instrucciones, en vez del objetivo anterior *`install-locales`*, instalarán el conjunto mínimo de locales necesario para que las pruebas se ejecuten correctamente:

```
mkdir -p /usr/lib/locale
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i en_HK -f ISO-8859-1 en_HK
localedef -i en_PH -f ISO-8859-1 en_PH
localedef -i en_US -f ISO-8859-1 en_US
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

Algunas de las locales instaladas por el anterior comando **make localedata/install-locales** no están correctamente soportadas por algunas de las aplicaciones que hay en los libros LFS y BLFS. Debido a los diversos problemas provocados por presunciones de los programadores, que rompen dichas locales, LFS no debería utilizarse con locales que utilicen conjuntos de caracteres multibyte (incluido UTF-8) o escritura de derecha a izquierda. Se requieren numerosos parches no oficiales e inestables para corregir estos problemas y el equipo de desarrolladores de LFS ha decidido no soportar ese tipo de locales complejas. Esto se aplica también a las locales ja_JA y fa_IR, que se han instalado sólo para pasar las pruebas de GCC y Gettext, y el programa **watch** (que es parte del paquete Procps) no fuciona correctamente en ellas. Varios intentos para evitar estas restricciones están documentados en las recetas relacionadas con internacionalización.

Construye las páginas de manual de linuxthreads, que son una gran referencia sobre la API de hilos (aplicable también a NPTL):

```
make -C ../glibc-2.3.4-20040701/linuxthreads/man
```

Instala dichas páginas:

```
make -C ../glibc-2.3.4-20040701/linuxthreads/man install
```

6.11.2. Configuración de Glibc

Necesitamos crear el fichero `/etc/nsswitch.conf`, porque aunque Glibc nos facilita los valores por defecto cuando este fichero no se encuentra o está corrupto, estos valores por defecto no funcionan bien con la conexión de red. También tendremos que configurar nuestra zona horaria.

Crea un nuevo fichero `/etc/nsswitch.conf` ejecutando lo siguiente:

```
cat > /etc/nsswitch.conf << "EOF"
# Inicio de /etc/nsswitch.conf

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

protocols: files
services: files
ethers: files
rpc: files

# Fin de /etc/nsswitch.conf
EOF
```

Para determinar la zona horaria local, ejecuta el siguiente guión:

```
tzselect
```

Después de contestar unas preguntas referentes a tu localización, el guión te mostrará el nombre de tu zona horaria, algo como *EST5EDT* o *Canada/Eastern*. Crea entonces el fichero `/etc/localtime` ejecutando:

```
cp --remove-destination /usr/share/zoneinfo/[xxx] \
    /etc/localtime
```

Sustituye `[xxx]` con el nombre de la zona horaria facilitado por **tzselect** (por ejemplo, Europe/Madrid).

Significado de la opción de cp:

`--remove-destination`
: Esto es necesario para forzar la eliminación del enlace simbólico que ya existe. La razón por la que copiamos en lugar de enlazar es para cubrir el caso en el que `/usr` está en otra partición. Esto puede ser importante cuando se arranca en modo de usuario único.

6.11.3. Configuración del cargador dinámico

Por defecto, el cargador dinámico (`/lib/ld-linux.so.2`) busca en `/lib` y `/usr/lib` las librerías dinámicas que necesitan los programas cuando los ejecutas. No obstante, si hay librerías en otros directorios que no sean `/lib` y `/usr/lib`, necesitas añadirlos al fichero de configuración `/etc/ld.so.conf` para que el cargador dinámico pueda encontrarlas. Dos directorios típicos que contienen librerías adicionales son `/usr/local/lib` y `/opt/lib`, así que añadimos estos directorios a la ruta de búsqueda del cargador dinámico.

Crea un nuevo fichero `/etc/ld.so.conf` ejecutando lo siguiente:

```
cat > /etc/ld.so.conf << "EOF"
# Inicio de /etc/ld.so.conf

/usr/local/lib
/opt/lib

# Fin de /etc/ld.so.conf
EOF
```

6.11.4. Contenido de Glibc

Programas instalados: catchsegv, gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nscd_nischeck, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump y zic
Librerías instaladas: ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd-compat.a, libc.[a,so], libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so y libutil.[a,so]

Descripciones cortas

catchsegv	Puede usarse para crear una traza de la pila cuando un programa termina con una violación de segmento.
gencat	Genera catálogos de mensajes.
getconf	Muestra los valores de configuración del sistema para variables específicas del sistema de ficheros.
getent	Obtiene entradas de una base de datos administrativa.
iconv	Realiza conversiones de juego de caracteres.
iconvconfig	Crea un fichero de configuración para la carga rápida del módulo **iconv**.
ldconfig	Configura las asociaciones en tiempo de ejecución para el enlazador dinámico.
ldd	Muestra las librerías compartidas requeridas por cada programa o librería especificados.
lddlibc4	Asiste a **ldd** con los ficheros objeto.
locale	Le dice al compilador que active o desactive el uso de las locales POSIX para operaciones integradas.
localedef	Compila las especificaciones de locales.
mtrace	Lee e interpreta un fichero de traza de memoria y muestra un sumario en formato legible.
nscd	Un demonio que suministra una caché para las peticiones más comunes al servidor de nombres.
nscd_nischeck	Comprueba si es necesario o no un modo seguro para búsquedas NIS+.
pcprofiledump	Vuelca la información generada por un perfil de PC.
pt_chown	Un programa de ayuda para **grantpt** que establece el propietario, grupo y permisos de acceso para un pseudo-terminal esclavo.
rpcgen	Genera código C para implementar el protocolo RPC.
rpcinfo	Hace una llamada RPC a un servidor RPC.
sln	Un programa **ln** enlazado estáticamente.
sprof	Lee y muestra los datos del perfil de los objetos compartidos.
tzselect	Pregunta al usuario información sobre la localización actual y muestra la descripción de la zona horaria correspondiente.
xtrace	Traza la ejecución de un programa mostrando la función actualmente ejecutada.
zdump	El visualizador de la zona horaria.
zic	El compilador de la zona horaria.
`ld.so`	El programa de ayuda para las librerías compartidas ejecutables.
`libBrokenLocale`	Usada por programas como Mozilla para resolver locales rotas.
`libSegFault`	El manejador de señales de violación de segmento.

`libanl`	Una librería de búsqueda de nombres asíncrona.
`libbsd-compat`	Proporciona la portabilidad necesaria para ejecutar ciertos programas BSD en Linux.
`libc`	La librería principal de C.
`libcrypt`	La librería criptográfica.
`libdl`	La librería de interfaz del enlazado dinámico.
`libg`	Una librería en tiempo de ejecución para **g++**.
`libieee`	La librería de punto flotante del IEEE.
`libm`	La librería matemática.
`libmcheck`	Contiene código ejecutado en el arranque.
`libmemusage`	Usada por **memusage** para ayudar a recoger información sobre el uso de memoria de un programa.
`libnsl`	La librería de servicios de red.
`libnss`	Las librerías Name Service Switch (Interruptor del Servicio de Nombres). Contienen funciones para resolver nombres de sistemas, usuarios, grupos, alias, servicios, protocolos y similares.
`libpcprofile`	Contiene funciones de perfiles utilizadas para observar la cantidad de tiempo de CPU utilizado por líneas concretas del código fuente.
`libpthread`	La librería de hilos POSIX.
`libresolv`	Proporciona funciones para la creación, envío e interpretación de paquetes de datos a servidores de nombres de dominio de Internet.
`librpcsvc`	Proporciona funciones para una miscelánea de servicios RPC.
`librt`	Proporciona funciones para muchas de las interfaces especificadas por el POSIX.1b Realtime Extension (Extensiones en Tiempo Real POSIX.1b).
`libthread_db`	Contiene funciones útiles para construir depuradores para programas multihilo.
`libutil`	Contiene código para funciones “estándar” usadas en diferentes utilidades Unix.

6.12. Reajustar las herramientas

Ahora que hemos instalado las nuevas y finales librerías de C, es hora de ajustar de nuevo el conjunto de herramientas. Las ajustaremos para que enlacen cualquier nuevo programa compilado contra estas nuevas librerías. De hecho es lo mismo que hicimos en la fase "Ajustar" al principio del Capítulo 5, aunque parezca lo contario. En el Capítulo 5 el cambio iba de los directorios `/{,usr}/lib` del anfitrión a los nuevos `/tools/lib`. Ahora es guiado de `/tools/lib` a los directorios `/{,usr}/lib` del LFS.

Comienza ajustando el enlazador. Para ello conservamos los directorios de fuentes y de construcción de la segunda fase de Binutils. Instala el enlazador ajustado ejecutando el siguiente comando desde el directorio `binutils-build`:

```
make -C ld INSTALL=/tools/bin/install install
```

Nota

Si de algún modo te saltaste el aviso sobre conservar los directorios de las fuentes y construcción del segundo paso de Binutils en el Capítulo 5, o los borraste accidentalmente, o no tienes acceso a ellos, ignora el comando anterior. El resultado será que el siguiente paquete, Binutils, se enlazará contra las librerías C que hay en `/tools` en vez de las de `/{,usr}/lib`. Esto no es lo ideal, pero nuestras pruebas han mostrado que los programas binarios de Binutils resultantes deberían ser idénticos.

Desde ahora todos los programas que compilemos se enlazarán solamente contra las librerías que hay en `/usr/lib` y `/lib`. La opción *`INSTALL=/tools/bin/install`* extra es necesaria porque el `Makefile` creado durante el segundo paso todavía contiene la referencia a `/usr/bin/install`, que obviamente aún no ha sido instalado. Algunas distribuciones tienen un enlace simbólico `ginstall` que tiene preferencia en el `Makefile` y puede crear problemas aquí. El comando anterior también evita esto.

Elimina los directorios de fuentes y de construcción de Binutils.

A continuación, corrige el fichero specs de GCC para que apunte al nuevo enlazador dinámico. Un comando sed lo consigue:

```
sed -i 's@ /tools/lib/ld-linux.so.2@ /lib/ld-linux.so.2@g' \
    `gcc --print-file specs`
```

Es buena idea inspeccionar visualmente el fichero de especificaciones para verificar que realmente se produjeron los cambios deseados.

Importante

Si estás trabajando sobre una plataforma en la que el nombre del enlazador dinámico no sea `ld-linux.so.2`, sustituye "ld-linux.so.2" en el comando anterior por el nombre del enlazador dinámico para tu plataforma. Si es necesario, consulta la Sección 5.3, "Notas técnicas sobre las herramientas".

Atención

En este punto es obligatorio parar y asegurarse de que las operaciones básicas (compilación y enlazado) de las nuevas herramientas funcionan como se espera. Para esto vamos a hacer una simple comprobación:

```
echo 'main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /lib'
```

Si todo funciona correctamente, no debe haber errores y la salida del último comando debe ser (con las diferencias para la plataforma sobre el nombre del enlazador dinámico):

```
[Requesting program interpreter: /lib/ld-linux.so.2]

[Intérprete de programa solicitado: /lib/ld-linux.so.2]
```

Comprueba que `/lib` aparezca como prefijo de tu enlazador dinámico.

Si no recibes una salida como la mostrada arriba, o no hay salida alguna, algo está seriamente mal. Necesitarás investigar y revisar tus pasos para encontrar el problema y corregirlo. La razón más probable es que algo salió mal durante el anterior arreglo del fichero specs. Deberás resolver todos los problemas antes de seguir con el proceso.

Una vez que todo funcione coorectamente, borra los ficheros de prueba:

```
rm dummy.c a.out
```

6.13. Binutils-2.15.91.0.2

El paquete Binutils contiene un enlazador, un ensamblador y otras utilidades para trabajar con ficheros objeto.

Tiempo estimado de construcción: 1.4 SBU
Espacio requerido en disco: 167 MB

La instalación de Binutils depende de: Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed y Texinfo

6.13.1. Instalación de Binutils

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como CFLAGS o CXXFLAGS, desactívala cuando construyas Binutils.

Verifica que tus pseudo-terminales (PTYs) funcionan adecuadamente dentro del entorno chroot. Comprueba que todo está correcto realizando una simple prueba:

```
expect -c "spawn ls"
```

Si recibes el siguiente mensaje, el entorno chroot no está correctamente configurado para operar con PTYs:

```
The system has no more ptys.
Ask your system administrator to create more.

El sistema no tiene más ptys.
Pídele al administrador del sistema que cree más.
```

Debes solucionar el problema antes de ejecutar los bancos de pruebas de Binutils y GCC.

La documentación de Binutils recomienda construirlo fuera del árbol de las fuentes, en un directorio de construcción dedicado:

```
mkdir ../binutils-build
cd ../binutils-build
```

Prepara Binutils para su compilación:

```
../binutils-2.15.91.0.2/configure --prefix=/usr \
    --enable-shared
```

Compila el paquete:

```
make tooldir=/usr
```

Normalmente, tooldir (el directorio donde se instalarán los ejecutables) se establece como `$(exec_prefix)/$(target_alias)`, lo que se convierte en `/usr/i686-pc-linux-gnu`. Como este es un sistema personalizado, no es necesario tener en `/usr` dicho directorio específico de un objetivo. Esa configuración se utilizaría si el sistema fuese usado para compilación cruzada (por ejemplo, para compilar un paquete en una máquina Intel, pero generando código que se ejecutará en máquinas PowerPC).

Importante

En esta sección, el banco de pruebas para Binutils se considera crítico. No te lo saltes bajo ninguna circunstancia.

Comprueba los resultados:

```
make check
```

Instala el paquete:

```
make tooldir=/usr install
```

Instala el fichero de cabecera `libiberty`, pues lo necesitan algunos paquetes:

```
cp ../binutils-2.15.91.0.2/include/libiberty.h /usr/include
```

6.13.2. Contenido de Binutils

Programas instalados: addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings y strip
Librerías instaladas: libiberty.a, libbfd.[a,so] y libopcodes.[a,so]

Descripciones cortas

addr2line Traduce direcciones de programas a nombres de ficheros y números de líneas. Dándole una dirección y un ejecutable, usa la información de depuración del ejecutable para averiguar qué fichero y número de línea está asociado con dicha dirección.

ar Crea, modifica y extrae desde archivos.

as Un ensamblador que ensambla la salida de **gcc** dentro de ficheros objeto.

c++filt Es usado por el enlazador para decodificar símbolos de C++ y Java, guardando las funciones sobrecargadas para su clasificación.

gprof Muestra los datos del perfil del gráfico de llamada.

ld Un enlazador que combina un número de ficheros objeto y de archivos en un único fichero, reubicando sus datos y estableciendo las referencias a los símbolos.

nm Lista los símbolos que aparecen en un fichero objeto dado.

objcopy Traduce un tipo de ficheros objeto a otro.

objdump Muestra información sobre el fichero objeto indicado, con opciones para controlar la información a mostrar. La información mostrada es útil fundamentalmente para los programadores que trabajan sobre las herramientas de compilación.

ranlib Genera un índice de los contenidos de un archivo, y lo coloca en el archivo. El índice lista cada símbolo definido por los miembros del archivo que son ficheros objeto reubicables.

readelf Muestra información sobre binarios de tipo ELF.

size Lista los tamaños de las secciones y el tamaño total para los ficheros objeto indicados.

strings	Muestra, para cada fichero indicado, las cadenas de caracteres imprimibles de al menos la longitud especificada (4 por defecto). Para los ficheros objeto muestra, por defecto, sólo las cadenas procedentes de las secciones de inicialización y carga. Para otros tipos de ficheros explora el fichero al completo.
strip	Elimina símbolos de ficheros objeto.
`libiberty`	Contiene rutinas usadas por varios programas GNU, incluidos **getopt**, **obstack**, **strerror**, **strtol** y **strtoul**.
`libbfd`	La librería del Descriptor de Fichero Binario.
`libopcodes`	Una librería para manejar mnemónicos. Se usa para construir utilidades como **objdump**. Los mnemónicos son las versiones en “texto legible” de las instrucciones del procesador.

6.14. GCC-3.4.1

El paquete GCC contiene la colección de compiladores GNU, que incluye los compiladores C y C++.

Tiempo estimado de construcción: 11.7 SBU
Espacio requerido en disco: 294 MB

La instalación de GCC depende de: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed y Texinfo

6.14.1. Instalación de GCC

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como `CFLAGS` o `CXXFLAGS`, desactívala cuando construyas GCC.

Desempaqueta los paquetes gcc-core y gcc-g++, que se desempaquetarán dentro de un mismo directorio. Además debes extraer el paquete gcc-testsuite. El paquete completo GCC contiene otros compiladores más. En *http://www.lfs-es.com/blfs-es-CVS/general/gcc.html* puedes encontrar las instrucciones para construirlos (en *http://www.linuxfromscratch.org/blfs/view/svn/general/gcc.html* se encuentra la versión original en inglés).

Aplica sólo el parche No-Fixincludes (pero no el parche Specs), que también se usó en el capítulo anterior:

```
patch -Np1 -i ../gcc-3.4.1-no_fixincludes-1.patch
```

GCC falla al compilar algunos paquetes ajenos a la instalación base de Linux From Scratch (como Mozilla y kdegraphics) cuando se usa en conjunción con la más nueva versión de Binutils. Aplica el siguiente parche para corregir dicho problema:

```
patch -Np1 -i ../gcc-3.4.1-linkonce-1.patch
```

Aplica una sustitución sed que suprimirá la instalación de `libiberty.a`. Queremos usar la versión de `libiberty.a` suministrada por Binutils:

```
sed -i 's/install_to_$(INSTALL_DEST) //' libiberty/Makefile.in
```

La documentación de GCC recomienda construirlo fuera del árbol de las fuentes, en un directorio de construcción dedicado:

```
mkdir ../gcc-build
cd ../gcc-build
```

Prepara GCC para su compilación:

```
../gcc-3.4.1/configure --prefix=/usr \
    --libexecdir=/usr/lib --enable-shared \
    --enable-threads=posix --enable-__cxa_atexit \
    --enable-clocale=gnu --enable-languages=c,c++
```

Compila el paquete:

```
make
```

Importante

> En esta sección, el banco de pruebas para GCC se considera crítico. No te lo saltes bajo ninguna circunstancia.

Comprueba los resultados, pero no pares en los errores:

```
make -k check
```

Algunos errores son conocidos y se mencionaron en el capítulo anterior. Las notas para el banco de pruebas que hay en la Sección 5.13, “GCC-3.4.1 - Fase 2” son aún más apropiadas aquí. Asegúrate de consultarlas si es necesario.

Instala el paquete:

```
make install
```

Algunos paquetes esperan que el preprocesador de C esté instalado en el directorio `/lib`. Para dar soporte a estos paquetes, crea un enlace simbólico:

```
ln -s ../usr/bin/cpp /lib
```

Muchos paquetes usan el nombre **cc** para llamar al compilador de C. Para satisfacer a estos paquetes, crea un enlace simbólico:

```
ln -s gcc /usr/bin/cc
```

Nota

En este punto es muy recomendable repetir la comprobación que realizamos anteriormente en este capítulo. Vuelve a la Sección 6.12, “Reajustar las herramientas” y repite las comprobaciones. Si los resultados son malos, entonces es muy posible que erróneamente hayas aplicado el parche Specs para GCC del Capítulo 5.

6.14.2. Contenido de GCC

Programas instalados: c++, cc (enlace a gcc), cpp, g++, gcc, gccbug y gcov
Librerías instaladas: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] y libsupc++.a

Descripciones cortas

cc	El compilador de C.
cpp	El preprocesador de C. Lo usa el compilador para expandir las sentencias #include, #define y similares en los ficheros fuente.
c++	El compilador de C++.
g++	El compilador de C++.
gcc	El compilador de C.
gccbug	Un guión del intérprete de comandos que ayuda a crear notificaciones de errores.
gcov	Una herramienta para pruebas de rendimiento. Se usa para analizar programas y encontrar qué optimizaciones tendrán el mayor efecto.
`libgcc`	Contienen el soporte en tiempo de ejecución para **gcc**.

`libstdc++` La librería estándar de C++.

`libsupc++` Proporciona rutinas de soporte para el lenguaje de programación c++.

6.15. Coreutils-5.2.1

El paquete Coreutils contiene utilidades para mostrar y establecer las características básicas del sistema.

Tiempo estimado de construcción: 0.9 SBU
Espacio requerido en disco: 69 MB

La instalación de Coreutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl y Sed

6.15.1. Instalación de Coreutils

Un problema conocido en el programa **uname** de este paquete es que la opción *-p* siempre devuelve unknown (desconocido). El siguiente parche corrige este comportamiento en arquitecturas Intel:

```
patch -Np1 -i ../coreutils-5.2.1-uname-2.patch
```

Evita que Coreutils instale binarios que serán instalados más tarde por otros paquetes:

```
patch -Np1 -i \
    ../coreutils-5.2.1-suppress_uptime_kill_su-1.patch
```

Prepara Coreutils para su compilación:

```
DEFAULT_POSIX2_VERSION=199209 ./configure --prefix=/usr
```

Compila el paquete:

```
make
```

El banco de pruebas de Coreutils hace ciertas suposiciones relativas a la presencia de ficheros y usuarios que no son válidos tan temprano en la construcción de LFS. Por tanto hay que preparar varias cosas antes de poder ejecutar las pruebas. Si decides no ejecutar el banco de pruebas, salta hasta “Instala el paquete”.

Crea dos grupos y un usuario ficticios:

```
echo "dummy1:x:1000:" >> /etc/group
echo "dummy2:x:1001:dummy" >> /etc/group
echo "dummy:x:1000:1000:::/bin/bash" >> /etc/passwd
```

Ahora todo está preparado para ejecutar el banco de pruebas. Primero ejecuta las pruebas que requieren que se ejecuten como *root*:

```
make NON_ROOT_USERNAME=dummy check-root
```

A continuación ejecuta el resto como usuario *dummy*:

```
src/su dummy -c "make RUN_EXPENSIVE_TESTS=yes check"
```

Cuando termines con las pruebas, elimina los grupos y el usuario ficticios:

```
sed -i '/dummy/d' /etc/passwd /etc/group
```

Instala el paquete:

```
make install
```

Mueve algunos programas a sus ubicaciones adecuadas:

```
mv /usr/bin/{[,basename,cat,chgrp,chmod,chown,cp,dd,df} /bin
mv /usr/bin/{date,echo,false,head,hostname,install,ln} /bin
mv /usr/bin/{ls,mkdir,mknod,mv,pwd,rm,rmdir,sync} /bin
mv /usr/bin/{sleep,stty,test,touch,true,uname} /bin
mv /usr/bin/chroot /usr/sbin
```

Finalmente, crea un enlace simbólico para cumplir con el FHS:

```
ln -s ../../bin/install /usr/bin
```

6.15.2. Contenido de Coreutils

Programas instalados: basename, cat, chgrp, chmod, chown, chroot, cksum, comm, cp, csplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, hostname, id, install, join, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, users, vdir, wc, who, whoami y yes

Descripciones cortas

basename Elimina cualquier ruta y sufijo indicado de un nombre de fichero.

cat Concatena ficheros en la salida estándar.

chgrp Cambia el grupo propietario de cada fichero dado al grupo especificado. El grupo puede indicarse tanto por su nombre como por su identificador numérico.

chmod Cambia los permisos de cada fichero dado al modo indicado. El modo puede ser una representación simbólica de los cambios a hacer o un número octal que representa los nuevos permisos.

chown Cambia el usuario y/o el grupo al que pertenece cada fichero dado al par usuario/grupo indicado.

chroot Ejecuta un comando usando el directorio especificado como directorio `/`.

cksum Muestra la suma de comprobación CRC (Comprobación Cíclica Redundante) y cuenta los bytes de cada fichero especificado.

comm Compara dos ficheros ordenados, sacando en tres columnas las líneas que son únicas y las líneas que son comunes.

cp Copia ficheros.

csplit Trocea un fichero en varios nuevos ficheros, separándolos de acuerdo a un patrón indicado o a un número de líneas, y muestra el número de bytes de cada nuevo fichero.

cut Imprime fragmentos de líneas, seleccionando los fragmentos de acuerdo a los campos o posiciones indicadas.

date Muestra la fecha y hora actual en un formato determinado o establece la fecha y hora del sistema.

dd Copia un fichero usando el tamaño y número de bloques indicado, mientras que, opcionalmente, realiza conversiones en él.

df	Muestra la cantidad de espacio disponible (y usado) en todos los sistemas de ficheros montados, o solo del sistema de ficheros en el que se encuentran los ficheros seleccionados.
dir	Lista el contenido del directirio indicado (lo mismo que **ls**).
dircolors	Imprime comandos para modificar la variable de entorno `LS_COLOR`, para cambiar el esquema de color usado por **ls**.
dirname	Elimina los sufijos que no son directorios del nombre de un fichero.
du	Muestra la cantidad de espacio en disco usado por el directorio actual o por cada directorio indicado (incluyendo todos sus subdirectorios) o por cada fichero indicado.
echo	Muestra la cadena indicada.
env	Ejecuta un programa en un entorno modificado.
expand	Convierte las tabulaciones a espacios.
expr	Evalúa expresiones.
factor	Muestra los factores primos de los números enteros especificados.
false	No hace nada, infructuoso. Siempre termina con un código de estado que indica un fallo.
fmt	Reformatea cada párrafo de los ficheros especificados.
fold	Reajusta la longitud de línea en cada fichero dado.
groups	Muestra los grupos a los que pertenece un usuario.
head	Imprime las 10 primeras líneas (o el número de líneas indicado) de un fichero.
hostid	Muestra el identificador numérico (en hexadecimal) de la máquina actual.
hostname	Muestra o establece el nombre de la máquina actual.
id	Muestra los identificadores efectivos de usuario y grupo, y los grupos a los que pertenece, del usuario actual o de un usuario dado.
install	Copia ficheros mientras establece sus permisos y, si es posible, su propietario y grupo.
join	Une a partir de dos ficheros las líneas que tienen campos de unión idénticos.
link	Crea un enlace duro con el nombre indicado de un fichero dado.
ln	Crea enlaces duros o blandos (simbólicos) entre ficheros.
logname	Muestra el nombre de acceso del usuario actual.
ls	Lista el contenido de cada directorio indicado.
md5sum	Muestra o verifica sumas de comprobación MD5 (Mensaje de Resúmen 5).
mkdir	Crea directorios con los nombres indicados.
mkfifo	Crea tuberías (FIFO, el primero en entrar, el primero en salir) con los nombres indicados.
mknod	Crea nodos de dispositivos con los nombres indicados. Un nodo de dispositivo es un fichero especial de caracteres o un fichero especial de bloques o una tubería.
mv	Mueve o renombra ficheros o directorios.
nice	Ejecuta un programa con una prioridad distinta.

nl Numera las líneas de los ficheros dados.

nohup Ejecuta un comando que no se interrumpe cuando se cierra la sesión, con su salida redirigida a un fichero de registro.

od Vuelca ficheros en octal y otros formatos.

paste Mezcla los ficheros indicados, uniendo secuencialmente las líneas correspondientes de uno y otro, separándolas con tabulaciones.

pathchk Comprueba si los nombres de ficheros son válidos o portables.

pinky Es un cliente finger ligero. Muestra algo de información sobre un determinado usuario.

pr Pagina y encolumna el texto de un fichero para imprimirlo.

printenv Muestra el entorno.

printf Muestra los argumentos dados de acuerdo al formato indicado. Muy parecido a la función printf de C.

ptx Genera un índice permutado de los contenidos de un fichero, con cada palabra clave en su contexto.

pwd Muestra el nombre del directorio de trabajo actual.

readlink Muestra el valor del enlace simbólico indicado.

rm Elimina ficheros o directorios.

rmdir Elimina directorios si están vacíos.

seq Muestra una secuencia de números, dentro de un cierto rango y con un cierto incremento.

sha1sum Muestra o verifica sumas de comprobación SHA1 de 160 bits.

shred Sobreescribe los ficheros indicados repetidamente con patrones extraños, haciendo realmente difícil recuperar los datos.

sleep Hace una pausa por el tiempo indicado.

sort Ordena las líneas de los ficheros indicados.

split Divide un fichero en trozos, por tamaño o por número de líneas.

stat Muestra el estado de ficheros o sistemas de ficheros.

stty Establece o muestra los ajuste de línea del terminal.

sum Muestra la suma de comprobación y el número de bloques para cada fichero dado.

sync Refresca los almacenadores intermedios de los sistemas de ficheros. Fuerza el guardado de los bloques modificados al disco y actualiza el superbloque.

tac Concatena en orden inverso los ficheros indicados.

tail Imprime las últimas 10 líneas (o el número de líneas indicado) de cada fichero dado.

tee Lee de la entrada estándar y escribe tanto en la salida estándar como en los ficheros indicados.

test Comprueba el tipo de los ficheros y compara valores.

touch	Cambia las fechas de modificación o acceso de cada fichero especificado, poniéndole la fecha actual. Si un fichero no existe crea uno vacío.
tr	Convierte, altera y borra caracteres de la entrada estándar.
true	No hace nada, conseguido. Siempre termina con un código de estado que indica éxito.
tsort	Realiza una ordenación topológica. Escribe una lista totalmente ordenada de acuerdo con el orden parcial del fichero especificado.
tty	Muestra el nombre de fichero del terminal conectado a la entrada estándar.
uname	Muestra información del sistema.
unexpand	Convierte los espacios en tabulaciones.
uniq	Elimina líneas consecutivas duplicadas.
unlink	Elimina el fichero indicado.
users	Muestra los nombres de los usuarios conectados actualmente.
vdir	Es lo mismo que **ls -l**.
wc	Muestra el número de líneas, palabras y bytes de un fichero, y una línea con el total si se ha especificado más de uno.
who	Muestra quién está conectado.
whoami	Muestra el nombre de usuario asociado con el identificador de usuario efectivo actual.
yes	Muestra en pantalla “y” o una cadena de texto dada indefinidamente, hasta que es matado.

6.16. Zlib-1.2.1

El paquete Zlib contiene rutinas de compresión y descompresión usadas por algunos programas.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 1.5 MB

La instalación de Zlib depende de: Binutils, Coreutils, GCC, Glibc, Make y Sed

6.16.1. Instalación de Zlib

El siguiente parche corrige una vulnerabilidad de Denegación de Servicio en la librería de compresión de Zlib:

```
patch -Np1 -i ../zlib-1.2.1-security-1.patch
```

Nota

Se sabe que Zlib construye incorrectamente sus librerías si en el entorno se ha especificado un `CFLAGS`. Si estás usando tu propia variable `CFLAGS`, asegúrate de añadirle la directiva *`-fPIC`* durante el siguiente comando de configuración, y elimínala posteriormente.

Prepara Zlib para su compilación:

```
./configure --prefix=/usr --shared
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala la librería compartida:

```
make install
```

Construye también la librería estática:

```
make clean
./configure --prefix=/usr
make
```

Para obtener de nuevo los resultados de las pruebas, ejecuta: **`make check`**.

Instala la librería estática:

```
make install
```

Corrige los permisos de la librería estática:

```
chmod 644 /usr/lib/libz.a
```

Es una buena política y práctica común colocar las librerías importantes en el directorio `/lib`. Esto es muy importante en los casos en que `/usr` se encuentra en una partición separada. Esencialmente, las librerías con componentes en tiempo de ejecución usadas por los programas de `/bin` o `/sbin` deben residir en `/lib` para que estén en la partición raiz y disponibles en el caso de que `/usr` sea inaccesible.

Por esta razón movemos los componentes en tiempo de ejecución de la Zlib compartida a `/lib`:

```
mv /usr/lib/libz.so.* /lib
```

Corrige el enlace simbólico `/usr/lib/libz.so`:

```
ln -sf ../../lib/libz.so.1 /usr/lib/libz.so
```

6.16.2. Contenido de Zlib

Librerías instaladas: libz[a,so]

Descripción corta

`libz` Contiene funciones de compresión y descompresión usadas por algunos programas.

6.17. Mktemp-1.5

El paquete Mktemp contiene programas usados para crear ficheros temporales seguros en guiones de intérpretes de comandos.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 317 KB

La instalación de Mktemp depende de: Coreutils, Make y Patch

6.17.1. Instalación de Mktemp

Muchos programas todavía usan el anticuado programa **tempfile**, que tiene una funcionalidad similar a **mktemp**. Parchea Mktemp para incluir un envoltorio **tempfile**:

```
patch -Np1 -i ../mktemp-1.5-add_tempfile-1.patch
```

Prepara Mktemp para su compilación:

```
./configure --prefix=/usr --with-libc
```

Significado de la opción de configure:

`--with-libc`
Esto hace que el programa **mktemp** utilice las funciones *mkstemp* y *mkdtemp* de la librería C del sistema.

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
make install-tempfile
```

6.17.2. Contenido de Mktemp

Programas instalados: mktemp y tempfile

Descripciones cortas

mktemp Crea ficheros temporales de forma segura. Es usado en guiones.

tempfile Crea ficheros temporales de una forma menos segura que **mktemp**. Se instala por retro-compatibilidad.

6.18. Iana-Etc-1.01

El paquete Iana-Etc contiene datos de servicios y protocolos de red.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 641 KB

La instalación de Iana-Etc depende de: Make

6.18.1. Instalación de Iana-Etc

Procesa los datos:

```
make
```

Instala el paquete:

```
make install
```

6.18.2. Contenido de Iana-Etc

Ficheros instalados: /etc/protocols y /etc/services

Descripciones cortas

`/etc/protocols`	Describe los diversos protocolos DARPA para Internet que están disponibles para el subsistema TCP/IP.
`/etc/services`	Proporciona un mapeado entre los nombres familiares de los servicios de Internet y los números de puerto y tipo de protocolo que tienen asignados.

6.19. Findutils-4.1.20

El paquete Findutils contiene programas para encontrar ficheros. Se suministran procesos para hacer búsquedas recursivas en un árbol de directorios, y para crear, mantener y consultar una base de datos (más rápida que la búsqueda recursiva, pero imprecisa si la base de datos no se ha actualizado recientemente).

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 7.5 MB

La instalación de Findutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

6.19.1. Instalación de Findutils

Prepara Findutils para su compilación:

```
./configure --prefix=/usr --libexecdir=/usr/lib/locate \
    --localstatedir=/var/lib/locate
```

La anterior directiva *`localstatedir`* cambia la localización de la base de datos de **locate** a `/var/lib/locate`, que cumple el FHS.

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.19.2. Contenido de Findutils

Programas instalados: bigram, code, find, frcode, locate, updatedb y xargs

Descripciones cortas

bigram Se usaba originalmente para generar bases de datos de **locate**.

code Se usaba originalmente para generar bases de datos de **locate**. Es el antecesor de **frcode**.

find Busca en los árboles de directorios indicados los ficheros que cumplan el criterio especificado.

frcode Es llamado por **updatedb** para comprimir la lista de nombres de ficheros. Utiliza "front-compression", que reduce el tamaño de la base de datos en un factor de 4 o 5.

locate Busca en una base de datos de nombres de ficheros y muestra los nombres que contienen la cadena indicada o cumplen un patrón dado.

updatedb Actualiza la base de datos de **locate**. Explora por completo el sistema de ficheros (incluidos otros sistemas de ficheros que se encuentren montados, a no ser que se le indique lo contrario) e inserta todos los nombres de ficheros que encuentre en la base de datos.

xargs Puede usarse para aplicar un comando a una lista de ficheros.

6.20. Gawk-3.1.4

El paquete Gawk contiene programas para manipular ficheros de texto.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 17 MB

La instalación de Gawk depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

6.20.1. Instalación de Gawk

Prepara Gawk para su compilación:

```
./configure --prefix=/usr --libexecdir=/usr/lib
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.20.2. Contenido de Gawk

Programas instalados: awk (enlace a gawk), gawk, gawk-3.1.4, grcat, igawk, pgawk, pgawk-3.1.4 y pwcat

Descripciones cortas

awk	Enlace a **gawk**
gawk	Un programa para manipular ficheros de texto. Es la implementación GNU de **awk**.
gawk-3.1.4	Enlace duro a **gawk**.
grcat	Vuelca la base de datos de grupos `/etc/group`.
igawk	Otorga a **gawk** la capacidad de incluir ficheros.
pgawk	Es la versión de **gawk** con soporte de perfiles.
pgawk-3.1.4	Enlace duro a **pgawk**.
pwcat	Vuelca la base de datos de contraseñas `/etc/passwd`.

6.21. Ncurses-5.4

El paquete Ncurses contiene librerías para el manejo de pantallas de caracteres independiente del terminal.

Tiempo estimado de construcción: 0.6 SBU
Espacio requerido en disco: 27 MB

La instalación de Ncurses depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make y Sed

6.21.1. Instalación de Ncurses

Prepara Ncurses para su compilación:

```
./configure --prefix=/usr --with-shared --without-debug
```

Compila el paquete:

```
make
```

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make install
```

Otorga permisos de ejecución a las librerías Ncurses:

```
chmod 755 /usr/lib/*.5.4
```

Corrige una librería que no debería ser ejecutable:

```
chmod 644 /usr/lib/libncurses++.a
```

Mueve las librerías al directorio `/lib`, que es donde se espera que residan:

```
mv /usr/lib/libncurses.so.5* /lib
```

Debido a que se han movido las librerías, algunos enlaces simbólicos apuntan a ficheros que no existen. Regenera esos enlaces simbólicos:

```
ln -sf ../../lib/libncurses.so.5 /usr/lib/libncurses.so
ln -sf libncurses.so /usr/lib/libcurses.so
```

6.21.2. Contenido de Ncurses

Programas instalados: captoinfo (enlace a tic), clear, infocmp, infotocap (enlace a tic), reset (enlace a tset), tack, tic, toe, tput y tset
Librerías instaladas: libcurses.[a,so] (enlace a libncurses.[a,so]), libform.[a,so], libmenu.[a,so], libncurses++.a, libncurses.[a,so] y libpanel.[a,so]

Descripciones cortas

captoinfo Convierte una descripción termcap en una descripción terminfo.

clear Limpia la pantalla si es posible.

infocmp	Compara o imprime en pantalla una descripción terminfo.
infotocap	Convierte una descripción terminfo en una descripción termcap.
reset	Reinicializa un terminal a sus valores por defecto.
tack	El comprobador de acciones terminfo. Se usa principalmente para verificar la precisión de una entrada de la base de datos terminfo.
tic	El compilador de entradas de descripciones terminfo. Transforma un fichero terminfo en formato fuente al formato binario requerido por las rutinas de las librerías ncurses. Los ficheros terminfo contienen información sobre las capacidades de un terminal.
toe	Lista todos los tipos de terminal disponibles, dando el nombre primario y la descripción de cada uno.
tput	Pone a disposición del intérprete de comandos la información sobre las capacidades dependientes del terminal. También sirve para inicializar o restablecer el terminal, o para devolver su nombre largo.
tset	Sirve para inicializar terminales.
`libcurses`	Enlace a `libncurses`
`libncurses`	Contienen funciones para mostrar texto de formas complicadas en la pantalla de un terminal. Un buen ejemplo del uso de estas funciones es el menú que se muestra en el proceso **make menuconfig** del núcleo.
`libform`	Contienen funciones para implementar formularios.
`libmenu`	Contienen funciones para implementar menús.
`libpanel`	Contienen funciones para implementar paneles.

6.22. Readline-5.0

El paquete Readline contiene la librería de línea de comandos Readline.

Tiempo estimado de construcción: 0.11 SBU
Espacio requerido en disco: 3.8 MB

La instalación de Readline depende de: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses y Sed

6.22.1. Instalación de Readline

El siguiente parche corrige un problema por el que Readline, en ocasiones, muestra sólo 33 caracteres en una línea y salta a la siguiente línea.

```
patch -Np1 -i ../readline-5.0-display_wrap-1.patch
```

Prepara Readline para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make SHLIB_XLDFLAGS=-lncurses
```

Significado de la opción de make:

`SHLIB_XLDFLAGS=-lncurses`
Esta opción fuerza a Readline a enlazarse contra la libería `libncurses`.

Instala el paquete:

```
make install
```

Asigna a las librerías dinámicas de Readline unos permisos más apropiados:

```
chmod 755 /usr/lib/*.5.0
```

Mueve las librerías dinámicas a una ubicación más correcta:

```
mv /usr/lib/lib{readline,history}.so.5* /lib
```

Debido a que se han movido las librerías, algunos enlaces simbólicos apuntan ahora a ficheros que no existen. Regenera dichos enlaces simbólicos:

```
ln -sf ../../lib/libhistory.so.5 /usr/lib/libhistory.so
ln -sf ../../lib/libreadline.so.5 /usr/lib/libreadline.so
```

6.22.2. Contenido de Readline

Librerías instaladas: libhistory.[a,so] y libreadline.[a,so]

Descripciones cortas

`libhistory`	Proporciona una interfaz de usuario consistente para la rellamada de líneas de historial.
`libreadline`	Asiste en la consistencia de la interfaz de usuario entre programas discrecionales que necesitan suministrar una interfaz de línea de comandos.

6.23. Vim-6.3

El paquete Vim contiene un poderoso editor de texto.

Tiempo estimado de construcción: 0.4 SBU
Espacio requerido en disco: 34 MB

La instalación de Vim depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses y Sed

Alternativas a Vim
Si prefieres otro editor en vez de Vim, como Emacs, Joe, o Nano, mira en *http://www.lfs-es.com/blfs-es-CVS/postlfs/editors.html* las instrucciones de instalación sugeridas (la versión original en inglés se encuentra en *http://www.linuxfromscratch.org/blfs/view/stable/postlfs/editors.html*).

6.23.1. Instalación de Vim

Primero, desempaqueta en el mismo directorio tanto `vim-6.3.tar.bz2` como (opcionalmente) `vim-6.3-lang.tar.gz`. Después, cambia la localización por defecto de los ficheros de configuración `vimrc` y `gvimrc` a `/etc`.

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
echo '#define SYS_GVIMRC_FILE "/etc/gvimrc"' >> src/feature.h
```

Prepara Vim para su compilación:

```
./configure --prefix=/usr --enable-multibyte
```

La opción *`--enable-multibyte`*, opcional pero muy recomendable, añade a **vim** el soporte para la edición de ficheros codificados con caracteres multibyte. Esto es necesario si se utiliza un conjunto de caracteres multibyte. También permite editar ficheros creados inicialmente en distribuciones Linux como Fedora Core, que utilizan UTF-8 como conjunto de caracteres por defecto.

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make test`**. Sin embargo, este banco de pruebas mostrará una caótica avalancha de caracteres por pantalla que puede causar problemas con los ajustes del terminal actual. Por tanto la ejecución de este banco de pruebas es opcional.

Instala el paquete

```
make install
```

Muchos usuarios tienden a utilizar **vi**, en vez de **vim**. Para permitirles ejecutar **vim** cuando teclean **vi**, crea un enlace simbólico:

```
ln -s vim /usr/bin/vi
```

Si vas a instalar el sistema X Window en tu sistema LFS, puede que sea necesario recompilar Vim después de instalar X. Vim incluye una bonita versión con interfaz gráfica que necesita X y algunas otras librerías instaladas. Para más información lee la documentación de Vim y la página de instalación de Vim en el libro BLFS, en *http://www.lfs-es.com/blfs-es-CVS/postlfs/editors.html#postlfs-editors-vim* (el original en inglés se encuentra en *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html#postlfs-editors-vim*).

6.23.2. Configuración de Vim

Por defecto, **vim** se ejecuta en modo no compatible con vi. Esto puede ser nuevo para los usuarios que han utilizado otros editores anteriormente. Hemos incluido a continuación la opción "nocompatible" para resaltar el hecho de que se va a usar este nuevo comportamiento. Esto también les recuerda a aquellos que quieran cambiar al modo "compatible" que debería aparecer al principio. Esto es necesario porque cambia otros ajustes, y las modificaciones deberían ir tras este ajuste. Crea un fichero de configuración por defecto de **vim** ejecutando lo siguiente:

```
cat > /etc/vimrc << "EOF"
" Inicio de /etc/vimrc

set nocompatible
set backspace=2
syntax on
if (&term == "iterm") || (&term == "putty")
  set background=dark
endif

" Fin de /etc/vimrc
EOF
```

La opción *set nocompatible* hace que **vim** se comporte de un modo (el modo por defecto) más útil que el modo compatible con vi. Elimina el "no" si quieres el antiguo comportamiento **vi**. La opción *set backspace=2* permite el retroceso en saltos de línea, autoindentación e inicio de inserción. La opción *syntax on* activa la coloración semántica de **vim**. Por último, el condicional *if* junto con *set background=dark* corrige lo que **vim** se imagina sobre el color de fondo de ciertos emuladores de terminal. Esto le da a la coloración semántica un mejor esquema de color para utilizarlo sobre el fondo negro de estos programas.

Se puede obtener información sobre las opciones disponibles ejecutando el siguiente comando:

```
vim -c ':options'
```

6.23.3. Contenido de Vim

Programas instalados: efm_filter.pl, efm_perl.pl, ex (enlace a vim), less.sh, mve.awk, pltags.pl, ref, rview (enlace a vim), rvim (enlace a vim), shtags.pl, tcltags, vi (enlace a vim), view (enlace a vim), vim, vim132, vim2html.pl, vimdiff (enlace a vim), vimm, vimspell.sh, vimtutor y xxd

Descripciones cortas

efm_filter.pl	Un filtro para crear un fichero de error que puede ser leído por **vim**.
efm_perl.pl	Formatea los mensajes de error del intérprete Perl para usarlos con el modo "quickfix" de **vim**.
ex	Arranca **vim** en modo ex.
less.sh	Un guión que arranca **vim** con less.vim.

mve.awk	Procesa los errores de **vim**.
pltags.pl	Crea un fichero de etiquetas para el código Perl, de modo que pueda usarse con **vim**.
ref	Comprueba la ortografía de los argumentos.
rview	Una versión restringida de **view**. No pueden ejecutarse comandos del intérprete de comandos y **view** no puede ser suspendido.
rvim	Una versión restringida de **vim**. No pueden ejecutarse comandos del intérprete de comandos y **vim** no puede ser suspendido.
shtags.pl	Genera un fichero de etiquetas para los guiones Perl.
tcltags	Genera un fichero de etiquetas para el código TCL.
view	Arranca **vim** en modo de sólo lectura.
vim	El editor.
vim132	Arranca **vim** con el terminal en modo de 132 columnas.
vim2html.pl	Convierte la documentación de Vim a HTML.
vimdiff	Edita dos o tres versiones de un fichero con **vim** y muestra las diferencias.
vimm	Activa el modelo de entrada del buscador de DEC en un terminal remoto.
vimspell.sh	Corrige un fichero y genera las sentencias de sintaxis necesarias para resaltar las palabras en **vim**. Este guión necesita el antiguo comando Unix **spell**, que no se incluye en el LFS ni en el BLFS.
vimtutor	Enseña las teclas y comandos básicos de **vim**.
xxd	Genera un volcado hexadecimal. También puede hacer lo contrario, por lo que puede usarse para parchear binarios.

6.24. M4-1.4.2

El paquete M4 contiene un procesador de macros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 3.0 MB

La instalación de M4 depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl y Sed

6.24.1. Instalación de M4

Prepara M4 para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.24.2. Contenido de M4

Programa instalado: m4

Descripción corta

m4 Copia los ficheros dados expandiendo en el proceso las macros que contengan. Estas macros pueden ser internas o definidas por el usuario y pueden tomar cualquier número de argumentos. Además de hacer la expansión de macros, **m4** tiene funciones internas para incluir los ficheros indicados, lanzar comandos UNIX, hacer aritmética entera, manipular texto de diversas formas, recursión, etc. El programa **m4** puede ser usado como interfaz para un compilador o como procesador de macros por sí mismo.

6.25. Bison-1.875a

El paquete Bison contiene un generador de analizadores sintácticos.

Tiempo estimado de construcción: 0.6 SBU
Espacio requerido en disco: 10.6 MB

La instalación de Bison depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make y Sed

6.25.1. Instalación de Bison

Prepara Bison para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.25.2. Contenido de Bison

Programas instalados: bison y yacc
Librería instalada: liby.a

Descripciones cortas

bison Genera, a partir de una serie de reglas, un programa para analizar la estructura de ficheros de texto. Bison es un sustituto de Yacc (Yet Another Compiler Compiler, Otro Compilador de Compiladores).

yacc Un envoltorio para **bison**, destinado a los programas que todavía llaman a **yacc** en lugar de a **bison**. Invoca a **bison** con la opción *`-y`*.

`liby.a` La librería Yacc que contiene la implementación de las funciones *yyerror* y *main* compatibles con Yacc. Esta librería normalmente no es muy útil, pero POSIX la solicita.

6.26. Less-382

El paquete Less contiene un visor de ficheros de texto.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 3.4 MB

La instalación de Less depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses y Sed

6.26.1. Instalación de Less

Prepara Less para su compilación:

```
./configure --prefix=/usr --bindir=/bin --sysconfdir=/etc
```

Significado de la opción de configure:

`--sysconfdir=/etc`
Esta opción le indica al programa creado por el paquete que busque en `/etc` sus ficheros de configuración.

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

6.26.2. Contenido de Less

Programas instalados: less, lessecho y lesskey

Descripciones cortas

less Un visor de ficheros o paginador. Muestra el contenido de un fichero con la posibilidad de recorrerlo, hacer búsquedas o saltar a marcas.

lessecho Necesario para expandir meta-caracteres, como * y *?*, en los nombres de ficheros en sistemas Unix.

lesskey Usado para especificar los códigos de teclas usados por **less**.

6.27. Groff-1.19.1

El paquete Groff contiene programas para procesar y formatear texto.

Tiempo estimado de construcción: 0.5 SBU
Espacio requerido en disco: 43 MB

La instalación de Groff depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make y Sed

6.27.1. Instalación de Groff

Groff espera que la variable de entorno PAGE contenga el valor por defecto para el tamaño de papel. Para los residentes en Estados Unidos, *PAGE=letter* es un valor adecuado. Para el resto, *PAGE=A4* puede ser más correcto.

Prepara Groff para su compilación:

```
PAGE=[tamaño_del_papel] ./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

Algunos programas de documentación, como **xman**, no funcionarán correctamente sin los siguientes enlaces simbólicos.

```
ln -s soelim /usr/bin/zsoelim
ln -s eqn /usr/bin/geqn
ln -s tbl /usr/bin/gtbl
```

6.27.2. Contenido de Groff

Programas instalados: addftinfo, afmtodit, eqn, eqn2graph, geqn (enlace a eqn), grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (enlace a tbl), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pfbtops, pic, pic2graph, post-grohtml, pre-grohtml, refer, soelim, tbl, tfmtodit, troff y zsoelim (enlace a soelim)

Descripciones cortas

addftinfo Lee un fichero de fuentes troff y añade alguna información adicional sobre la métrica de la fuente, que es usada por el sistema **groff**.

afmtodit Crea un fichero de fuentes para usarlo con **groff** y **grops**.

eqn Compila las descripciones de las fórmulas embebidas en los ficheros de entrada troff a comandos que pueda entender **troff**.

eqn2graph Convierte una ecuación EQN en una imagen.

geqn Enlace a **eqn**

grn Un preprocesador **groff** para ficheros gremlin.

grodvi	Un controlador para **groff** que genera formatos dvi de TeX.
groff	Una interfaz para el sistema de formateado de documentos groff. Normalmente lanza el programa **troff** y un post-procesador apropiado para el dispositivo seleccionado.
groffer	Muestra ficheros groff y páginas de manual en las X y en consola.
grog	Lee ficheros y averigua cuál de las opciones *-e*, *-man*, *-me*, *-mm*, *-ms*, *-p*, *-s* y *-t* de **groff** se necesitan para imprimir los ficheros, y muestra el comando de **groff** incluyendo esas opciones.
grolbp	Un controlador de **groff** para las impresoras Canon CAPSL (series LBP-4 y LBP-8 de impresoras láser)
grolj4	Un controlador para **groff** que produce salidas en el formato PCL5 adecuado para impresoras HP Laserjet 4.
grops	Transforma la salida de GNU **troff** a PostScript.
grotty	Transforma la salida de GNU **troff** en un formato adecuado para dispositivos tipo máquina de escribir.
gtbl	La implementación GNU de **tbl**.
hpftodit	Crea un fichero de fuentes para usar con **groff -Tlj4** a partir de ficheros de marcas de fuentes métricas de HP.
indxbib	Hace un índice inverso para la base de datos bibliográfica, un fichero específico para usarlo con **refer**, **lookbib** y **lkbib**.
lkbib	Busca en las bases de datos bibliográficas referencias que contengan las claves especificadas y muestra cualquier referencia encontrada.
lookbib	Muestra un aviso en la salida de error estándar (excepto si la entrada estándar no es un terminal), lee de la entrada estándar una línea conteniendo un grupo de palabras clave, busca en las bases de datos bibliográficas de un fichero especificado las referencias que contengan dichas claves, muestra cualquier referencia encontrada en la salida estándar y repite el proceso hasta el final de la entrada.
mmroff	Un preprocesador simple para **groff**.
neqn	Formatea ecuaciones para salida ASCII (Código Estándar Americano para Intercambio de Información).
nroff	Un guión que emula al comando **nroff** usando **groff**.
pfbtops	Transforma una fuente en formato `.pfb` de PostScript a ASCII.
pic	Compila descripciones de gráficos embebidos dentro de ficheros de entrada troff o TeX a comandos que puedan ser entendidos por TeX o **troff**.
pic2graph	Convierte un diagrama PIC en una imagen.
post-grohtml	Transforma la salida de GNU **troff** a HTML.
pre-grohtml	Transforma la salida de GNU **troff** a HTML.
refer	Copia el contenido de un fichero en la salida estándar, excepto que las líneas entre *.[* y *.]* son interpretadas como citas, y las líneas entre *.R1* y *.R2* son interpretadas como comandos sobre cómo deben ser procesadas las citas.
soelim	Lee ficheros y reemplaza líneas de la forma *fichero .so* por el contenido de *fichero*.

tbl Compila descripciones de tablas embebidas dentro de ficheros de entrada troff a comandos que puedan ser entendidos por **troff**.

tfmtodit Crea un fichero de fuentes para su uso con **groff -Tdvi**.

troff Es altamente compatible con Unix **troff**. Normalmente debe ser invocado usando el comando **groff**, que también lanzará los preprocesadores y post procesadores en el orden correcto y con las opciones necesarias.

zsoelim La implementación GNU de **soelim**.

6.28. Sed-4.1.2

El paquete Sed contiene un editor de flujos.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 5.2 MB

La instalación de Sed depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Texinfo

6.28.1. Instalación de Sed

Prepara Sed para su compilación:

```
./configure --prefix=/usr --bindir=/bin
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.28.2. Contenido de Sed

Programa instalado: sed

Descripción corta

sed Se usa para filtrar y transformar ficheros de texto en una sola pasada.

6.29. Flex-2.5.31

El paquete Flex contiene una utilidad para generar programas que reconocen patrones de texto.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 3.4 MB

La instalación de Flex depende de: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make y Sed

6.29.1. Instalación de Flex

Flex contiene varios errores conocidos. Corrígelos con el siguiente parche:

```
patch -Np1 -i ../flex-2.5.31-debian_fixes-2.patch
```

Las autotools de GNU detectan que el código fuente de Flex fue modificado por el parche anterior e intentan actualizar la página de manual. Esto no funciona correctamente en muchos sistemas y la página original es correcta, así que asegúrate de que no sea regenerada:

```
touch doc/flex.1
```

Prepara Flex para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Ciertos paquetes esperan encontrar la librería `lex` en el directorio `/usr/lib`. Crea un enlace simbólico para solventar esto:

```
ln -s libfl.a /usr/lib/libl.a
```

Algunos programas aún no conocen **flex** e intentan encontrar a su predecesor **lex**. Para complacer a estos programas, crea un guión envoltorio de nombre `lex` que llame a **flex** en modo de emulación lex:

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Inicio de /usr/bin/lex

exec /usr/bin/flex -l "$@"

# Fin de /usr/bin/lex
EOF
chmod 755 /usr/bin/lex
```

6.29.2. Contenido de Flex

Programas instalados: flex, flex++ (enlace a flex) y lex
Librería instalada: libfl.a

Descripciones cortas

flex	Una herramienta para generar programas capaces de reconocer patrones de texto. Su versatilidad permite establecer las reglas de búsqueda, erradicando la necesidad de desarrollar un programa especializado.
flex++	Invoca una versión de **flex** usada exclusivamente por analizadores C++.
lex	Guión que ejecuta **flex** en el modo de emulación de **lex**.
`libfl.a`	La librería `flex`.

6.30. Gettext-0.14.1

El paquete Gettext contiene utilidades para la internacionalización y localización. Esto permite a los programas compilarse con Soporte de Lenguaje Nativo (NLS), lo que les permite mostrar mensajes en el idioma nativo del usuario.

Tiempo estimado de construcción: 0.5 SBU
Espacio requerido en disco: 55 MB

La instalación de Gettext depende de: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make y Sed

6.30.1. Instalación de Gettext

Prepara Gettext para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**. Esto tarda mucho tiempo, unos 7 SBUs.

Instala el paquete:

```
make install
```

6.30.2. Contenido de Gettext

Programas instalados: autopoint, config.charset, config.rpath, envsubst, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext y xgettext
Librerías instaladas: libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] y libgettextsrc[a,so]

Descripciones cortas

autopoint	Copia los ficheros estándar de infraestructura de Gettext a las fuentes de un paquete.
config.charset	Saca una tabla dependiente del sistema de los alias de codificación de los caracteres.
config.rpath	Saca un grupo de variables dependientes del sistema, describiendo cómo fijar la ruta de búsqueda en tiempo de ejecución de las librerías compartidas en un ejecutable.
envsubst	Sustituye variables de entorno en cadenas del formato del intérprete de comandos.
gettext	Traduce un mensaje en lenguaje natural al lenguaje del usuario, buscando las traducciones en un catálogo de mensajes.
gettextize	Copia todos los ficheros estándar Gettext en el directorio indicado de un paquete, para iniciar su internacionalización
hostname	Muestra el nombre en la red de un sistema en varios formatos.
msgattrib	Filtra los mensajes de un catálogo de traducción de acuerdo con sus atributos, y manipula dichos atributos.
msgcat	Concatena y mezcla los ficheros `.po` indicados.

msgcmp	Compara dos ficheros `.po` para comprobar si ambos contienen el mismo conjunto de cadenas de identificadores de mensajes.
msgcomm	Busca los mensajes que son comunes en los ficheros `.po` indicados.
msgconv	Convierte un catálogo de traducción a una codificación de caracteres diferente.
msgen	Crea un catálogo de traducción en inglés.
msgexec	Aplica un comando a todas las traducciones de un catálogo de traducción.
msgfilter	Aplica un filtro a todas las traducciones de un catálogo de traducción.
msgfmt	Compila el binario de un catálogo de mensajes a partir de un catálogo de traducciones.
msggrep	Extrae todos los mensajes de un catálogo de traducción que cumplan cierto criterio o pertenezcan a alguno de los ficheros fuente indicados.
msginit	Crea un nuevo fichero `.po`, inicializando la información con valores procedentes del entorno del usuario.
msgmerge	Combina dos traducciones directas en un único fichero.
msgunfmt	Descompila catálogos de mensajes binarios en traducciones directas de texto.
msguniq	Unifica las traducciones duplicadas en un catálogo de traducción.
ngettext	Muestra traducciones en lenguaje nativo de un mensaje textual cuya forma gramatical depende de un número.
xgettext	Extrae las líneas de mensajes traducibles de los ficheros fuente indicados, para hacer la primera plantilla de traducción.
`libasprintf`	Define la clase *autosprintf* que hace utilizable la salida formateada de las rutinas de C en programas C++, para usar con las cadenas <*string*> y los flujos <*iostream*>.
`libgettextlib`	Una librería privada que contiene rutinas comunes utilizadas por diversos programas de Gettext. No es indicada para uso general.
`libgettextpo`	Utilizada para escribir programas especializados que procesan ficheros `.po`. Esta librería se utiliza cuando las aplicaciones estándar incluidas con Gettext no son suficiente (como **msgcomm**, **msgcmp**, **msgattrib** y **msgen**).
`libgettextsrc`	Una librería privada que contiene rutinas comunes utilizadas por diversos programas de Gettext. No es indicada para uso general.

6.31. Inetutils-1.4.2

El paquete Inetutils contiene programas para trabajo básico en red.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 11 MB

La instalación de Inetutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses y Sed

6.31.1. Instalación de Inetutils

Inetutils tiene ciertos problemas con los núcleos Linux de la serie 2.6. Corrígelos aplicando el siguiente parche:

```
patch -Np1 -i ../inetutils-1.4.2-kernel_headers-1.patch
```

No vamos a instalar todos los programas que vienen en Inetutils. Sin embargo, el sistema de construcción de Inetutils insistirá en instalar todas las páginas de manual. El siguiente parche corregirá esta situación:

```
patch -Np1 -i ../inetutils-1.4.2-no_server_man_pages-1.patch
```

Prepara Inetutils para su compilación:

```
./configure --prefix=/usr --libexecdir=/usr/sbin \
    --sysconfdir=/etc --localstatedir=/var \
    --disable-logger --disable-syslogd \
    --disable-whois --disable-servers
```

Significado de las opciones de configure:

`--disable-logger`
Esta opción evita que Inetutils instale el programa **logger**, que sirve para que los guiones le pasen mensajes al Demonio de Registro de Eventos del Sistema. Hacemos esto porque luego Util-linux instalará una versión mejor.

`--disable-syslogd`
Esta opción evita que Inetutils instale el Demonio de Registro de Eventos del Sistema, que será instalado con el paquete Sysklogd.

`--disable-whois`
Esta opción desactiva la construcción del cliente **whois** de Inetutils, que está demasiado anticuado. En el libro BLFS hay instrucciones para un cliente **whois** mucho mejor.

`--disable-servers`
Esto desactiva la construcción de los diferentes servidores incluidos como parte del paquete Inetutils. Estos servidores no se consideran apropiados para un sistema LFS básico. Algunos son inseguros por naturaleza y sólo se consideran seguros en redes de confianza. Puedes encontrar más información en *http://www.lfs-es.com/blfs-es-CVS/basicnet/inetutils.html* (el original en inglés se encuentra en *http://www.linuxfromscratch.org/blfs/view/svn/basicnet/inetutils.html*). Ten en cuenta que para muchos de estos servidores hay disponibles sustitutos mejores.

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

Mueve el programa **ping** al lugar indicado por el FHS:

```
mv /usr/bin/ping /bin
```

6.31.2. Contenido de Inetutils

Programas instalados: ftp, ping, rcp, rlogin, rsh, talk, telnet y tftp

Descripciones cortas

ftp	El programa para transferencia de ficheros de ARPANET.
ping	Envía paquetes de petición de eco e informa cuánto tardan las respuestas.
rcp	Copia ficheros de forma remota.
rlogin	Realiza entradas remotas a un sistema.
rsh	Ejecuta un intérprete de comandos remoto.
talk	Permite hablar con otro usuario.
telnet	Una interfaz de usuario para el protocolo TELNET.
tftp	Un programa para la transferencia trivial de ficheros.

6.32. Iproute2-2.6.8-040823

El paquete Iproute2 contiene programas para el trabajo básico y avanzado en redes basadas en IPV4.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 0.6 MB

La instalación de Iproute2 depende de: GCC, Glibc, Make, Linux-Headers y Sed

6.32.1. Instalación de Iproute2

El binario **arpd** incluido en este paquete depende de Berkeley DB. Ya que **arpd** no es un requisito muy común en un sistema Linux básico, eliminamos la dependencia de Berkeley DB aplicando un parche con los comandos siguientes. Si necesitas el binario **arpd**, puedes encontrar las instrucciones para compilar Berkeley DB en el libro BLFS en *http://www.lfs-es.com/blfs-es-CVS/content/databases.html#db* (el original en inglés se encuentra en *http://www.linuxfromscratch.org/blfs/view/svn/content/databases.html#db*).

```
patch -Np1 -i ../iproute2-2.6.8_040823-remove_db-1.patch
```

Prepara Iproute2 para su compilación:

```
./configure
```

Compila el paquete:

```
make SBINDIR=/sbin
```

Significado de la opción de make:

`SBINDIR=/sbin`
Esto asegura que los binarios de Iproute2 se instalarán en `/sbin`. Esta es la localización correcta según el FHS, pues algunos de los binarios de Iproute2 se utilizan en los guiones de arranque.

Instala el paquete:

```
make SBINDIR=/sbin install
```

6.32.2. Contenido de Iproute2

Programas instalados: ifstat, ip, nstat, routef, routel, rtmon, rtstat, ss y tc.

Descripciones cortas

ifstat Muestra las estadísticas de las interfaces, incluida la cantidad de paquetes enviados y recibidos por la interfaz.

ip El ejecutable principal. Tiene diferentes funciones:

ip link *[dispositivo]* permite a los usuarios ver el estado del dispositivo y hacer cambios.

ip addr permite a los usuarios ver las direcciones y sus propiedades, añadir nuevas direcciones y borrar las antiguas.

ip neighbor permite a los usuarios ver los enlaces de vecindad, añadir nuevas entradas de vecindad y borrar las antiguas.

ip rule permite a los usuarios ver las políticas de enrutado y cambiarlas.

ip route permite a los usuarios ver las tablas de enrutado y cambiar las reglas de las tablas.

ip tunnel permite a los usuarios ver los túneles IP y sus propiedades, y cambiarlos.

ip maddr permite a los usuarios ver las direcciones multienlace y sus propiedades, y cambiarlas.

ip mroute permite a los usuarios establecer, cambiar o borrar el enrutado multienlace.

ip monitor permite a los usuarios monitorizar continuamente el estado de los dispositivos, direcciones y rutas.

nstat Muestra las estadísticas de la red.

routef Un componente de **ip route**. Este es para refrescar las tablas de enrutado.

routel Un componente de **ip route**. Este es para listar las tablas de enrutado.

rtmon Utilidad para la monitorización de rutas.

rtstat Utilidad para el estado de rutas.

ss Similar al comando **netstat**. Muestra las conexiones activas.

tc Ejecutable para el control del tráfico. Este es para las implementaciones Quality Of Service (QOS, Calidad de Servicio) y Class Of Service (COS, Clase de Servicio).

tc qdisc permite a los usuarios establecer la disciplina de colas.

tc class permite a los usuarios establecer clases basadas en la planificación de las disciplinas de colas.

tc estimator permite a los usuarios hacer una estimacón del flujo de red en una red.

tc filter permite a los usuarios establecer el filtrado de paquetes QOS/COS.

tc policy permite a los usuarios establecer las políticas QOS/COS.

6.33. Perl-5.8.5

El paquete Perl contiene el Lenguaje Práctico de Extracción e Informe.

Tiempo estimado de construcción: 2.9 SBU
Espacio requerido en disco: 143 MB

La instalación de Perl depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make y Sed

6.33.1. Instalación de Perl

Si quieres un control total sobre la forma en que Perl se configura, puedes ejecutar el guión interactivo **Configure** y modificar a mano el modo en el que se construye este paquete. Si te basta con los valores autodetectados, prepara Perl para su compilación con:

```
./configure.gnu --prefix=/usr -Dpager="/bin/less -isR"
```

Significado de la opción de configure:

`-Dpager="/bin/less -isR"`
Esto corrige un error en el código de **perldoc** con la invocación del programa **less**.

Compila el paquete:

```
make
```

Para ejecutar el banco de pruebas, crea primero un fichero `/etc/hosts` básico, requerido por un grupo de pruebas para resolver el nombre localhost:

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

Ahora, si lo deseas, ejecuta las pruebas:

```
make test
```

Instala el paquete:

```
make install
```

6.33.2. Contenido de Perl

Programas instalados: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.5 (enlace a perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (enlace a s2p), pstruct (enlace a c2ph), s2p, splain y xsubpp
Librerías instaladas: Varios cientos que no podemos listar aquí

Descripciones cortas

a2p Traduce de awk a Perl.

c2ph Vuelca estructuras C similares a las generadas por **cc -g -S**.

dprofpp Muestra datos de perfiles Perl.

en2cxs Construye una extensión Perl para el módulo Encode, a partir de cualquier Mapa de

	Caracteres Unicode o Ficheros de Codificación Tcl.
find2perl	Traduce comandos **find** a código Perl.
h2ph	Convierte ficheros de cabecera `.h` de C en ficheros de cabecera `.ph` de Perl.
h2xs	Convierte ficheros de cabecera `.h` de C en extensiones de Perl.
libnetcfg	Puede usarse para configurar `libnet`.
perl	Combina algunas de las mejores características de C, sed, awk y sh en un único y poderoso lenguaje.
perl5.8.5	Enlace duro a **perl**.
perlbug	Genera informes de errores sobre Perl o sobre los módulos incorporados y los envía por correo.
perlcc	Genera ejecutables a partir de programas Perl.
perldoc	Muestra una parte de la documentación en formato pod que se incluye en el árbol de instalación de Perl o en un guión de Perl.
perlivp	El Procedimiento de Verificación de la Instalación de Perl. Puede usarse para verificar que Perl y sus librerías se han instalado correctamente.
piconv	La versión Perl del convertidor de codificación de caracteres **iconv**.
pl2pm	Es una herramienta que ayuda a convertir ficheros `.pl` de Perl4 en módulos `.pm` de Perl5.
pod2html	Convierte ficheros de formato pod a formato HTML.
pod2latex	Convierte ficheros de formato pod a formato LaTeX.
pod2man	Convierte datos pod en entradas formateadas *roff.
pod2text	Convierte datos pod en texto formateado ASCII.
pod2usage	Muestra mensajes de uso a partir de documentos pod incluidos en ficheros.
podchecker	Comprueba la sintaxis de los ficheros de documentación en formato pod.
podselect	Muestra las secciones elegidas de la documentación pod.
psed	Una versión Perl del editor de flujo **sed**.
pstruct	Vuelca estructuras C similares a las generadas por **cc -g -S**.
s2p	Traduce de sed a Perl.
splain	Se usa para forzar diagnósticos de avisos exhaustivos en Perl.
xsubpp	Convierte el código XS de Perl en código C.

6.34. Texinfo-4.7

El paquete Texinfo contiene programas usados para leer, escribir y convertir documentos Info.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 17 MB

La instalación de Texinfo depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses y Sed

6.34.1. Instalación de Texinfo

El siguiente parche corrige un problema que hace que el programa **info** falle en ocasiones cuando se pulsa la tecla *Delete* del teclado:

```
patch -Np1 -i ../texinfo-4.7-segfault-1.patch
```

Prepara Texinfo para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

Opcionalmente, instala los componentes que pertenecen a una instalación de TeX:

```
make TEXMF=/usr/share/texmf install-tex
```

Significado del parámetro de make:

`TEXMF=/usr/share/texmf`
: La variable `TEXMF` del Makefile fija la ubicación de la raíz del árbol de TeX si, por ejemplo, más adelante se instala un paquete TeX.

El sistema de documentación Info utiliza un fichero de texto plano para almacenar su listado de entradas de menú. Este fichero se encuentra en `/usr/share/info/dir`. Desgraciadamente, debido a problemas ocasionales en los Makefile de diversos paquetes, en ocasiones puede quedarse desfasado con respecto a los manuales Info realmente instalados en el sistema. Si necesitas recrear el fichero `/usr/share/info/dir`, el siguiente comando opcional hará el trabajo:

```
cd /usr/share/info
rm dir
for f in *
do install-info $f dir 2>/dev/null
done
```

6.34.2. Contenido de Texinfo

Programas instalados: info, infokey, install-info, makeinfo, texi2dvi y texindex

Descripciones cortas

info	Lee documentos Info. Los documentos Info son como las páginas de manual, pero tienden a ser más profundos que una simple explicación de las opciones de un programa. Por ejemplo, compara **man tar** con **info tar**.
infokey	Compila un fichero fuente que contiene opciones de Info en un formato binario.
install-info	Se usa para instalar ficheros Info y actualizar las entradas en el fichero índice de Info.
makeinfo	Convierte documentos fuente Texinfo a ficheros Info, texto plano, o HTML.
texi2dvi	Formatea un documento Texinfo, convirtiéndolo en un fichero independiente del dispositivo que puede ser impreso.
texindex	Se usa para ordenar ficheros índice de Texinfo.

6.35. Autoconf-2.59

El paquete Autoconf contiene programas para generar guiones del intérprete de comandos que pueden configurar automáticamente el código fuente.

Tiempo estimado de construcción: 0.5 SBU
Espacio requerido en disco: 7.7 MB

La instalación de Autoconf depende de: Bash, Coreutils, Diffutils, Grep, M4, Make, Perl y Sed

6.35.1. Instalación de Autoconf

Prepara Autoconf para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**. Esto tarda bastante tiempo, unos 2 SBUs

Instala el paquete:

```
make install
```

6.35.2. Contenido de Autoconf

Programas instalados: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate e ifnames

Descripciones cortas

autoconf Genera guiones del intérprete de comandos que automáticamente configuran paquetes de código fuente, adaptándolos a muchas clases de sistemas tipo UNIX. Los guiones de configuración que genera son independientes, para ejecutarlos no es necesario el programa **autoconf**.

autoheader Es una herramienta para crear plantillas de declaraciones #define de C, utilizadas por el guión configure.

autom4te Es un envoltorio para el procesador de macros M4.

autoreconf Ejecuta automáticamente, y en el orden correcto, **autoconf**, **autoheader**, **aclocal**, **automake**, **gettextize** y **libtoolize** para ahorrar tiempo cuando se hacen cambios en las plantillas de **autoconf** y **automake**.

autoscan Ayuda a crear un fichero `configure.in` para un paquete de software. Analiza los ficheros fuente en un árbol de directorios buscando problemas comunes de portabilidad y crea un fichero `configure.scan` que sirve como versión preliminar del fichero `configure.in` para dicho paquete.

autoupdate Modifica un fichero `configure.in` que todavía llame a las macros de **autoconf** por sus antiguos nombres para que utilice los nombre de macro actuales.

ifnames Ayuda a escribir ficheros `configure.in` para un paquete de software. Escribe los identificadores que el paquete usa en condicionales del preprocesador de C. Si un paquete está preparado para tener cierta portabilidad, este programa ayuda a determinar lo que **configure** necesita comprobar. Puede corregir ciertas carencias en un fichero `configure.in` generado por **autoscan**.

6.36. Automake-1.9.1

El paquete Automake contiene programas para generar Makefiles que se utilizan con Autoconf.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 6.8 MB

La instalación de Automake depende de: Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl y Sed

6.36.1. Instalación de Automake

Prepara Automake para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**. Esto tarda bastante tiempo, unos 5 SBUs.

Instala el paquete:

```
make install
```

6.36.2. Contenido de Automake

Programas instalados: acinstall, aclocal, aclocal-1.9.1, automake, automake-1.9.1, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, symlink-tree e ylwrap

Descripciones cortas

acinstall Guión que instala ficheros M4 con estilo aclocal.

aclocal Genera ficheros `aclocal.m4` basados en el contenido de ficheros `configure.in`.

aclocal-1.9.1 Enlace duro a **aclocal**.

automake Herramienta para generar automáticamente los `Makefile.in` a partir de ficheros `Makefile.am`. Para crear todos los ficheros `Makefile.in` para un paquete, ejecuta este programa en el directorio de más alto nivel. Mediante la exploración de los `configure.in` automáticamente encuentra cada `Makefile.am` apropiado y genera el correspondiente `Makefile.in`.

automake-1.9.1 Enlace duro a **automake**.

compile Un envoltorio (wrapper) para compiladores.

config.guess Guión que intenta averiguar el triplete canónico para la construcción, anfitrión o arquitectura destino dada.

config.sub Guión con subrutinas para la validación de configuraciones.

depcomp	Guión para compilar un programa que, aparte de la salida deseada, también genera información sobre las dependencias.
elisp-comp	Compila en octetos código Lisp de Emacs.
install-sh	Guión que instala un programa, guión o fichero de datos.
mdate-sh	Guión que imprime la fecha de modificación de un fichero o directorio.
missing	Guión que actúa como sustituto común de programas GNU no encontrados durante una instalación.
mkinstalldirs	Guión que genera una árbol de directorios.
py-compile	Compila un programa Python.
symlink-tree	Guión para crear un árbol de enlaces simbólicos de un árbol de directorios.
ylwrap	Un envoltorio para **lex** y **yacc**.

6.37. Bash-3.0

El paquete Bash contiene la “Bourne-Again SHell”.

Tiempo estimado de construcción: 1.2 SBU
Espacio requerido en disco: 27 MB

La instalación de Bash depende de: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurse y Sed

6.37.1. Instalación de Bash

El siguiente parche sólo es necesario si no instalaste Readline. Este parche corrige un problema por el que en ocasiones Bash limita a 33 los caractéres de una línea antes de saltar a la siguiente. Si instalaste Readline siguiendo las instrucciones, este parche no es necesario, pues el parche aplicado a Readline ya resuelve este problema:

```
patch -Np1 -i ../bash-3.0-display_wrap-1.patch
```

Prepara Bash para su compilación:

```
./configure --prefix=/usr --bindir=/bin \
    --without-bash-malloc --with-installed-readline
```

Significado de la opción de configure:

`--with-installed-readline`
Esta opción le indica a Bash que utilice la librería `readline` que se encuentra en el sistema, en vez de utilizar su propia versión de Readline.

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make tests`**.

Instala el paquete:

```
make install
```

Lanza el programa **bash** recién compilado (sustituyendo al que estabas ejecutando hasta ahora):

```
exec /bin/bash --login +h
```

Nota

Los parámetros utilizados hacen del proceso **bash** un intérprete interactivo de ingreso y continúa desactivando su tabla interna de rutas para que los nuevos programas sean encontrados a medida que estén disponibles.

6.37.2. Contenido de Bash

Programas instalados: bash, bashbug y sh (enlace a bash)

Descripciones cortas

bash Un intérprete de comandos ampliamente usado. Realiza muchos tipos de expansiones y sustituciones en una línea de comandos dada antes de ejecutarla, lo que hace de este intérprete una herramienta poderosa.

bashbug Un guión que ayuda al usuario en la composición y envío de informes de errores relacionados con **bash**, en un formato estándar.

sh Enlace simbólico al programa **bash**. Cuando se invoca como **sh**, **bash** intenta imitar el comportamiento de las versiones antiguas de **sh** lo mejor posible, mientras que también cumple los estándares POSIX.

6.38. File-4.10

El paquete File contiene una utilidad para determinar el tipo de los ficheros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 6.3 MB

La instalación de File depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed y Zlib

6.38.1. Instalación de File

Prepara File para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

6.38.2. Contenido de File

Programa instalado: file
Librerías instaladas: libmagic.[a,so]

Descripciones cortas

file	Intenta clasificar los ficheros indicados. Lo hace realizando varias pruebas: pruebas de sistemas de ficheros, pruebas de números mágicos y pruebas de lenguajes.
`libmagic`	Contiene rutinas para reconocimiento de números mágicos, usados por el programa **file**.

6.39. Libtool-1.5.8

El paquete Libtool contiene el guión de GNU para soporte genérico de librerías. Oculta la complejidad del uso de librerías compartidas tras una interfaz consistente y portable.

Tiempo estimado de construcción: 1.5 SBU
Espacio requerido en disco: 20 MB

La instalación de Libtool depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make y Sed

6.39.1. Instalación de Libtool

Prepara Libtool para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.39.2. Contenido de Libtool

Programas instalados: libtool y libtoolize
Librerías instaladas: libltdl.[a,so]

Descripciones cortas

libtool	Proporciona servicios de soporte generalizados para la compilación de librerías.
libtoolize	Proporciona una forma estándar de añadir soporte para **libtool** a un paquete.
`libltdl`	Oculta las diversas dificultades para abrir la carga dinámica de las librerías.

6.40. Bzip2-1.0.2

El paquete Bzip2 contiene programas para comprimir y descomprimir ficheros. En ficheros de texto consigue una mejor compresión que el tradicional **gzip**.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 3.0 MB

La instalación de Bzip2 depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc y Make

6.40.1. Instalación de Bzip2

Prepara Bzip2 para su compilación:

```
make -f Makefile-libbz2_so
make clean
```

La opción *-f* provocará que Bzip2 sea construido usando un fichero `Makefile` diferente, en este caso el fichero `Makefile-libbz2_so`, el cual crea una librería dinámica `libbz2.so` y enlaza las utilidades de Bzip2 con ella.

Compila el paquete:

```
make
```

Si reinstalas Bzip2, primero tendrás que hacer un **`rm -f /usr/bin/bz*`**, en caso contrario el siguiente **make install** fallará.

Instala los programas:

```
make install
```

Instala el binario dinámico **bzip2** en el directorio `/bin`, crea algunos enlaces simbólicos necesarios y haz limpieza:

```
cp bzip2-shared /bin/bzip2
cp -a libbz2.so* /lib
ln -s ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm /usr/bin/{bunzip2,bzcat,bzip2}
ln -s bzip2 /bin/bunzip2
ln -s bzip2 /bin/bzcat
```

6.40.2. Contenido de Bzip2

Programas instalados: bunzip2 (enlace a bzip2), bzcat (enlace a bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless y bzmore
Librerías instaladas: libbz2.a, libbz2.so (enlace a libbz2.so.1.0), libbz2.so.1.0 (enlace a libbz2.so.1.0.2) y libbz2.so.1.0.2

Descripciones cortas

bunzip2	Descomprime ficheros que han sido comprimidos con **bzip2**.
bzcat	Descomprime hacia la salida estándar.
bzcmp	Ejecuta **cmp** sobre ficheros comprimidos con **bzip2**.

bzdiff	Ejecuta **diff** sobre ficheros comprimidos con **bzip2**.
bzgrep	Ejecuta **grep** sobre ficheros comprimidos con **bzip2**.
bzegrep	Ejecuta **egrep** sobre ficheros comprimidos con **bzip2**.
bzfgrep	Ejecuta **fgrep** sobre ficheros comprimidos con **bzip2**.
bzip2	Comprime ficheros usando el algoritmo de compresión de texto por ordenación de bloques Burrows-Wheeler con codificación Huffman. La compresión es, en general, considerablemente superior a la obtenida por otros compresores más convencionales basados en el algoritmo “Lempel-Ziv”, como **gzip**.
bzip2recover	Intenta recuperar datos de ficheros comprimidos dañados.
bzless	Ejecuta **less** sobre ficheros comprimidos con **bzip2**.
bzmore	Ejecuta **more** sobre ficheros comprimidos con **bzip2**.
`libbz2`	La librería que implementa la compresión sin pérdidas por ordenación de bloques, usando el algoritmo de Burrows-Wheeler.

6.41. Diffutils-2.8.1

El paquete Diffutils contiene programas que muestran las diferencias entre ficheros o directorios.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 7.5 MB

La instalación de Diffutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

6.41.1. Instalación de Diffutils

Prepara Diffutils para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make install
```

6.41.2. Contenido de Diffutils

Programas instalados: cmp, diff, diff3 y sdiff

Descripciones cortas

cmp Compara dos ficheros e informa en dónde o en qué bytes difieren.

diff Compara dos ficheros o directorios e informa qué líneas de los ficheros difieren.

diff3 Compara tres ficheros línea a línea.

sdiff Mezcla dos ficheros y muestra los resultados interactivamente.

6.42. Kbd-1.12

El paquete Kbd contiene ficheros de mapas de teclado y utilidades para el teclado.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 12 MB

La instalación de Kbd depende de: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, Gzip, M4, Make y Sed

6.42.1. Instalación de Kbd

Prepara Kbd para su compilación:

```
./configure
```

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

6.42.2. Contenido de Kbd

Programas instalados: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, getunimap, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (enlace a psfxtable), psfgettable (enlace a psfxtable), psfstriptable (enlace a psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setlogcons, setmetamode, setvesablank, showconsolefont, showkey, unicode_start y unicode_stop

Descripciones cortas

chvt	Cambia la terminal virtual que aparece en primer plano.
deallocvt	Desasigna las terminales virtuales no usadas.
dumpkeys	Vuelca las tablas de traducción del teclado.
fgconsole	Muestra el número del terminal virtual activo.
getkeycodes	Muestra la tabla de correspondencias de código de exploración (scan code) a código de teclas del núcleo.
getunimap	Muestra el mapa unicode actualmente usado.
kbd_mode	Muestra o establece el modo del teclado.
kbdrate	Establece la repetición y retardo del teclado.
loadkeys	Carga las tablas de traducción del teclado.
loadunimap	Carga la tabla de correspondencia de unicode a fuente del núcleo.
mapscrn	Un programa obsoleto que carga una tabla de correspondencia de caracteres de salida, definida por el usuario, en el controlador de la consola. Esto lo hace ahora **setfont**.

openvt	Comienza un programa en un nuevo terminal virtual (VT).
psf*	Son un grupo de herramientas para manejar tablas de caracteres Unicode para fuentes de consola.
resizecons	Cambia la idea del núcleo sobre el tamaño de la consola.
setfont	Permite cambiar las fuentes EGA y VGA de la consola.
setkeycodes	Carga las entradas de la tabla de correspondencia de código de exploración (scan code) a código de teclas del núcleo. Es últil si el teclado tiene teclas inusuales.
setleds	Establece los LEDs y las opciones del teclado. Mucha gente encuentra útil tener el bloqueo numérico (Num Lock) activado por defecto.
setlogcons	Envía los mensajes del núcleo a la consola.
setmetamode	Define cómo se manejan las teclas meta del teclado.
setvesablank	Permite afinar el salvapantallas incorporado en el hardware (una pantalla en blanco).
showconsolefont	Muestra la fuente de pantalla EGA/VGA actual de la consola.
showkey	Muestra los códigos de exploración, códigos de tecla y códigos ASCII de las teclas presionadas en el teclado.
unicode_start	Pone el teclado y la consola en modo UNICODE. Nunca uses esto en LFS, pues las aplicaciones no están configuradas para soportar UNICODE.
unicode_stop	Revierte el teclado y la consola del modo UNICODE.

6.43. E2fsprogs-1.35

El paquete E2fsprogs contiene las utilidades para manejar el sistema de ficheros `ext2`. También soporta los sistemas de ficheros `ext3` con registro de transacciones.

Tiempo estimado de construcción: 0.6 SBU
Espacio requerido en disco: 4.9 MB

La instalación de E2fsprogs depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed y Texinfo

6.43.1. Instalación de E2fsprogs

Se recomienda construir E2fsprogs en un subdirectorio del árbol de las fuentes:

```
mkdir build
cd build
```

Prepara E2fsprogs para su compilación:

```
../configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs --disable-evms
```

Significado de las opciones de configure:

`--with-root-prefix=""`
Ciertos programas (como el programa **e2fsck**) se consideran esenciales. Cuando, por ejemplo, `/usr` no está montado, estos programas esenciales deben estar disponibles. Pertenecen a directorios como `/lib` y `/sbin`. Si no se le pasase esta opción al configure de E2fsprogs, los programas se instalarían en el directorio `/usr`, que no es donde deberían estar.

`--enable-elf-shlibs`
Esto crea las librerías compartidas utilizadas por algunos de los programas de este paquete.

`--disable-evms`
Esto desactiva la construcción del módulo para el Enterprise Volume Management System (EVMS, Sistema Empresarial de Manejo de Volúmenes). Este módulo no está actualizado a la última intefaz interna de EVMS, y EVMS no se instala como parte del sistema base LFS. Para más información mira la página web de EVMS en *http://evms.sourceforge.net/*.

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala la mayor parte del paquete:

```
make install
```

Instala las librerías compartidas:

```
make install-libs
```

6.43.2. Contenido de E2fsprogs

Programas instalados: badblocks, blkid, chattr, compile_et, debugfs, dumpe2fs, e2fsck, e2image, e2label, findfs, fsck, fsck.ext2, fsck.ext3, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mklost+found, resize2fs, tune2fs y uuidgen.
Librerías instaladas: libblkid.[a,so], libcom_err.[a,so], libe2p.[a,so], libext2fs.[a,so], libss.[a,so] y libuuid.[a,so]

Descripciones cortas

badblocks Busca bloques dañados en un dispositivo (normalmente una partición de disco).

blkid Una utilidad de línea de comandos para localizar y mostrar atributos de dispositivos de bloque.

chattr Cambia los atributos de los ficheros en un sistema de ficheros `ext2` y también en sistemas de ficheros `ext3`, la versión con registro de transacciones del sistema de ficheros `ext2`.

compile_et Un compilador de tablas de error. Convierte una tabla de códigos de error y mensajes en un fichero fuente C apropiado para usar con la librería `com_err`.

debugfs Un depurador de sistemas de ficheros. Puede usarse para examinar y cambiar el estado de un sistema de ficheros `ext2`.

dumpe2fs Muestra la información del superbloque y de los grupos de bloques del sistema de ficheros presente en un determinado dispositivo.

e2fsck Se usa para chequear, y opcionalmente reparar, sistemas de ficheros `ext2` y también `ext3`.

e2image Se usa para salvar información crítica de un sistema de ficheros `ext2` en un fichero.

e2label Muestra o cambia la etiqueta de un sistema de ficheros `ext2` situado en el dispositivo especificado.

findfs Encuentra un sistema de ficheros por su etiqueta o UUID (Identificador Universal Único).

fsck Se usa para chequear, y opcionalmente reparar, un sistema de ficheros. Por defecto comprueba los sistemas de ficheros listados en `/etc/fstab`.

fsck.ext2 Por defecto comprueba sistema de ficheros `ext2`.

fsck.ext3 Por defecto comprueba sistemas de ficheros `ext3`.

logsave Salva la salida de un comando en un fichero de registro.

lsattr Muestra los atributos de un fichero en un sistema de ficheros `ext2`.

mk_cmds Convierte una tabla de nombres de comandos y mensajes de ayuda en un fichero fuente C preparado para usarlo con la librería del subsistema `libss`.

mke2fs Se usa para crear sistemas de ficheros `ext2` en un dispositivo dado.

mkfs.ext2 Por defecto crea un sistema de ficheros `ext2`.

mkfs.ext3 Por defecto crea un sistema de ficheros `ext3`.

mklost+found	Se usa para crear un directorio `lost+found` en un sistema de ficheros `ext2`. Reserva bloques de disco para este directorio facilitando la tarea de **e2fsck**.
resize2fs	Se usa para redimensionar sistemas de ficheros `ext2`.
tune2fs	Ajusta los parámetros de un sistema de ficheros `ext2`.
uuidgen	Crea un nuevo UUID. Cada nuevo UUID puede considerarse razonablemente único por muchos UUID que se hayan creado en el sistema local o en otros sistemas en el pasado o en el futuro.
`libblkid`	Contiene rutinas para la identificación de dispositivos y extracción de marcas.
`libcom_err`	Rutina para mostrar errores comunes.
`libe2p`	Usada por **dumpe2fs**, **chattr** y **lsattr**.
`libext2fs`	Contiene rutinas para permitir a los programas de nivel de usuario manipular un sistema de ficheros `ext2`.
`libss`	Usada por **debugfs**.
`libuuid`	Contiene rutinas para generar identificadores únicos para objetos que pueden estar accesibles más allá del sistema local.

6.44. Grep-2.5.1

El paquete Grep contiene programas para buscar dentro de ficheros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 5.8 MB

La instalación de Grep depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed y Texinfo

6.44.1. Instalación de Grep

Prepara Grep para su compilación:

```
./configure --prefix=/usr --bindir=/bin --with-included-regex
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.44.2. Contenido de Grep

Programas instalados: egrep (enlace a grep), fgrep (enlace a grep) y grep

Descripciones cortas

egrep Muestra las líneas que coincidan con una expresión regular extendida.

fgrep Muestra las líneas que coincidan con una lista de cadenas fijas.

grep Muestra las líneas que coincidan con una expresión regular básica.

6.45. Grub-0.95

El paquete Grub contiene el GRand Unified Bootloader (Gran Gestor de Arranque Unificado).

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 10 MB

La instalación de Grub depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses y Sed

6.45.1. Instalación de Grub

Se sabe que este programa se comporta mal si se cambian sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Si tienes definida cualquier variable de entorno que altere las optimizaciones por defecto, como CFLAGS o CXXFLAGS, desactívala cuando construyas Grub.

Prepara Grub para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **make check**.

Advierte que los resultados de las pruebas mostrarán siempre el error “ufs2_stage1_5 is too big”. Se debe a un problema del compilador, pero puede ignorarse a menos que pienses arrancar desde una partición UFS. Dichas particiones normalmente sólo se utilizan en estaciones de trabajo Sun.

Instala el paquete:

```
make install
mkdir /boot/grub
cp /usr/share/grub/i386-pc/stage{1,2} /boot/grub
```

Sustituye i386-pc por el directorio apropiado para tu hardware.

El directorio i386-pc contiene también una serie de ficheros *stage1_5 para diferentes sistemas de ficheros. Mira los disponibles y copia el apropiado al directorio /boot/grub. La mayoría copiareis el fichero e2fs_stage1_5 y/o reiserfs_stage1_5.

6.45.2. Contenido de Grub

Programas instalados: grub, grub-install, grub-md5-crypt, grub-terminfo y mbchk

Descripciones cortas

grub	El intérprete de comandos del GRand Unified Bootloader (Gran Gestor de Arranque Unificado).
grub-install	Instala GRUB en el dispositivo indicado.
grub-md5-crypt	Encripta una contraseña en formato MD5.

grub-terminfo	Genera un comando terminfo a partir de un nombre terminfo. Puede utilizarse si tienes un terminal poco común.
mbchk	Comprueba el formato de un núcleo multiarranque.

6.46. Gzip-1.3.5

El paquete Gzip contiene programas para comprimir y descomprimir ficheros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 2.6 MB

La instalación de Gzip depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make y Sed

6.46.1. Instalación de Gzip

Prepara Gzip para su compilación:

```
./configure --prefix=/usr
```

El guión **gzexe** guarda en su código la localización del binario **gzip**. Como luego vamos a cambiar la ubicación del binario, el siguiente comando asegura que la nueva ubicación se guarde dentro del guión.

```
sed -i 's@"BINDIR"@/bin@g' gzexe.in
```

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

Mueve los programas al directorio `/bin`:

```
mv /usr/bin/gzip /bin
rm /usr/bin/{gunzip,zcat}
ln -s gzip /bin/gunzip
ln -s gzip /bin/zcat
ln -s gunzip /bin/uncompress
```

6.46.2. Contenido de Gzip

Programas instalados: gunzip (enlace a gzip), gzexe, gzip, uncompress (enlace a gunzip), zcat (enlace a gzip), zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore y znew

Descripciones cortas

gunzip	Descomprime ficheros que hayan sido comprimidos con **gzip**.
gzexe	Crea ficheros ejecutables autodescomprimibles.
gzip	Comprime los ficheros indicados usando codificación Lempel-Ziv (LZ77).
uncompress	Descomprime ficheros comprimidos.
zcat	Descomprime en la salida estándar los ficheros indicados comprimidos con **gzip**.
zcmp	Ejecuta **cmp** sobre ficheros comprimidos.
zdiff	Ejecuta **diff** sobre ficheros comprimidos.
zegrep	Ejecuta **egrep** sobre ficheros comprimidos.

zfgrep Ejecuta **fgrep** sobre ficheros comprimidos.

zforce Fuerza la extensión `.gz` en todos los ficheros comprimidos para que **gzip** no los comprima dos veces. Esto puede ser útil para ficheros con el nombre truncado después de una transferencia de ficheros.

zgrep Ejecuta **grep** sobre ficheros comprimidos.

zless Ejecuta **less** sobre ficheros comprimidos.

zmore Ejecuta **more** sobre ficheros comprimidos.

znew Recomprime ficheros del formato de **compress** al formato de **gzip**, o sea, de `.Z` a `.gz`.

6.47. Man-1.5o

El paquete Man contiene programas para encontrar y visualizar páginas de manual.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 1.9MB

La instalación de Man depende de: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make y Sed

6.47.1. Instalación de Man

Haremos tres ajustes a las fuentes de Man.

El primero es un parche que permite que Man funcione mejor con las versiones recientes de Groff. En concreto, las páginas de manual se mostrarán ahora usando el ancho total del terminal, en vez de estar limitado a 80 caracteres:

```
patch -Np1 -i ../man-1.5o-80cols-1.patch
```

El segundo es una sustitución sed para añadir el modificador `-R` a la variable `PAGER` para que las secuencias de escape puedan ser manejadas correctamente por Less:

```
sed -i 's@-is@&R@g' configure
```

El tercero es también una sustitución sed para comentar la línea “MANPATH /usr/man” del fichero `man.conf` y prevenir resultados duplicados al usar programas como **whatis**:

```
sed -i 's@MANPATH./usr/man@#&@g' src/man.conf.in
```

Prepara Man para su compilación:

```
./configure -confdir=/etc
```

Significado de la opción de configure:

`-confdir=/etc`
: Esto le indica al programa **man** que busque el fichero de configuración `man.conf` en el directorio `/etc`.

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

Nota

Para desactivar las secuencias de escape SGR, edita el fichero `man.conf` y añade el argumento `-c` a la variable de entorno `NROFF`.

Si el conjunto de caractéres utilizados es de 8 bits, busca la línea que comienza con “NROFF” en `/etc/man.conf` y verifica que es similar a esta:

```
NROFF  /usr/bin/nroff -Tlatin1 -mandoc
```

Advierte que “latin1” debe usarse aunque no sea el conjunto de caracteres de tu locale. El motivo es que, según la especificación, **groff** no sabe cómo escribir caracteres ajenos al ISO 8859-1 sin ciertos códigos de escape extraños. Cuando se formatean páginas de manual, **groff** piensa que están codificadas en ISO 8859-1 y la opción *`-Tlatin1`* le dice a **groff** que utilice la mísma codificación para la salida. Puesto que **groff** no recodifica los caractéres de entrada, el formateado resultante tiene en realidad la misma codificación que la entrada, y por tanto es usable como entrada para un visualizador.

Esto no soluciona el problema de que el programa **man2dvi** no funciona con las páginas de manual traducidas que no estén en ISO 8859-1. Igualmente, no funciona con conjuntos de caracteres multibyte. El primer problema no tiene solución por ahora. El segundo no nos afecta pues la instalación de LFS no incluye soporte para conjuntos de caracteres multibyte.

Información adicional sobre la compresión de páginas de manual e info se puede encontrar en el libro BLFS en *http://www.lfs-es.com/blfs-es-CVS/postlfs/compressdoc.html* (el original en inglés se encuentra en *http://www.linuxfromscratch.org/blfs/view/stable/postlfs/compressdoc.html*).

6.47.2. Contenido de Man

Programas instalados: apropos, makewhatis, man, man2dvi, man2html y whatis

Descripciones cortas

apropos Busca una cadena en la base de datos whatis y muestra las descripciones cortas de los comandos del sistema que contengan dicha cadena.

makewhatis Construye la base de datos whatis. Lee todas las páginas de manual encontradas en las rutas "manpath" y por cada página escribe el nombre de la página y una descripción corta en la base de datos whatis.

man Formatea y muestra las páginas de manual.

man2dvi Convierte una página de manual a formato dvi.

man2html Convierte una página de manual a formato HTML.

whatis Busca palabras clave en la base de datos whatis y muestra las descripciones cortas de los comandos del sistema que contengan la palabra clave dada como una palabra completa.

6.48. Make-3.80

El paquete Make contiene un programa para compilar paquetes grandes.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 8.8 MB

La instalación de Make depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep y Sed

6.48.1. Instalación de Make

Prepara Make para su compilación:

```
./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.48.2. Contenido de Make

Programa instalado: make

Descripción corta

make Determina automáticamente qué partes de un paquete grande necesitan ser recompiladas y lanza los comandos para hacerlo.

6.49. Module-Init-Tools-3.0

El paquete Module-Init-Tools contiene programas para manejar módulos del núcleo en núcleos Linux con versión mayor o igual a 2.5.47.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 650 KB

La instalación de Module-Init-Tools depende de: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Glibc, Grep, M4, Make y Sed

6.49.1. Instalación de Module-Init-Tools

Prepara Module-Init-Tools para su compilación:

```
./configure --prefix="" --enable-zlib
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.49.2. Contenido de Module-Init-Tools

Programas instalados: depmod, genksyms, insmod, insmod_ksymoops_clean, kallsyms (enlace a insmod), kernelversion, ksyms (enlace a insmod), lsmod (enlace a insmod), modinfo, modprobe (enlace a insmod) y rmmod (enlace a insmod)

Descripciones cortas

depmod	Crea un fichero de dependencias basándose en los símbolos que encuentra en el conjunto existente de módulos del núcleo. A este fichero lo usa **modprobe** para cargar automáticamente los módulos necesarios.
genksyms	Genera información sobre la versión de los símbolos.
insmod	Instala un módulo dentro del núcleo en ejecución.
insmod_ksymoops_clean	Borra los símbolos del núcleo (ksyms) guardados y los módulos a los que no se ha accedido en los últimos 2 días.
kallsyms	Extrae todos los símbolos del núcleo para la depuración.
kernelversion	Informa sobre la versión mayor del núcleo en ejecución.
ksyms	Muestra los símbolos exportados del núcleo.
lsmod	Muestra todos los módulos cargados.
modinfo	Examina un fichero objeto asociado con un módulo del núcleo y muestra la información que pueda encontrar.
modprobe	Usa un fichero de dependencias, creado por **depmod**, para cargar

	automáticamente los módulos necesarios.
rmmod	Descarga módulos del núcleo en ejecución.

6.50. Patch-2.5.4

El paquete Patch contiene un programa para modificar ficheros.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 1.9 MB

La instalación de Patch depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make y Sed

6.50.1. Instalación de Patch

Prepara Patch para su compilación. La opción del preprocesador *-D_GNU_SOURCE* sólo es necesaria en la plataforma PowerPC, en otras plataformas puedes ignorarla:

```
CPPFLAGS=-D_GNU_SOURCE ./configure --prefix=/usr
```

Compila el paquete:

```
make
```

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make install
```

6.50.2. Contenido de Patch

Programa instalado: patch

Descripción corta

patch Modifica ficheros según lo indicado en un fichero parche. Normalmente un parche es una lista de diferencias creada por el programa **diff**. Al aplicar estas diferencias a los ficheros originales, **patch** crea las versiones parcheadas.

6.51. Procps-3.2.3

El paquete Procps contiene programas para monitorizar procesos.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 6.2 MB

La instalación de Procps depende de: Bash, Binutils, Coreutils, GCC, Glibc, Make y Ncurses

6.51.1. Instalación de Procps

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

6.51.2. Contenido de Procps

Programas instalados: free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w y watch
Librería instalada: libproc.so

Descripciones cortas

free	Muestra la cantidad total de memoria libre y usada en el sistema, tanto física como de intercambio (swap).
kill	Envía señales a los procesos.
pgrep	Visualiza procesos basándose en su nombre u otros atributos
pkill	Envía señales a procesos basándose en su nombre u otros atributos
pmap	Muestra el mapa de memoria del proceso indicado.
ps	Facilita una instantánea de los procesos actuales.
skill	Envía señales a procesos que coincidan con un criterio dado.
snice	Cambia la prioridad de planificación de los procesos que coincidan con un criterio dado.
sysctl	Modifica los parámetros del núcleo en tiempo de ejecución.
tload	Imprime un gráfico de la carga promedio actual del sistema.
top	Muestra los procesos más activos en CPU. Proporciona una vista dinámica de la actividad del procesador en tiempo real.
uptime	Muestra cuánto tiempo hace que el sistema está en ejecución, cuántos usuarios están conectados y la carga media del sistema.
vmstat	Muestra estadísticas de la memoria virtual, dando información sobre los procesos, memoria, paginación, entrada/salida por bloques y actividad del procesador.
w	Muestra qué usuarios hay actualmente en el sistema, en qué terminal y desde cuándo.

watch Ejecuta un comando repetidamente, mostrando su primera salida a pantalla completa. Esto te permite observar los cambios en la salida al pasar el tiempo.

`libproc` Contiene funciones usadas por la mayoría de los programas de este paquete.

6.52. Psmisc-21.5

El paquete Psmisc contiene programas para mostrar información sobre procesos.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 2.2 MB

La instalación de Psmisc depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses y Sed

6.52.1. Instalación de Psmisc

Prepara Psmisc para su compilación:

```
./configure --prefix=/usr --exec-prefix=""
```

Significado de la opción de configure:

`--exec-prefix=""`
Esto hace que los binarios se instalen en `/bin` en lugar de `/usr/bin`. Como los programas de Psmisc se usan a menudo en los guiones de inicio, deben estar también disponibles cuando la partición `/usr` no esté montada.

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

No hay razón para que los programas **pstree** y **pstree.x11** residan en `/bin`. Por tanto los moveremos a `/usr/bin`. Igualmente, no es necesario que **pstree.x11** sea un programa independiente, así que lo convertiremos en un enlace simbólico a **pstree**:

```
mv /bin/pstree* /usr/bin
ln -sf pstree /usr/bin/pstree.x11
```

El programa **pidof** de Psmisc no se instala por defecto. Normalmente esto no es ningún problema, ya que más tarde instalaremos el paquete Sysvinit, el cual nos facilita una versión mejor del programa **pidof**. Pero si no vas a usar Sysvinit, debes completar la instalación de Psmisc creando el siguiente enlace simbólico:

```
ln -s killall /bin/pidof
```

6.52.2. Contenido de Psmisc

Programas instalados: fuser, killall, pstree y pstree.x11 (enlace a pstree)

Descripciones cortas

fuser Muestra los números de identificación (PID) de los procesos que usan los ficheros o sistemas de ficheros especificados.

killall	Mata procesos por su nombre. Envía una señal a todos los procesos que ejecutan alguno de los comandos especificados.
pstree	Muestra los procesos en ejecución en forma de árbol.
pstree.x11	Es igual que **pstree** excepto que espera confirmación antes de salir.

6.53. Shadow-4.0.4.1

El paquete Shadow contiene programas para manejar contraseñas de forma segura.

Tiempo estimado de construcción: 0.4 SBU
Espacio requerido en disco: 11 MB

La instalación de Shadow depende de: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

6.53.1. Instalación de Shadow

Prepara Shadow para su compilación:

```
./configure --libdir=/usr/lib --enable-shared
```

Evita un problema que impide que funcione la internacionalización en Shadow:

```
echo '#define HAVE_SETLOCALE 1' >> config.h
```

Shadow declara incorrectamente la función malloc(), provocando que falle la compilación. Corrígelo:

```
sed -i '/extern char/d' libmisc/xmalloc.c
```

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

Shadow utiliza dos ficheros para configurar los ajustes de autenticación para el sistema. Instala estos ficheros de configuración:

```
cp etc/{limits,login.access} /etc
```

En vez de usar el método por defecto, *crypt*, utiliza el método de encriptación de contraseñas *MD5*, que es más seguro y además permite contraseñas de más de 8 caracteres. También es necesario cambiar la obsoleta localización `/var/spool/mail`, que Shadow utiliza por defecto para los buzones de los usuarios, a `/var/mail`, que es la localización usada hoy en día. Ambas cosas pueden hacerse modificando el fichero de configuración correspondiente mientras lo copiamos a su destino:

```
cp etc/login.defs.linux /etc/login.defs
sed -i -e 's@#MD5_CRYPT_ENAB.no@MD5_CRYPT_ENAB yes@' \
    -e 's@/var/spool/mail@/var/mail@' /etc/login.defs
```

Mueve algunos enlaces mal ubicados a sus lugares correctos:

```
mv /bin/sg /usr/bin
mv /bin/vigr /usr/sbin
mv /usr/bin/passwd /bin
```

Mueve las librerías dinámicas de Shadow a un lugar más apropiado:

```
mv /usr/lib/lib{shadow,misc}.so.0* /lib
```

Como algunos paquetes esperan encontrar las librerías que acabamos de mover en `/usr/lib`, crea los siguientes enlaces simbólicos:

```
ln -sf ../../lib/libshadow.so.0 /usr/lib/libshadow.so
ln -sf ../../lib/libmisc.so.0 /usr/lib/libmisc.so
```

La opción *-D* del programa **useradd** requiere el directorio `/etc/default` para funcionar correctamente:

```
mkdir /etc/default
```

Coreutils ya ha instalado un programa **groups** mejor en `/usr/bin`. Borra el instalado por Shadow:

```
rm /bin/groups
```

6.53.2. Configuración de Shadow

Este paquete contiene utilidades para añadir, modificar o eliminar usuarios y grupos, establecer y cambiar sus contraseñas y otras tareas administrativas. Puedes encontrar una completa explicación de lo que significa *password shadowing* (ocultación de contraseñas) en el fichero `doc/HOWTO` dentro del árbol de las fuentes. Hay una cosa que debes recordar si decides usar soporte para Shadow: los programas que necesiten verificar contraseñas (administradores de sesión, programas FTP, demonios pop3, etc) necesitarán ser compatibles con shadow: esto es, necesitan ser capaces de trabajar con contraseñas ocultas.

Para habilitar las contraseñas ocultas, ejecuta el siguiente comando:

```
pwconv
```

Para habilitar las contraseñas de grupo ocultas, ejecuta:

```
grpconv
```

Bajo circunstancias normales aún no habrás creado ninguna contraseña. Sin embargo, si más tarde regresas a esta sección para activar la ocultación, debes restablecer cualquier contraseña actual de usuario con el comando **passwd**, o cualquier contraseña de grupo con el comando **gpasswd**.

6.53.3. Establecer la contraseña de root

Elige una contraseña para el usuario *root* y establécela mediante:

```
passwd root
```

6.53.4. Contenido de Shadow

Programas instalados: chage, chfn, chpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, groups, grpck, grpconv, grpunconv, lastlog, login, logoutd, mkpasswd, newgrp, newusers, passwd, pwck, pwconv, pwunconv, sg (enlace a newgrp), useradd, userdel, usermod, vigr (enlace a vipw) y vipw
Librerías instaladas: libshadow[.a,so]

Descripciones cortas

chage Se usa para cambiar el número máximo de días entre cambios obligatorios de contraseña.

chfn Se usa para cambiar el nombre completo de un usuario y otra información.

chpasswd Se usa para actualizar las contraseñas de un grupo de cuentas de usuario de una sola vez.

chsh	Cambia el intérprete de comandos por defecto que se ejecuta cuando el usuario entra al sistema.
expiry	Comprueba y refuerza la política actual de expiración de contraseñas.
faillog	Sirve para examinar el contenido del registro de ingresos fallidos al sistema, establecer un máximo de fallos para bloquear una cuenta de usuario y reiniciar el contador de fallos.
gpasswd	Se usa para agregar y eliminar miembros y administradores a los grupos.
groupadd	Crea un nuevo grupo con el nombre especificado.
groupdel	Borra el grupo con el nombre especificado.
groupmod	Modifica el nombre o el identificador (GID) de un grupo especificado.
groups	Muestra los grupos a los que pertenece un usuario dado.
grpck	Verifica la integridad de los ficheros de grupos, `/etc/group` y `/etc/gshadow`.
grpconv	Crea o actualiza el fichero de grupos ocultos a partir de un fichero de grupos normal.
grpunconv	Actualiza `/etc/group` a partir de `/etc/gshadow`, borrando este último.
lastlog	Muestra el último acceso de cada usuario o de un usuario especificado.
login	Lo utiliza el sistema para permitir el ingreso de un usuario.
logoutd	Es un demonio que refuerza las restricciones de ingreso en base a horas y puertos de acceso.
mkpasswd	Genera contraseñas aleatorias.
newgrp	Se usa para cambiar el identificador de grupo (GID) actual durante una sesión de acceso.
newusers	Crea o actualiza un grupo de cuentas de usuario de una sola vez.
passwd	Se utiliza para cambiar la contraseña de la cuenta de un usuario o grupo.
pwck	Verifica la integridad de los ficheros de contraseñas, `/etc/passwd` y `/etc/shadow`.
pwconv	Crea o actualiza el fichero de contraseñas ocultas a partir de un fichero de contraseñas normal.
pwunconv	Actualiza `/etc/passwd` a partir de `/etc/shadow`, borrando este último.
sg	Ejecuta un comando dado estableciendo el GID del usuario al del grupo indicado.
su	Ejecuta un intérprete de comandos sustituyendo los identificadores de usuario y grupo.
useradd	Crea un nuevo usuario con el nombre especificado o actualiza la información por defecto de un nuevo usuario.
userdel	Borra la cuenta de usuario indicada.
usermod	Modifica el nombre, identificador (UID), intérprete de comandos, grupo inicial, directorio personal, etc, del usuario indicado.
vigr	Edita los ficheros `/etc/group` o `/etc/gshadow`.
vipw	Edita los ficheros `/etc/passwd` o `/etc/shadow`.
`libshadow`	Contiene funciones usadas por la mayoría de los programas de este paquete.

6.54. Sysklogd-1.4.1

El paquete Sysklogd contiene programas para registrar los mensajes del sistema, como aquellos generados por el núcleo cuando sucede algo inusual.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 0.5 MB

La instalación de Sysklogd depende de: Binutils, Coreutils, GCC, Glibc y Make

6.54.1. Instalación de Sysklogd

Sysklogd tiene problemas con los núcleos Linux de la serie 2.6. Corrígelos aplicando el siguiente parche:

```
patch -Np1 -i ../sysklogd-1.4.1-kernel_headers-1.patch
```

También hay un comportamiento extraño en la lógica de manejo de señales que en ocasiones confunde al guión de inicio **sysklogd**. Corrige este error aplicando otro parche:

```
patch -Np1 -i ../sysklogd-1.4.1-signal-1.patch
```

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

6.54.2. Configuración de Sysklogd

Crea un nuevo fichero `/etc/syslog.conf` ejecutando lo siguiente:

```
cat > /etc/syslog.conf << "EOF"
# Inicio de /etc/syslog.conf

auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *

# Fin de /etc/syslog.conf
EOF
```

6.54.3. Contenido de Sysklogd

Programas instalados: klogd y syslogd

Descripciones cortas

klogd Un demonio del sistema que intercepta y registra los mensajes del núcleo.

syslogd Registra los mensajes que los programas del sistema ofrecen.

6.55. Sysvinit-2.85

El paquete Sysvinit contiene programas para controlar el arranque, ejecución y cierre del sistema.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 0.9 MB

La instalación de Sysvinit depende de: Binutils, Coreutils, GCC, Glibc y Make

6.55.1. Instalación de Sysvinit

Sysvinit-2.85 tiene un error de “buffer overflow” (desbordamiento de pila). Bajo ciertas condiciones, modifica los valores de las variables de entorno. Corrígelo con:

```
patch -Np1 -i ../sysvinit-2.85-proclen-1.patch
```

Cuando se cambia de nivel de ejecución (por ejemplo cuando apagamos el sistema) el programa **init** envía las señales de finalización a aquellos procesos que él mismo inició y que no deben estar en ejecución en el nuevo nivel. Mientras lo hace, **init** muestra mensajes del tipo “Sending processes the TERM signal” (Enviando la señal TERM a los procesos), que parece indicar que se está enviando dicha señal a todos los procesos que hay en ejecución. Para evitar esta confusión, puedes modificar las fuentes para que ese mensaje diga en su lugar “Sending processes started by init the TERM signal” (Enviando la señal TERM a los procesos iniciados por init):

```
sed -i 's@Sending processes@& started by init@g' \
    src/init.c
```

Compila el paquete:

```
make -C src
```

Instala el paquete:

```
make -C src install
```

6.55.2. Configuración de Sysvinit

Crea un nuevo fichero `/etc/inittab` ejecutando lo siguiente:

```
cat > /etc/inittab << "EOF"
# Inicio de /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty -I '\033(K' tty1 9600
2:2345:respawn:/sbin/agetty -I '\033(K' tty2 9600
3:2345:respawn:/sbin/agetty -I '\033(K' tty3 9600
4:2345:respawn:/sbin/agetty -I '\033(K' tty4 9600
5:2345:respawn:/sbin/agetty -I '\033(K' tty5 9600
6:2345:respawn:/sbin/agetty -I '\033(K' tty6 9600

# Fin de /etc/inittab
EOF
```

La opción *`-I '\033(K'`* le indica a **agetty** que envíe al terminal esta secuencia de escape antes de hacer nada más. Esta secuencia de escape cambia el conjunto de caractéres de la consola a uno definido por el usuario, que puede modificarse ejecutando el programa **setfont**. El guión de inicio **console** incluido en el paquete LFS-Bootscripts llama al programa **setfont** durante el arranque del sistema. Enviar esta secuencia de escape es necesario para las personas que utilizan fuentes de pantalla que no son ISO 8859-1, pero no afecta a los anglo-parlantes.

6.55.3. Contenido de Sysvinit

Programas instalados: halt, init, killall5, last, lastb (enlace a last), mesg, pidof (enlace a killall5), poweroff (enlace a halt), reboot (enlace a halt), runlevel, shutdown, sulogin, telinit (enlace a init), utmpdump y wall

Descripciones cortas

halt Suele invocar a **shutdown** con la opción *-h*, excepto cuando el sistema ya se encuentra en el nivel de ejecución 0, en cuyo caso le indica al núcleo que apague el sistema. Anota en `/var/log/wtmp` que el sistema se va a cerrar.

init El primer proceso que se inicia cuando el núcleo ha inicializado el hardware, el cual toma el control sobre el arranque e inicia todos los procesos que se le han indicado.

killall5 Envía una señal a todos los procesos, excepto a los procesos de su propia sesión para que no mate el intérprete de comandos desde el que fue llamado.

last Muestra los últimos usuarios conectados (y desconectados), buscando hacia atrás en el fichero `/var/log/wtmp`. También muestra los inicios y paradas del sistema y los cambios de nivel de ejecución.

lastb Muestra los intentos fallidos de acceso al sistema, que se registran en `/var/log/btmp`.

mesg Controla si otros usuarios pueden o no enviar mensajes al terminal del usuario actual.

pidof Muestra los identificadores de proceso (PIDs) de los programas especificados.

poweroff Le indica al núcleo que cierre el sistema y apague la máquina (ver **halt**).

reboot Le indica al núcleo que reinicie el sistema (ver **halt**).

runlevel Muestra los niveles de ejecución anterior y actual tal y como figura en el último registro de nivel de ejecución de `/var/run/utmp`.

shutdown Provoca el cierre del sistema de una forma segura, enviando señales a todos los procesos y notificando a todos los usuarios conectados.

sulogin Permite el ingreso de *root* al sistema. Suele ser invocado por **init** cuando el sistema entra en el modo monousuario.

telinit Le indica a **init** a qué nivel de ejecución debe cambiar.

utmpdump Muestra el contenido de un fichero de registro de accesos dado en un formato comprensible por el usuario.

wall Envía un mensaje a todos los usuarios conectados.

6.56. Tar-1.14

El paquete Tar contiene un programa de archivado.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 10 MB

La instalación de Tar depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make y Sed

6.56.1. Instalación de Tar

Prepara Tar para su compilación:

```
./configure --prefix=/usr --bindir=/bin --libexecdir=/usr/sbin
```

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make check`**.

Instala el paquete:

```
make install
```

6.56.2. Contenido de Tar

Programas instalados: rmt y tar

Descripciones cortas

rmt Manipula remotamente una unidad de cinta magnética mediante una comunicación de conexión entre procesos.

tar Crea y extrae ficheros de un archivo, también conocido como paquete tar (tarball).

6.57. Udev-030

El paquete Udev contiene programas para la creación dinámica de nodos de dispositivos.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 5.2 MB

La instalación de Udev depende de: Coreutils y Make

6.57.1. Instalación de Udev

Compila el paquete:

```
make udevdir=/dev
```

udevdir=/dev
Esto le indica a **udev** en qué directorio se deben crear los nodos de dispositivos.

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make udevdir=/dev install
```

La configuración por defecto de Udev no es la ideal, así que instala aquí los ficheros de configuración:

```
cp ../udev-config-2.permissions \
    /etc/udev/permissions.d/25-lfs.permissions
cp ../udev-config-1.rules /etc/udev/rules.d/25-lfs.rules
```

6.57.2. Contenido de Udev

Programas instalados: udev, udevd, udevsend, udevstart, udevinfo y udevtest
Directorio instalado: /etc/udev

Descripciones cortas

udev Crea nodos de dispositivos en `/dev` o renombra interfaces de red (no en el LFS) en respuesta a eventos hotplug.

udevd Un demonio que reordena los eventos hotplug antes de suministrárselos a **udev**, para evitar diversas condiciones raras.

udevsend Envía eventos hotplug a **udevd**.

udevstart Crea los nodos de dispositivos en `/dev` que se corresponden con los controladores compilados directamente dentro del núcleo. Realiza la tarea simulando eventos hotplug presumiblemente rechazados por el núcleo antes de la invocación de este programa (por ejemplo, debido a que el sistema de ficheros raíz no se había montado) y suministrando dichos eventos hotplug sintéticos a **udev**.

udevinfo Permite a los usuarios consultar en la base de datos de **udev** información sobre cualquier dispositivo que actualmente se encuentre presente en el sistema. También proporciona una forma de consultar cualquier dispositivo en el árbol `sysfs` para ayudar a crear reglas Udev.

udevtest Simula una ejecución de **udev** para el dispositivo indicado, mostrando el nombre del nodo

que el auténtico **udev** habría creado o (no en el LFS) el nombre de la interfaz de red renombrada.

`/etc/udev` Contiene los ficheros de configuración, de permisos de dispositivos y de reglas para la denominación de dispositivos de **udev**.

6.58. Util-linux-2.12b

El paquete Util-linux contiene una miscelánea de utilidades. Entre otras hay utilidades para manejar sistemas de ficheros, consolas, particiones y mensajes.

Tiempo estimado de construcción: 0.2 SBU
Espacio requerido en disco: 16 MB

La instalación de Util-linux depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed y Zlib

6.58.1. Notas sobre la conformidad con el estándar FHS

El estándar FHS recomienda que usemos `/var/lib/hwclock` para la ubicación del fichero `adjtime`, en lugar del habitual `/etc`. Para hacer que **hwclock** sea conforme a FHS, ejecuta lo siguiente:

```
sed -i 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
    hwclock/hwclock.c
mkdir -p /var/lib/hwclock
```

6.58.2. Instalación de Util-linux

GCC-3.4.1 compila mal **sfdisk** si se utiliza el nivel de optimización que tiene por defecto. El siguiente parche corrige el problema:

```
patch -Np1 -i ../util-linux-2.12b-sfdisk-2.patch
```

Prepara Util-linux para su compilación:

```
./configure
```

Compila el paquete:

```
make HAVE_KILL=yes HAVE_SLN=yes
```

Significado de los parámetros de make:

`HAVE_KILL=yes`
Esto evita que el programa **kill** (que ya ha sido instalado por Procps) sea construido e instalado de nuevo.

`HAVE_SLN=yes`
Esto evita que el programa **sln** (un **ln** enlazado estáticamente, ya instalado por Glibc) se vuelva a construir e instalar.

Este paquete no incluye un banco de pruebas.

Instala el paquete:

```
make HAVE_KILL=yes HAVE_SLN=yes install
```

6.58.3. Contenido de Util-linux

Programas instalados: agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, ipcrm, ipcs, isosize, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, pg, pivot_root, ramsize (enlace a rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (enlace a rdev), script, setfdprm, setsid, setterm, sfdisk, swapdev, swapoff (enlace a swapon), swapon, tunelp, ul, umount, vidmode (enlace a rdev), whereis y write

Descripciones cortas

agetty Abre un puerto de terminal, espera la introducción de un nombre de usuario e invoca al comando **login**.

arch Muestra la arquitectura de la máquina.

blockdev Permite llamar a los controles de entrada/salida (ioctls) de los dispositivos de bloque desde la línea de comandos.

cal Muestra un calendario simple.

cfdisk Se usa para manipular la tabla de particiones del dispositivo indicado.

chkdupexe Encuentra ejecutables duplicados.

col Elimina avances de línea inversos.

colcrt Filtra la salida de **nroff** para terminales a los que les faltan ciertas características como el sobrerefresco o semilíneas.

colrm Elimina las columnas indicadas.

column Formatea un fichero a múltiples columnas.

ctrlaltdel Establece la función de la combinación de teclas Ctrl+Alt+Del para un reinicio duro o blando.

cytune Ajusta los parámetros de los controladores de línea serie para tarjetas Cyclades.

ddate Muestra la fecha Discordante, o convierte las fechas Gregorianas en fechas Discordantes.

dmesg Muestra los mensajes de arranque del núcleo.

elvtune Puede usarse para afinar el rendimiento y la interactividad de un dispositivo de bloque.

fdformat Formatea un disquete a bajo nivel.

fdisk Se usa para manipular la tabla de particiones del dispositivo indicado.

fsck.cramfs Realiza una comprobación de consistencia sobre el sistema de ficheros Cramfs del dispositivo indicado

fsck.minix Realiza una comprobación de consistencia en sistemas de ficheros Minix.

getopt Analiza opciones de la línea de comandos indicada.

hexdump Muestra un fichero en hexadecimal o en otro formato.

hwclock Se usa para leer o ajustar el reloj del ordenador, también llamado RTC (Reloj en Tiempo Real) o reloj BIOS (Sistema Básico de Entrada/Salida).

ipcrm Elimina el recurso IPC (Comunicación Entre Procesos) especificado.

ipcs Facilita información sobre el estado IPC.

isosize Muestra el tamaño de un sistema de ficheros iso9660.

line Copia una única línea.

logger Crea entradas en el registro del sistema.

look Muestra líneas que comienzan con una cadena dada.

losetup Activa y controla los dispositivos de bucle (loop).

mcookie Genera galletas mágicas (magic cookies, números hexadecimales aleatorios de 128 bits) **para xauth**.

mkfs Construye un sistema de ficheros en un dispositivo (normalmente una partición del disco duro).

mkfs.bfs Crea un sistema de ficheros bfs de SCO (Operaciones Santa Cruz).

mkfs.cramfs Crea un sistema de ficheros Cramfs.

mkfs.minix Crea un sistema de ficheros Minix.

mkswap Inicializa el dispositivo o fichero indicado para usarlo como área de intercambio (swap).

more Un filtro para paginar texto pantalla a pantalla.

mount Monta el sistema de ficheros de un dispositivo dado en el directorio indicado del árbol de ficheros del sistema.

namei Muestra los enlaces simbólicos en la ruta de nombres indicada.

pg Muestra un fichero de texto a pantalla completa.

pivot_root Hace que el sistema de ficheros indicado sea el raíz del proceso actual.

ramsize Se usa para establecer el tamaño del disco RAM en una imagen de arranque.

raw Utilizado para enlazar un dispositivo Linux de caracteres directo a un dispositivo de bloque.

rdev Muestra y establece el dispositivo raíz, entre otras cosas, en una imagen de arranque.

readprofile Lee la información sobre perfiles del núcleo.

rename Renombra ficheros, sustituyendo la cadena indicada con otra.

renice Altera la prioridad de los procesos en ejecución.

rev Invierte el orden de las líneas de un fichero.

rootflags Se usa para establecer las opciones de partición raíz en una imagen de arranque.

script Hace un guión a partir de una sesión de terminal.

setfdprm Establece los parámetros facilitados por el usuario para los disquetes.

setsid Lanza programas en una nueva sesión.

setterm Establece los parámetros del terminal.

sfdisk Un manipulador de la tabla de particiones del disco.

swapdev	Se usa para establecer el dispositivo de intercambio en una imagen de arranque.
swapoff	Desactiva los dispositivos y ficheros de paginación e intercambio.
swapon	Activa los dispositivos y ficheros de paginación e intercambio.
tunelp	Ajusta los parámetros de la línea de impresión.
ul	Un filtro para traducir marcas de texto a la secuencia de escape que indica subrayado para el terminal en uso.
umount	Desmonta un sistema de ficheros del árbol de ficheros del sistema.
vidmode	Establece el modo de vídeo en una imagen de arranque.
whereis	Localiza el binario, la fuente y la página del manual de un comando.
write	Envía un mensaje a otro usuario *si* ese usuario no ha desactivado dichos mensajes.

6.59. Sobre los símbolos de depuración

La mayoría de los programas y librerías se compilan por defecto incluyendo los símbolos de depuración (con la opción `-g` de **gcc**). Esto significa que, cuando se depura un programa o librería que fue compilado incluyendo la información de depuración, el depurador no nos da sólo las direcciones de memoria, sino también los nombres de las rutinas y variables.

Sin embargo, la inclusión de estos símbolos de depuración agranda sustancialmente un programa o librería. Para tener una idea del espacio que ocupan estos símbolos, echa un vistazo a lo siguiente:

- Un binario bash con símbolos de depuración: 1200 KB
- Un binario bash sin símbolos de depuración: 480 KB
- Los ficheros de Glibc y GCC (`/lib` y `/usr/lib`) con símbolos de depuración: 87 MB
- Los ficheros de Glibc y GCC sin símbolos de depuración: 16 MB

Los tamaños pueden variar algo dependiendo del compilador y la librería C utilizadas, pero cuando comparamos programas con y sin símbolos de depuración, la diferencia generalmente está en una relación de entre 2 y 5.

Como muchas personas probablemente nunca usen un depurador en su sistema, eliminando estos símbolos se puede liberar una gran cantidad de espacio del disco. Para tu comodidad, la siguiente sección muestra cómo eliminar todos los símbolos de depuración de los programas y librerías. Puedes encontrar información sobre otras formas de optimizar tu sistema en la receta *http://www.lfs-es.com/recetas/optimization.html* (el original en inglés se encuentra en *http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt*).

6.60. Eliminar los símbolos de nuevo.

Si no eres un programador y no planeas depurar el software de tu sistema, puedes reducir tu sistema en unos 200 MB eliminando los símbolos de depuración de los binarios y librerías. Este proceso no produce ningún otro inconveniente que no sea no poder depurar los programas nunca más.

La mayoría de la gente que usa el comando mencionado más adelante no experimenta ningún problema. Pero es fácil cometer un error al escribirlo e inutilizar tu sistema, por lo que antes de ejecutar el comando **strip** posiblemente sea buena idea hacer una copia de respaldo en el estado actual.

Antes de hacer la eliminacón de símbolos, se ha de tener mucho cuidado para asegurar que no se esté ejecutando ningún binario que vaya a ser procesado. Si no estás seguro de si entraste al entorno chroot con el comando mostrado en la Sección 6.3, “Entrar al entorno chroot”, entonces sal primero del chroot:

```
logout
```

Luego vuelve a entrar con:

```
chroot $LFS /tools/bin/env -i \
    HOME=/root TERM=$TERM PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin \
    /tools/bin/bash --login
```

Ahora puedes procesar con tranquilidad los binarios y librerías:

```
/tools/bin/find /{,usr/}{bin,lib,sbin} -type f \
   -exec /tools/bin/strip --strip-debug '{}' ';'
```

Se avisará de que no se reconoce el formato de un buen número de ficheros. Puedes ignorar esos avisos, sólo indican que se trata de guiones en vez de binarios.

Si el espacio en disco es escaso, se puede usar la opción *`--strip-all`* sobre los binarios que hay en `/{,usr/}{bin,sbin}` para ganar varios megabytes más. Pero no uses dicha opción sobre las librerías: las destruirías.

6.61. Limpieza

A partir de ahora, cuando salgas del entorno chroot y desees entrar de nuevo en él, deberás ejecutar el siguiente comando chroot modificado:

```
chroot "$LFS" /usr/bin/env -i \
    HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin \
    /bin/bash --login
```

La razón para esto es que, puesto que ya no son necesarios los programas que hay en `/tools`, el directorio puede borrarse para ganar espacio. Antes de borrar realmente el directorio, sal del chroot y reentra con el anterior comando. Igualmente, antes de eliminar `/tools`, empaquétalo y guárdalo en lugar seguro, en caso de que quieras construir otro sistema LFS.

Nota

Al eliminar `/tools` también se eliminan las copias temporales de Tcl, Expect y DejaGnu que fueron usadas para ejecutar los bancos de pruebas. Si quieres usar estos programas más adelante, necesitarás recompilarlos y reinstalarlos. Las instrucciones de instalación son las mismas de Capítulo 5, excepto por el cambio de la ruta de `/tools` a `/usr`. En el libro BLFS se expone un método ligeramente diferente para instalar Tcl (consúltalo en *http://www.lfs-es.com/blfs-es-CVS/view/svn/general/tcl.html*, o en el original en inglés en *http://www.linuxfromscratch.org/blfs/view/svn/general/tcl.html*).

Los paquetes y parches almacenados en `/sources` pueden moverse a una localización más normal, como `/usr/src/packages`. Puedes borrar por completo el directorio si has quemado su contenido en un CD.

Capítulo 7. Configurar los guiones de arranque del sistema

7.1. Introducción

Este capítulo detalla cómo instalar los guiones de arranque y cómo configurarlos adecuadamente. Muchos de estos guiones funcionarán sin necesidad de modificarlos, pero algunos necesitan ficheros de configuración adicionales, pues manejan información dependiente del hardware.

En este libro se utilizan guiones de inicio al estilo System-V porque son ampliamente utilizados. Para consultar otras opciones, una receta que detalla la configuración del inicio al estilo BSD se encuentra en *http://www.lfs-es.com/recetas/bsd-init.html* (la versión original en inglés se encuentra en *http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt*). Buscando “depinit” en las listas de correo de LFS encontrarás otra alternativa.

Si decides usar algún otro estilo de guiones de inicio, sáltate este capítulo y pasa al Capítulo 8.

7.2. LFS-Bootscripts-2.2.2

El paquete LFS-Bootscripts contiene un conjunto de guiones de arranque.

Tiempo estimado de construcción: 0.1 SBU
Espacio requerido en disco: 0.3 MB

La instalación de LFS-Bootscripts depende de: Bash y Coreutils

7.2.1. Instalación de LFS-Bootscripts

Instala el paquete:

```
make install
```

7.2.2. Contenido de LFS-Bootscripts

Guiones instalados: checkfs, cleanfs, console, functions, halt, ifdown, ifup, localnet, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysklogd, template y udev

Descripciones cortas

checkfs	Comprueba los sistemas de ficheros justo antes de ser montados (con la excepción de los que usan registros de transacciones [journal] o los que se montan desde la red).
cleanfs	Elimina los ficheros que no deben guardarse cuando se arranca de nuevo el sistema, como aquellos en `/var/run/` y `/var/lock/`. Regenera `/var/run/utmp` y elimina los ficheros `/etc/nologin`, `/fastboot` y `/forcefsck` si existen.
console	Carga el mapa de teclado especificado como el adecuado para el tipo de teclado. También establece la fuente de pantalla.
functions	Contiene funciones usadas por diferentes guiones, como el chequeo de errores y de estado.
halt	Cierra el sistema.
ifdown	Ayuda al guión network con los dispositivos de red.
ifup	Ayuda al guión network con los dispositivos de red.
localnet	Establece el nombre de máquina usado por el sistema (hostname) y activa el dispositivo de red interna (loopback).
mountfs	Monta todos los sistemas de ficheros que no estén marcados como *noauto* o que no se monten a través de la red.
mountkernfs	Se usa para montar los sistemas de ficheros suministrados por el núcleo, como `/proc`.
network	Activa las interfaces de red, como las tarjetas de red, y establece la puerta de enlace por defecto (gateway) cuando es necesario.
rc	El controlador maestro de los niveles de arranque. Es el responsable de lanzar todos los demás guiones, uno a uno, en una secuencia determinada por el nombre del enlace simbólico a procesar.
reboot	Reinicia el sistema.

sendsignals	Se asegura de que todos los procesos terminen antes de parar o reiniciar el sistema.
setclock	Fija el reloj del núcleo a la hora local en caso de que el reloj del ordenador no esté fijado a la hora UTC.
static	Suministra la funcionalidad necesaria para asignar una dirección IP estática a una interfaz de red.
swap	Activa y desactiva las particiones y ficheros de intercambio (swap).
sysklogd	Lanza y detiene los demonios de registro de eventos del sistema y del núcleo.
template	Una plantilla para crear guiones de arranque personalizados para otros demonios.
udev	Activa **udev** y crea los nodos de dispositivos en `/dev`.

7.3. ¿Cómo funcionan los guiones de arranque?

Linux utiliza como sistema de inicio SysVinit, que se basa en el concepto de *niveles de ejecución*. Este sistema de inicio puede variar ampliamente de un sistema a otro, por lo tanto no se debe asumir que porque las cosas funcionen en <inserta el nombre de una distribución> tengan que funcionar en LFS también. LFS tiene su propia manera de hacer las cosas, la cual suele respetar los estándares aceptados.

SysVinit (al que llamaremos "init" a partir de este momento) se basa en un esquema de niveles de ejecución. Hay 7 (desde el 0 al 6) niveles de ejecución (en realidad existen más, pero son para casos especiales y es raro utilizarlos. La página man de init describe estos detalles) y cada uno de ellos indica lo que debe hacer el sistema durante el arranque. El nivel de ejecución por omisión es el 3. He aquí una breve descripción de los distintos niveles de ejecución como suelen implementarse:

```
0: parada del sistema
1: modo monousuario
2: modo multiusuario sin red
3: modo multiusuario con red
4: reservado para personalizar, si no, hace lo mismo que el 3
5: Igual que el 4. Normalmente se utiliza para iniciar el entorno
   gráfico (mediante xdm de X o kdm de KDE)
6: reinicio del sistema
```

Para cambiar el nivel de ejecución se utiliza el comando **init *[nivel de ejecución]*** donde *[nivel de ejecución]* representa el nivel de ejecución que se desea arrancar. Por ejemplo, para reiniciar el sistema se utilizaría el comando **init 6**. El comando **reboot** no es más que un alias de dicho comando, al igual que el comando **halt** lo es de **init 0**.

Debajo de `/etc/rc.d` existe una serie de directorios `rc?.d` (donde ? representa el número del nivel de ejecución), más el directorio `rcsysinit.d`, que contienen un conjunto de enlaces simbólicos. Los nombres de estos enlaces simbólicos empiezan con K o con S seguidos de 2 cifras. Los enlaces que comienzan por una *K* indican la parada (kill) de un servicio, mientras que la *S* indica su inicio (start). Las dos cifras determinan el orden de ejecución, desde 00 hasta 99; cuanto menor sea el número, antes se ejecutará. Cuando **init** cambia a otro nivel de ejecución, los servicios apropiados son parados y otros son iniciados.

Los guiones reales se enuentran en `/etc/rc.d/init.d`. Ellos son los que hacen el trabajo y todos los enlaces simbólicos apuntan a ellos. Los enlaces de parada e inicio apuntan al mismo guión en `/etc/rc.d/init.d`. Esto se debe a que los guiones pueden invocarse con parámetros diferentes como `start`, `stop`, `restart`, `reload` y `status`. Cuando se encuentra un enlace K, se ejecuta el guión apropiado con el argumento `stop`. Cuando se encuentra un enlace S, se ejecuta el guión apropiado con el argumento `start`.

Hay una excepción a esta explicación. Los enlaces que comienzan por *S* en los directorios `rc0.d` y `rc6.d` no inician nada. Estos guiones se invocan siempre con el parámetro `stop` para parar algo. La lógica tras esto es que cuando el usuario va a parar o reiniciar el sistema no es necesario iniciar nada. El sistema sólo necesita ser detenido.

He aquí una descripción de lo que hace cada parámetro:

start
: Inicia el servicio.

stop
: Para el servicio.

restart
: El servicio se para y se vuelve a iniciar.

reload
: Se actualiza la configuración del servicio. Este parámetro se utiliza tras la modificación del fichero de configuración cuando no es necesario reiniciar el servicio.

status
: Dice si el servicio se está ejecutando y con qué identificador de proceso (PID).

Eres libre de modificar la forma en que funciona el proceso de arranque (después de todo es tu propio sistema LFS). Los ficheros aquí mostrados son un ejemplo de cómo puede hacerse.

7.4. Manejo de dispositivos y módulos en un sistema LFS

En el Capítulo 6 se instaló el paquete Udev. Antes de entrar en detalles sobre cómo funciona, repasaremos los anteriores métodos de manejo de dispositivos.

Tradicionalmente, los sistemas Linux en general utilizan un método estático de creación de dispositivos, implicando que un gran número de nodos de dispositivo son creados en `/dev` (literalmente, cientos de nodos) sin tener en cuenta si el dispositivo hardware correspondiente existe en realidad. Esto se hace típicamente mediante un guión **MAKEDEV**, que contiene una serie de llamadas al programa **mknod** con los números mayor y menor correspondientes a cada posible dispositivo que pudiera existir en el mundo. Con el uso del método Udev, sólo se crearán los nodos correspondientes a aquellos dispositivos detectados por el núcleo. Debido a que estos nodos de dispositivo se crearán cada vez que se inicie el sistema, se almacenarán en un `ramfs` (un sistema de ficheros que existe por completo en memoria y no ocupa espacio en disco). Los nodos de dispositivo no necesitan mucho espacio, por lo que la memoria utilizada es muy poca.

7.4.1. Historia

En Febrero de 2000, un nuevo sistema de ficheros llamado `devfs` fue incluido en los núcleos 2.3.46 y estuvo disponible en la serie 2.4 de los núcleos estables. Aunque estaba presente en las propias fuentes del núcleo, este método de creación dinámica de dispositivos nunca recibió mucho apoyo por parte del equipo de desarrolladores del núcleo.

El principal problema con el sistema adoptado por `devfs` era el modo en el que manejaba la detección, creación y denominación de dispositivos. El último punto, la denominación de los nodos, fue quizás el más crítico. Está generalmente aceptado que si los nombres de dispositivos son configurables, entonces las políticas de denominación deberían ser establecidas por un administrador del sistema y no impuestas por un desarrollador en particular. El sistema de ficheros `devfs` sufre también de extraños comportamientos inherentes a su diseño y que no pueden corregirse sin una revisión sustancial del núcleo. También se ha marcado como descartado debido a la falta de mantenimiento reciente.

Con el desarrollo del árbol inestable 2.5 del núcleo, posteriormente liberado como núcleos estables de la serie 2.6, aparece un nuevo sistema de ficheros virtual llamado `sysfs`. El trabajo de `sysfs` es exportar una visión de la estructura del sistema a los procesos de usuario. Con esta representación visible a nivel de usuario, la posibilidad de encontrar un sustituto para `devfs` a nivel de usuario se hace mucho más real.

7.4.2. Implementación de Udev

Arriba se mencionó brevemente el sistema de ficheros `sysfs`. Uno podría preguntarse cómo conoce `sysfs` los dispositivos presentes en el sistema y qué números de dispositivo debe usar. Los crontroladores que se han compilado directamente dentro del núcleo registran sus objetos en `sysfs` a medida que son detectados por el núcleo. Para los controladores compilados como módulos, esto sucederá cuando se cargue el módulo. Una vez montado el sistema de ficheros `sysfs` (en `/sys`), los datos registrados en `sysfs` por los controladores están disponibles para los procesos de usuario y para que **udev** cree los nodos de dispositivo.

El guión de inicio **S10udev** se ocupa de la creación de dichos nodos de dispositivo cuando se inicia Linux. Este guión comienza registrando **/sbin/udev** como manejador de eventos “hotplug”. Los eventos hotplug (explicados más adelante) no deberían generarse durante esta fase, pero se registra **udev** por si ocurriesen. Entonces, el programa **udevstart** recorre el sistema de ficheros `/sys` y crea en `/dev` los dispositivos que coinciden con las descripciones. Por ejemplo, `/sys/class/tty/vcs/dev` contiene la cadena “7:0”. Esta cadena la utiliza **udevstart** para crear `/dev/vcs` con el número mayor *7* y menor *0*. Los permisos para todos y cada uno de los dispositivos creados por **udevstart** se establecen mediante el uso de los ficheros que hay en el directorio `/etc/udev.d/permissions.d/`. Estos se encuentran numerados de forma similar a los guiones de arranque de LFS. Si **udev** no puede encontrar un fichero de permisos para el dispositivo que está creando, usará por defecto los permisos *600* y propietario *root:root*. El nombre de los nodos creados en el directorio `/dev` se configuran según las reglas especificadas en los ficheros que hay dentro del directorio `/etc/udev/rules.d/`.

Una vez completado el proceso anterior, estarán disponibles para el usuario todos los dispositivos realmente presentes y cuyos controladores estén compilados dentro del núcleo. ¿Qué hay de aquellos dispositivos cuyos controladores son módulos?

Anteriormente mencionamos el concepto de un “manejador de eventos hotplug”. Cuando el núcleo detecte la conexión de un nuevo dispositivo, generará un evento de conexión en caliente (hotplug) y mirará en `/proc/sys/kernel/hotplug` para encontrar el programa de nivel de usuario que maneja la conexión de dispositivos. El guión de inicio **udev** registró **udev** como dicho manejador. Cuando se generan estos eventos hotplug, el núcleo le indica a **udev** que compruebe en el sistema de ficheros `/sys` la información relativa a este nuevo dispositivo y que cree para él la entrada en `/dev`.

Esto nos expone a un problema que existe con **udev** y que antes existía con `devfs`. Se conoce comúnmente como el problema de “el huevo y la gallina”. La mayoría de distribuciones Linux manejan la carga de módulos mediante entradas en `/etc/modules.conf`. Los accesos a un nodo de dispositivo provocan que se cargue el módulo del núcleo correspondiente. Este método no funcionará con **udev** debido a que el nodo de dispositivo no existe hasta que se cargue el módulo. Para solucionar esto, se añadió al paquete LFS-Bootscripts el guión de inicio **S05modules** junto con el fichero `/etc/sysconfig/modules`. Mediante la adición de los nombres de los módulos al fichero `modules`, estos módulos se cargarán al iniciar el ordenador. Esto permite a **udev** detectar los dispositivos y crear los nodos corespondientes.

Ten en cuenta que en máquinas lentas o para dispositivos que crean muchos nodos, el proceso de creación puede tadar varios segundos en completarse. Esto significa que algunos nodos de dispositivo no estarán disponibles inmediatamente.

7.4.3. Manejo dinámico de dispositivos

Cuando conectas un dispositivo, como un reproductor MP3 por USB, el núcleo reconoce que el disposiivo se encuentra ahora conectado y genera un evento hotplug. Si el controlador se encuentra cargado (porque fue compilado dentro del núcleo o cargado por el guión de arranque **S05modules**), se llamará a **udev** para crear los nodos de dispositivo adecuados según los datos de `sysfs` disponibles en `/sys`. Si el controlador para el dispositivo recién conectado se encuentra disponible como módulo pero no está cargado, entoces la conexión del dispositivo al sistema sólo causará que el bus del núcleo genere un evento hotplug que notifica a nivel de usuario la conexión del nuevo dispositivo y este no será enlazado a un controlador. En efecto, no sucede nada y el propio dispositivo no puede utilizarse aún.

Si construyes un sistema que contiene gran cantidad de controladores compilados como módulos en vez de directamente en el núcleo, el uso de **S05modules** puede que no sea práctico. El paquete Hotplug (mira *http://linux-hotplug.sourceforge.net/*) puede ser de ayuda en estos casos. Cuando se instala el paquete Hotplug, este responderá a los eventos hotplug del bus del núcleo antes mencionados. El paquete Hotplug cargará el módulo correspondiente y hará que este dispositivo esté disponible creando los nodos.

7.4.4. Problemas con la creación de dispositivos

Hay unos cuantos problemas conocidos cuando se trata la creación automática de nodos de dispositivos:

1) Puede que un controlador del núcleo no exporte sus datos a `sysfs`.

Esto es muy común con controladores suministrados por el fabricante y ajenos al árbol del núcleo. Para estos controladores no se crearán sus nodos. Para crear manualmente los dispositivos, utiliza el fichero de configuración `/etc/sysconfig/createfiles`. Consulta el fichero `devices.txt` de la documentación del núcleo, o la documentación de dicho controlador, para encontrar los números mayor y menor adecuados.

2) Se necesita un controlador que no pertenece al hardware. Esto es muy común con los módulos de compatibilidad Open Sound System (OSS, Sistema Abierto de Sonido) del proyecto Advanced Linux Sound Architecture (ALSA, Arquitectura Avanzada de Sonido en Linux). Se puede manejar este tipo de dispositivos de dos formas:

- Añadiendo los nombres de los módulos a `/etc/sysconfig/modules`
- Usando una línea “install” en `/etc/modprobe.conf`. Esto le indica al comando **modprobe** que “cuando se cargue este módulo, cargue también y al mismo tiempo este otro módulo”. Por ejemplo:

```
install snd-pcm modprobe -i snd-pcm ; modprobe \
    snd-pcm-oss ; true
```

  Esto provocará que el sistema cargue tanto el módulo *snd-pcm* como el módulo *snd-pcm-oss* cuando se haga cualquier petición para cargar el módulo *snd-pcm*.

7.4.5. Lecturas útiles

En los siguientes sitios hay disponible documentación de ayuda adicional:

- Una implementación de devfs a nivel de usuario:
  http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf
- FAQ de Udev:
  http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ
- El modelo de controladores del núcleo Linux:
  http://public.planetmirror.com/pub/lca/2003/proceedings/papers/Patrick_Mochel/Patrick_Mochel.pdf

7.5. Configuración del guión setclock

El guión **setclock** lee la hora del reloj interno del ordenador, conocido también como reloj BIOS o CMOS (Semiconductor de Oxido de Metal Complementario). Si el reloj hardware está establecido a la hora UTC, este guión la convierte a la hora local mediante el fichero `/etc/localtime` (que le indica al programa **hwclock** en qué zona horaria se encuentra el usuario). No hay manera de detectar automáticamente si el reloj utiliza UTC o no, así que esto se debe configurar manualmente.

Si no puedes recordar si el reloj hardware está en UTC o no, averígualo ejecutando el comando **`hwclock --localtime --show`**. Esto mostrará la hora actual según el reloj hardware. Si dicha hora coincide con la de tu reloj, entonces el reloj hardware está a la hora local. Si la salida de **hwclock** no es la hora local, seguramente esté en la hora UTC. Verifica esto añadiendo o restando la cantidad de horas correspondiente a tu zona local a la hora mostrada por **hwclock**. Por ejemplo, si vives en la zona horaria MST, conocida también como GMT -0700, añade siete horas a la hora local. Ten en cuente también el Horario de Ahorro de Energía, que implica descontar una hora (o añadir sólo seis en el ejemplo) durante los meses de verano.

Cambia abajo el valor de la variable `UTC` a *`0`* (cero) si el reloj hardware *no* utiliza la hora UTC.

Crea un nuevo fichero `/etc/sysconfig/clock` ejecutando lo siguiente:

```
cat > /etc/sysconfig/clock << "EOF"
# Inicio de /etc/sysconfig/clock

UTC=1

# Fin de /etc/sysconfig/clock
EOF
```

En *http://www.lfs-es.com/recetas/time.html* hay disponible una buena receta que trata sobre la hora en LFS (en *http://www.linuxfromscratch.org/hints/downloads/files/time.txt* se encuentra la versión original en inglés). En ella se explican conceptos como las zonas horarias, UTC y la variable de entorno `TZ`.

7.6. Configurar la consola Linux

Esta sección explica cómo configurar el guión de arranque **console**, el cual establece el mapa del teclado y la fuente de consola. Si no se van a utilizar caracteres no ASCII (los caracteres de la Libra inglesa y el Euro son ejemplos de caracteres no ASCII) y el teclado es del tipo U.S., sáltate esta sección. Sin el fichero de configuración el guión de inicio **console** no hará nada.

El guión **console** utiliza `/etc/sysconfig/console` como fichero de configuración. Decide qué mapa de teclado y fuente de pantalla se usarán. El CÓMO específico para tu idioma puede ayudarte en esto. Con el paquete LFS-Bootscripts se instaló un fichero `/etc/sysconfig/console` prefabricado con los ajustes conocidos para diversos paises, por lo que basta con descomentar la sección correspondiente si tu país está entre los definidos. Si aún tienes dudas, mira en el directorio `/usr/share/kbd` los mapas de teclados y fuentes de pantalla válidos. Lee las páginas de manual de **loadkeys** y **setfont** para determinar los argumentos correctos para estos programas. Una vez decidido, crea el fichero de configuración con el siguiente comando:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="[argumentos para loadkeys]"
FONT="[argumentos para setfont]"
EOF
```

Por ejemlo, para los usuarios de idioma español que también quieran usar el caracter del Euro (accesible presionano AltGr+e), la siguiente configuración es correcta:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="es euro2"
FONT="lat9-16 -u iso01"
EOF
```

Nota

La línea `FONT` anterior es correcta sólo para el conjunto de caracteres ISO 8859-15. Si se utiliza ISO 8859-1 y, por tanto, el símbolo de la Libra en vez del Euro, la línea `FONT` correcta sería:

```
FONT="lat1-16"
```

Si no se establece la variable `KEYMAP` o `FONT`, el guión de inicio **console** no ejecutará el programa correspondiente.

En algunos mapas de teclado, las teclas Retroceso y Borrar envían caracteres diferentes a los del mapa de teclado incluido en el núcleo. Esto confunde a algunas aplicaciones. Por ejemplo, Emacs muestra su ayuda (en vez de borrar el caracter anterior al cursor) cuando se pulsa Retroceso. Para comprobar si el mapa del teclado en uso está afectado (esto sólo funciona con mapas de teclado i386) haz:

```
zgrep '\W14\W' [/ruta/a/tu/mapa_de_teclado]
```

Si el keycode 14 es “Backspace” en lugar de “Delete”, crea el siguiente fragmento de mapa de teclado para corregir el problema:

```
mkdir -p /etc/kbd && cat > /etc/kbd/bs-sends-del <<"EOF"
                  keycode  14 = Delete Delete Delete Delete
              alt keycode  14 = Meta_Delete
        altgr alt keycode  14 = Meta_Delete
                  keycode 111 = Remove
    altgr control keycode 111 = Boot
      control alt keycode 111 = Boot
altgr control alt keycode 111 = Boot
EOF
```

Indícale al guión **console** que cargue este fragmento tras el mapa de teclado principal:

```
cat >>/etc/sysconfig/console <<"EOF"
KEYMAP_CORRECTION="/etc/kbd/bs-sends-del"
EOF
```

Para compilar el mapa de teclado directamente en el núcleo, en vez de establecerlo cada vez desde el guión de arranque **console**, sigue las instrucciones que se muestran en la Sección 8.3, “Linux-2.6.8.1”. Haciendo esto te aseguras de que el teclado funcione siempre como se espera, incluso cuando arranques en modo de mantenimiento (pasándole *`init=/bin/sh`* al núcleo), pues el guión de inicio **console** no se ejecutará en dicha situación. Adicionalmente, el núcleo no establecerá automaticamente la fuente de pantalla. Esto no debería ser demasiado problema, pues los caracteres ASCII se manejarán correctamente y es improbable que pudieses necesitar caracteres no ASCII estando en modo de mantenimiento.

Puesto que el núcleo establecería el mapa del teclado, se puede omitir la variable `KEYMAP` del fichero de configuración `/etc/sysconfig/console`. También puede dejarse en su sitio, si se desea, sin que haya consecuencias. Mantenerlo puede ser beneficioso si posees varios núcleos diferentes y te es difícil asegurar que el mapa de teclado se haya compilado dentro de todos ellos.

7.7. Crear el fichero /etc/inputrc

El fichero `/etc/inputrc` se ocupa del mapeado del teclado para situaciones concretas. Este fichero es el fichero de inicio usado por Readline, la librería para cuestiones de entrada usada por Bash y otros intérpretes de comandos.

Para más información, mira la página Info de Bash, sección *Readline Init File (Fichero de Inicio de Readline)*. La página Info de Readline es también una buena fuente de información.

Los valores globales se establecen en `/etc/inputrc`. Los valores personales del usuario se establecen en `~/.inputrc`. El fichero `~/.inputrc` sobreescribe los valores globales. En una página posterior se configura Bash para usar `/etc/inputrc` si no hay un `.inputrc` del usuario cuando se lee `/etc/profile` (normalmente al entrar al sistema). Para hacer que el sistema utilice ambos, o para evitar un manejo global del teclado, es buena idea añadir un fichero `.inputrc` por defecto en el directorio `/etc/skel` para utilizar con los nuevos usuarios.

A continuación hay un `/etc/inputrc` básico, con comentarios para explicar lo que hace cada opción. Advierte que los comentarios no pueden estar en la misma línea que los comandos.

Para crear un `.inputrc` en `/etc/skel` usando el siguiente comando, cambia la salida del comando a `/etc/skel/.inputrc` y asegúrate de comprobar/establecer sus permisos. Copia dicho fichero a `/etc/inputrc` y al directorio personal de cada usuario que ya exista en el sistema, incluido *root*, que necesita una versión privada de este fichero. Utiliza el parámetro `-p` de **cp** para manterner los permisos y cambia el usuario y grupo a los adecuados en cada caso.

```
cat > /etc/inputrc << "EOF"
# Inicio de /etc/inputrc
# Modificado por Chris Lynn <roryo@roryo.dynup.net>

# Se asegura de no mostrar todo en la primera línea
set horizontal-scroll-mode Off

# Activa la entrada de 8 bits
set meta-flag On
set input-meta On

# Desactiva la supresión del bit 8
set convert-meta Off

# Mantiene el bit 8 para ser mostrado
set output-meta On

# none, visible o audible
set bell-style none

# Todo lo siguiente mapea la secuencia de escape
# del valor contenido en el primer argumento a las
# funciones específicas de readline

"\eOd": backward-word
"\eOc": forward-word

# Para la consola linux
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert

# Para xterm
"\eOH": beginning-of-line
"\eOF": end-of-line

# Para Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line

# Fin de /etc/inputrc
EOF
```

7.8. Los ficheros de inicio de Bash

El intérprete de comandos **/bin/bash** (al que nos referiremos como “el intérprete”) utiliza una colección de ficheros de inicio para ayudar a crear un entorno de trabajo. Cada fichero tiene un uso específico y pueden generar diferentes entornos de ingreso o interactivos. Los ficheros del directorio `/etc` proporcionan ajustes globales. Si existe un fichero diferente en el directorio personal, este puede sobreescribir los ajustes globales.

Un intérprete de ingreso interactivo se inicia tras ingresar en el sistema, usando **/bin/login**, mediante la lectura del fichero `/etc/passwd`. Un intérprete interactivo de no ingreso se inicia en la línea de comandos (es decir, `[prompt]$`**/bin/bash**). Un intérprete no interactivo está presente usualmente cuando se ejecuta un guión del intérprete de comandos. Es no interactivo porque está procesando un guión y no esperando indicaciones del usuario entre comandos.

Para más información, consulta **info bash** - Nodo: Bash Startup Files and Interactive Shells (Ficheros de inicio de Bash e intérpretes interactivos.

Los ficheros `/etc/profile` y `~/.bash_profile` son leidos cuando el intérprete se invoca como un intérprete interactivo de ingreso.

El siguiente fichero `/etc/profile` básico establece algunas variables de entorno necesarias para el soporte de idioma nativo. Al establecerlas correctamente se obtiene:

- La salida de los programas traducida al idioma nativo.
- Correcta clasificación de los caracteres en letras, dígitos y otros tipos. Esto es necesario para que Bash acepte correctamente los caracteres no ASCII en la línea de comandos en idiomas diferentes al inglés.
- La correcta ordenación alfabética propia del país.
- Un apropiado tamaño de papel por defecto.
- Un formato correcto para los valores monetarios, horarios y fechas.

Este guión establece también la variable de entoro `INPUTRC` que hace que Bash y Readline utilicen el fichero `/etc/inputrc` creado anteriormente.

Sustituye a continuación *`[ll]`* con el código de dos letras del idioma deseado (por ejemplo, “es”) y *`[CC]`* con el código de dos letras de tu país (por ejemplo, “ES”). También puede ser necesario especificar (y en realidad esta es la forma preferida) la codificación de caracteres (por ejemplo, “iso8859-15”) tras un punto (por lo que el resultado sería “es_ES.iso8859-15”). Para más información, ejecuta el siguiente comando:

```
man 3 setlocale
```

La lista de todas las locales soportadas (instaladas) por Glibc se puede obtener ejecutando el siguiente comando:

```
locale -a
```

Una vez hayas determinado los ajustes correctos para el idioma, crea el fichero `/etc/profile`:

```
cat > /etc/profile << "EOF"
# Inicio de /etc/profile

export LC_ALL=[ll]_[CC]
export LANG=[ll]_[CC]
export INPUTRC=/etc/inputrc

# Fin de /etc/profile
EOF
```

Nota

Las locales “C” (la que se tiene por defecto) y “en_US” (la recomendada para los usuarios de habla inglesa de los Estados Unidos) son diferentes.

Configurar el esquema del teclado, la fuente de pantalla y las variables de entorno relacionadas con las locales son los únicos pasos necesarios para soportar las codificaciones ordinarias de un byte y dirección de escritura de izquierda a derecha. Los casos más complejos (incluidas las locales basadas en UTF-8) necesitan pasos y parches adicionales debido a que muchas aplicaciones tienden a funcionar incorrectamente bajo tales condiciones. Estos pasos y parches no se incluyen en el libro LFS y dichas locales no están soportadas de ninguna forma por el sistema LFS.

7.9. Configuración del guión sysklogd

El guión `sysklogd` invoca al programa **syslogd** con la opción *-m 0*. Esta opción deshabilita la marca de tiempo periódica que **syslogd** escribe por defecto en el fichero de registro cada 20 minutos. Para habilitar esta marca de tiempo periódica, edita el guión `sysklogd` y realiza los cambios necesarios. Para más información mira **`man syslogd`**.

7.10. Configuración del guión localnet

Una de las cosas que hace el guión localnet es establecer el nombre de la máquina. Es necesario configurar dicho nombre en `/etc/sysconfig/network`.

Crea el fichero `/etc/sysconfig/network` e introduce el nombre de tu máquina ejecutando:

```
echo "HOSTNAME=[lfs]" > /etc/sysconfig/network
```

Debes substituir *`[lfs]`* por el nombre con el que debe de conocerse tu máquina. No escribas el FQDN (nombre completo de la máquina, incluido su dominio). Esa información la escribiremos más tarde en el fichero `/etc/hosts`

7.11. Creación del fichero /etc/hosts

Si se va a configurar una tarjeta de red, decide la dirección IP, el FQDN y los posibles alias para escribirlos en el fichero `/etc/hosts`. La sintaxis es:

```
<dirección IP> miordenador.example.org alias
```

A no ser que tu computadora sea visible en Internet (es decir, tengas un dominio registrado y asignado un bloque de direcciones IP válido, la mayoría no tenemos esto), deberías asegurarte de que la dirección IP queda dentro del rango de direcciones IP de la red privada. Los rangos válidos son:

```
Clases de redes
A     10.0.0.0
B     Del 172.16.0.0 al 172.31.0.0
C     Del 192.168.0.0 al 192.168.255.0
```

Una dirección IP válida puede ser 192.168.1.1. Un FQDN válido para esa dirección IP podría ser www.linuxfromscratch.org (no uses este, pues es un dominio válido registrado y podría causarte problemas con el servidor de nombres de dominio).

Aunque no vayas a configurar la tarjeta de red necesitas un FQDN. Algunos programas lo necesitan para funcionar correctamente.

Crea el fichero `/etc/hosts` ejecutando:

```
cat > /etc/hosts << "EOF"
# Inicio de /etc/hosts (versión con trajeta de red)

127.0.0.1 localhost
[192.168.1.1] [<HOSTNAME>.example.org] [HOSTNAME]

# Fin de /etc/hosts (versión con tarjeta de red)
EOF
```

Debes cambiar los valores *`[192.168.1.1]`* y *`[<HOSTNAME>.example.org]`* por los tuyos específicos o los requeridos (si la máquina estará conectada a una red ya existente y el administrador de la red/sistema es el que asigna una dirección IP).

Si no se va a configurar una tarjeta de red, crea el fichero `/etc/hosts` ejecutando:

```
cat > /etc/hosts << "EOF"
# Inicio de /etc/hosts (versión sin tarjeta de red)

127.0.0.1 [<HOSTNAME>.example.org] [HOSTNAME] localhost

# Fin de /etc/hosts (versión sin tarjeta de red)
EOF
```

7.12. Configuración del guión network

Esta sección solamente es aplicable en el caso de que vayas a configurar una tarjeta de red.

Si no tienes tarjeta de red es muy probable que no vayas a crear ninguna configuración relacionada con ellas. En ese caso, elimina los enlaces simbólicos `network` de todos los directorios de los niveles de ejecución (`/etc/rc.d/rc*.d`)

7.12.1. Creación de los ficheros de configuración de la interfaz de red

Qué interfaces de red activa o desactiva el guión network depende de los ficheros situados en el directorio `/etc/sysconfig/network-devices`. Este directorio debe contener ficheros del tipo `ifconfig.xyz`, donde "xyz" es el nombre de la interfaz de red (como eth0 o eth0:1).

Si decides renombrar o mover el directorio `/etc/sysconfig/network-devices`, asegúrate de editar el fichero `/etc/sysconfig/rc` y actualizar la opción "network_devices" con la nueva ruta.

En este directorio se crean nuevos ficheros. El siguiente comando crea un fichero `ipv4` de ejemplo para el dispositivo *eth0*:

```
cd /etc/sysconfig/network-devices &&
mkdir ifconfig.eth0 &&
cat > ifconfig.eth0/ipv4 << "EOF"
ONBOOT=yes
SERVICE=ipv4-static
IP=192.168.1.1
GATEWAY=192.168.1.2
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

Los valores de estas variables se deben cambiar en todos los ficheros por los valores apropiados. Si la variable `ONBOOT` tiene el valor "yes", el guión network activará la NIC (Interfaz de Tarjeta de Red) correspondiente durante el arranque del sistema. Si contiene cualquier otro valor, el guión network ignorará la NIC correspondiente y no la activará.

La entrada `SERVICE` define el método para obtener la dirección IP. Los guiones de arranque de LFS tienen un formato de asignación de IP modular, y mediante la creación de ficheros adicionales en `/etc/sysconfig/network-devices/services` se permiten otros métodos de asignación IP. Esto se utiliza comúnmente para DHCP (Protocolo de Configuración Dinámica del Anfitrión), que se explica en el libro BLFS.

La variable `GATEWAY` debería contener la dirección IP de la puerta de enlace por efecto, si hay alguna. Si no, comenta la variable.

La variable `PREFIX` debe contener el número de bits usados en la subred. Cada octeto de una dirección IP tiene 8 bits. Si la máscara de subred es 255.255.255.0, estonces está usando los primeros tres octetos (24 bits) para especificar el número de red. Si la máscara de red es 255.255.255.240, podría estar usando los primeros 28 bits. Los prefijos mayores de 24 bits son usados normalmente por ISPs (Suministradores de Servicios de Internet) para DSL o cable. En este ejemplo (PREFIX=24), la máscara de red es 255.255.255.0. Ajústalo de acuerdo a tu propia subred.

7.12.2. Creación del fichero /etc/resolv.conf

Si el sistema va a estar conectado a Internet, necesitará algún tipo de resolución de nombres DNS para resolver los nombres de dominio de Internet a direcciones IP y viceversa. Esto se consigue mejor colocando la dirección IP del servidor DNS, facilitado por el ISP o administrador de red, en `/etc/resolv.conf`. Crea este fichero ejecutando lo siguiente:

```
cat > /etc/resolv.conf << "EOF"
# Inicio de /etc/resolv.conf

domain {[tu nombre de dominio]}
nameserver [dirección IP del servidor de nombres primario]
nameserver [dirección IP del servidor de nombres secundario]

# Fin de /etc/resolv.conf
EOF
```

Sustituye *`[dirección IP del servidor de nombres]`* con la dirección IP del servidor DNS más apropiado para tu configuración. Con frecuencia hay más de una entrada (los requisitos establecen servidores secundarios como respaldo). Si sólo necesitas o deseas un servidor DNS, elimina la segunda línea *nameserver* del fichero. La dirección IP pueder ser incluso un enrutador de la red local.

Capítulo 8. Hacer el sistema LFS arrancable

8.1. Introducción

Es hora de hacer arrancable el sistema LFS. En este capítulo se explica la creación de un fichero `fstab`, la construcción de un núcleo para el nuevo sistema LFS y la instalación del gestor de arranque Grub para que el sistema LFS se pueda seleccionar para arrancar al inicio.

8.2. Creación del fichero /etc/fstab

El fichero `/etc/fstab` lo utilizan ciertos programas para determinar dónde se montan por defecto los sistemas de ficheros, cuáles deben verificarse y en qué orden. Crea una nueva tabla de sistemas de ficheros:

```
cat > /etc/fstab << "EOF"
# Inicio de /etc/fstab

# sistema de   punto de     tipo   opciones     volcado  orden de
# ficheros     montaje                                  chequeo

/dev/[xxx]     /            [fff]  defaults        1     1
/dev/[yyy]     swap         swap   pri=1           0     0
proc           /proc        proc   defaults        0     0
sysfs          /sys         sysfs  defaults        0     0
devpts         /dev/pts     devpts gid=4,mode=620  0     0
shm            /dev/shm     tmpfs  defaults        0     0
# Fin de /etc/fstab
EOF
```

Reemplaza *`[xxx]`*, *`[yyy]`* y *`[fff]`* con los valores apropiados para tu sistema, por ejemplo `hda2`, `hda5` y `ext2`. Para ver todos los detalles de los seis campos de este fichero, consulta **man 5 fstab**.

Cuando se utiliza un sistema de ficheros con registro de transaciones, los valores *`1 1`* que aparecen al final de la línea deberían cambiarse a *`0 0`*, ya que no se necesita volcar ni verificar estas particiones.

El punto de montaje `/dev/shm` para `tmpfs` se incluye para permitir la activación de la memoria compartida POSIX. Tu núcleo debe tener compilado en su interior el soporte requerido para que funcione (más datos sobre esto en la siguiente sección). Ten en cuenta que actualmente muy poco software utiliza en realidad la memoria compartida POSIX. Por tanto, puedes considerar como opcional el montaje de `/dev/shm`. Para más información consulta `Documentation/filesystems/tmpfs.txt` en el árbol de fuentes del núcleo.

Existen otras líneas que podrían añadirse al fichero `fstab`. Un ejemplo es la línea para dispositivos USB:

```
usbfs         /proc/bus/usb usbfs   devgid=14,devmode=0660 0 0
```

Esta opción sólo funcionará si se compila dentro del núcleo “Support for Host-side USB” (Soporte para USB del lado del anfitrión) y “USB device filesystem” (Sistema de ficheros para dispositivos USB).

8.3. Linux-2.6.8.1

El paquete Linux contiene el núcleo y los ficheros de cabecera.

Tiempo estimado de construcción: 4.20 SBU
Espacio requerido en disco: 181 MB

La instalación de Linux depende de: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl y Sed

8.3.1. Instalación del núcleo

Construir el núcleo comprende varios pasos: configuración, compilación e instalación. Mira en el fichero `README` del árbol de fuentes del núcleo los métodos de configuración del núcleo alternativos al utilizado en este libro.

Prepara la compilación ejecutando el siguiente comando:

```
make mrproper
```

Esto asegura que el árbol del núcleo está completamente limpio. El equipo del núcleo recomienda que se ejecute este comando antes de cada compilación del núcleo. No debes confiar en que el árbol de las fuentes esté limpio tras desempaquetarlo.

Igualmente, asegurate de que el núcleo no intente pasar los eventos hotplug al espacio de usuario hasta que el espacio de usuario esté preparado:

```
sed -i 's@/sbin/hotplug@/bin/true@' kernel/kmod.c
```

Si en la Sección 7.6, “Configurar la consola Linux”, decidiste compilar el mapa del teclado dentro del núcleo, ejecuta el siguiente comando:

```
loadkeys -m /usr/share/kbd/keymaps/[ruta al mapa del teclado] > \
    drivers/char/defkeymap.c
```

Por ejemplo, usa `/usr/share/kbd/keymaps/i386/qwerty/es.map.gz` si tienes un teclado español.

Configura el núcleo mediante una interfaz de menús:

```
make menuconfig
```

Alternativamente, **`make oldconfig`** puede ser más adecuado en algunas situaciones. Lee el fichero `README` para más detalles.

Nota

Cuando configures el núcleo, asegúrate de activar la opción “Support for hot-pluggable devices” (Soporte para dispositivos conectables en caliente) que se encuentra en el menú “General Setup” (Configuración general). Esto activa los eventos hotplug usados por **udev** para poblar el directorio `/dev` con nodos de dispositivo.

Si lo deseas, sáltate la configuración del núcleo copiando el fichero de configuración del núcleo, `.config`, de tu sistema anfitrión (asumiendo que esté disponible) al directorio `linux-2.6.8.1`. Sin embargo, no recomendamos esta opción. Con frecuencia es mejor explorar todos los menús de configuración y crear tu propia configuración del núcleo desde cero.

Para el soporte de la memoria compartida POSIX, asegúrate de que esté activada la opción de configuración del núcleo “Virtual memory file system support” (Soporte del sistema de ficheros de memoria virtual). Se encuentra en el menú “File systems” (Sistemas de ficheros) y normalmente está activada por defecto.

Los guiones de arranque de LFS asumen que tanto “Support for Host-side USB” como “USB device filesystem” han sido compilados dentro del núcleo o no se han compilado. Los guiones de arranque no funcionarán bién si se trata de un módulo (usbcore.ko).

Nota

NPTL requiere que el núcleo sea compilado con GCC 3.x, en este caso 3.4.1. Se sabe que compilarlo con 2.95.x provoca fallos en el banco de pruebas de Glibc, por lo que no es recomendable compilar el núcleo con GCC 2.95.x.

Compila la imagen del núcleo y los módulos:

```
make
```

Si utilizas los módulos del núcleo puede que necesites un fichero `/etc/modprobe.conf`. La información relativa a los módulos y a la configuración del núcleo en general puedes encontrarla en el directorio `/usr/src/linux-2.6.8.1/Documentation`, que contiene la documentación del núcleo. La página de manual de *modprobe.conf* puede que también sea de interés.

Ten mucho cuidado cuando leas otra documentación, pues normalmente sólo es aplicable a los núcleos 2.4.x. Hasta donde nosotros sabemos, los temas de configuración del núcleo específicos para Hotplug y Udev no están documentados. El problema es que Udev creará un nodo de dispositivo sólo si Hotplug o un guión escrito por el usuario inserta el módulo correspondiente en el núcleo, y no todos los módulos son detectables por Hotplug. Advierte que sentencias como la mostrada a continuación en el fichero `/etc/modprobe.conf` no funcionarán con Udev:

```
alias char-major-XXX cualquier-módulo
```

Debido a las complicaciones con Hotplug, Udev y los módulos, recomendamos encarecidamente comenzar con una configuración del núcleo que sea por completo no modular, especialmente si es la primera vez que utilizas Udev.

Instala los módulos, si la configuración del núcleo los utiliza:

```
make modules_install
```

Si tienes muchos módulos y muy poco espacio, puede que quieras considerar eliminarles los símbolos y comprimirlos. Para muchos dicha compresión no compensa el tiempo, pero si realmente estás presionado por el espacio, mira *http://www.linux-mips.org/archives/linux-mips/2002-04/msg00031.html*.

Tras completar la compilación se necesitan algunos pasos adicionales para completar la instalación. Es necesario copiar varios ficheros al directorio `/boot`.

La ruta a la imagen del núcleo puede variar dependiendo de la plataforma que utilices. Ejecuta el siguiente comando para instalar el núcleo:

```
cp arch/i386/boot/bzImage /boot/lfskernel-2.6.8.1
```

`System.map` es un fichero de símbolos para el núcleo. Mapea los puntos de entrada de cada una de las funciones en la API del núcleo, al igual que las direcciones de las estructuras de datos del núcleo para el núcleo en ejecución. Ejecuta el siguiente comando para instalar el fichero de mapa:

```
cp System.map /boot/System.map-2.6.8.1
```

`.config` es el fichero de configuración del núcleo creado por el paso **make menuconfig** anterior. Contiene todas las selecciones de configuración para el núcleo que se acaba de compilar. Es buena idea guardar este fichero como referencia futura:

```
cp .config /boot/config-2.6.8.1
```

Es importante advertir que los ficheros del directorio de las fuentes del núcleo no son propiedad de *root*. Cuando se desempaqueta un paquete como usuario *root* (como hacemos dentro del chroot), los ficheros acaban teniendo los identificadores de usuario y grupo que tenían en la máquina en la que se empaquetaron. Esto normalmente no es problema para cualquier otro paquete que instales debido a que eliminas las fuentes tras la instalación. Pero con frecuencia el árbol de las fuentes de Linux se guarda durante mucho tiempo, por lo que es posible que el ID de usuario del empaquetador sea asignado a alguien en tu máquina y entonces dicha persona podría tener permiso de escritura en las fuentes del núcleo.

Si vas a guardar el árbol de las fuentes del núcleo, ejecuta **`chown -R 0:0`** sobre el directorio `linux-2.6.8.1` para asegurar que todos los ficheros son propiedad de *root*.

8.3.2. Contenido de Linux

Ficheros instalados: núcleo, cabeceras del núcleo y System.map

Descripciones cortas

`núcleo`	El corazón del sistema GNU/Linux. Cuando enciendes tu ordenador, lo primero que se carga del sistema operativo es el núcleo. Éste detecta e inicializa todos los componentes hardware del ordenador, poniendo estos componentes a disposición del software como si fuesen un árbol de ficheros y convierte una CPU única en una máquina multi-tarea capaz de ejecutar concurrentemente varios programas casi al mismo tiempo.
`cabeceras del núcleo`	Definen la interfaz a los servicios proporcionados por el núcleo. Las cabeceras del directorio `include` del sistema deben ser *siempre* aquellas contra las que se compiló Glibc y, por tanto, *nunca* deben reemplazarse al actualizar el núcleo.
`System.map`	Un listado de direcciones y símbolos. Mapea los puntos de entrada y direcciones de todas las funciones y estructuras de datos del núcleo.

8.4. Hacer el sistema LFS arrancable

Tu nuevo y brillante sistema LFS está casi completo. Una de las últimas cosas por hacer es asegurarse de que puede ser arrancado. Las siguientes instrucciones sólo son aplicables en ordenadores de arquitectura IA-32, o sea PCs. La información sobre "cargadores de arranque" para otras arquitecturas debería estar disponible en las localizaciones usuales de recursos específicos para esas arquitecturas.

El arranque puede ser una tarea compleja. Primero, unas palabras de advertencia. Familiarízate con tu actual gestor de arranque y con cualquier otro sistema operativo presente en tu(s) disco(s) duro(s) que desees mantener arrancable. Asegúrate de que tienes preparado un disco de arranque de emergencia para poder "rescatar" el ordenador si este quedase inutilizable (no arrancable).

Anteriormente compilamos e instalamos el gestor de arranque Grub en preparación para este paso. El proceso consiste en escribir ciertos ficheros especiales de Grub a su localización específica en el disco duro. Antes de hacer esto te recomendamos encarecidamente que crees un disquete de arranque de Grub como respaldo. Inserta un disquete en blanco y ejecuta los siguientes comandos:

```
dd if=/boot/grub/stage1 of=/dev/fd0 bs=512 count=1
dd if=/boot/grub/stage2 of=/dev/fd0 bs=512 seek=1
```

Saca el disquete y guárdalo en lugar seguro. Ahora inicia el intérprete de comandos de **grub**:

```
grub
```

Grub utiliza su propia estructura de nombres para los discos de la forma *(hdn,m)*, donde *n* es el número del disco duro y *m* es el número de la partición, comenzando ambos desde 0. Por ejemplo, la partición `hda1` es *(hd0,0)* para Grub, y `hdb3` es *(hd1,2)*. Al contrario que Linux, Grub no considera los dispositivos CD-ROM como discos duros. Por ejemplo, si tienes un CD en `hdb` y un segundo disco duro en `hdc`, este segundo disco duro seguiría siendo *(hd1)*.

Usando la información anterior, determina la denominación apropiada para tu partición raíz (o partición de arranque, si usas una separada). Para los siguientes ejemplos asumiremos que tu partición raíz (o la de arranque) es `hda4`

Indícale a Grub dónde debe buscar sus ficheros `stage{1,2}`. Puedes utilizar el tabulador para que Grub te muestre las alternativas:

```
root (hd0,3)
```

Aviso

El siguiente comando sobreescribirá tu actual gestor de arranque. No ejecutes el comando si esto no es lo que quieres. Por ejemplo, si utilizas otro gestor de arranque para administrar tu MBR (Master Boot Record, Registro Maestro de Arranque). En este escenario, posiblemente tenga más sentido instalar Grub en el "sector de arranque" de la partición LFS, en cuyo caso dicho comando sería **`setup (hd0,3)`**.

Indícale a Grub que se instale en el MBR de `hda`:

```
setup (hd0)
```

Si todo está bien, Grub informará que ha encontrado sus ficheros en `/boot/grub`. Esto es todo para activarlo. Cierra el intérprete de comandos de **grub**:

```
quit
```

Crea un fichero de “lista de menú” para definir el menú de arranque de Grub:

```
cat > /boot/grub/menu.lst << "EOF"
# Inicio de /boot/grub/menu.lst

# Inicia por defecto la primera entrada del menú.
default 0

# Espera 30 segundos antes de iniciar la entrada por defecto.
timeout 30

# Usa bonitos colores.
color green/black light-green/black

# La primera entrada es para LFS.
title LFS 6.0
root (hd0,3)
kernel /boot/lfskernel-2.6.8.1 root=/dev/hda4
EOF
```

Si lo desas, añade una entrada para la distribución anfitriona. Tendrá un aspecto similar a este:

```
cat >> /boot/grub/menu.lst << "EOF"
title Red Hat
root (hd0,2)
kernel /boot/kernel-2.4.20 root=/dev/hda3
initrd /boot/initrd-2.4.20
EOF
```

Si necesitas un arranque dual a Windows, la siguiente entrada debería permitirte iniciarlo:

```
cat >> /boot/grub/menu.lst << "EOF"
title Windows
rootnoverify (hd0,0)
chainloader +1
EOF
```

Si **info grub** no te dice todo lo que quieres saber, puedes encontrar más información sobre Grub en su sitio web, localizado en: *http://www.gnu.org/software/grub/*.

Capítulo 9. El final

9.1. El final

¡Bien hecho! ¡El nuevo sistema LFS está instalado! Te deseamos mucha diversión con tu flamante sistema Linux hecho a la medida.

Puede ser una buena idea crear un fichero `/etc/lfs-release`. Teniendo este fichero te será muy fácil (y a nosotros, si es que vas a pedir ayuda en algún momento) saber qué versión de LFS tienes instalada en tu sistema. Crea este fichero ejecutando:

```
echo 6.0 > /etc/lfs-release
```

9.2. Registrarse

Ahora que has terminado el libro, ¿qué te parecería poder registrarte como usuario de LFS? Visita *http://www.linuxfromscratch.org/cgi-bin/lfscounter.cgi* y regístrate como usuario de LFS introduciendo tu nombre y la primera versión de LFS que has usado.

Arranquemos ahora el sistema LFS.

9.3. Reinicio del sistema

Ahora que se han instalado todos los programas, es hora de reiniciar el ordenador. Sin embargo, debes tener en cuenta varias cosas. El sistema que has creado en este libro es bastante reducido y muy posiblemente no tenga la funcionalidad que podrías necesitar para seguir adelante. Instalar varios paquetes adicionales del libro BLFS mientras aún estás en el entorno chroot te dejará en una mejor posición para continuar una vez que reinicies tu nueva instalación LFS. Al instalar un navegador web en modo texto, como Lynx, podrás facilmente ver el libro BLFS en una terminal mientras compilas los paquetes en otra. El paquete GPM te permitirá copìar y pegar en tu terminal virtual. Por último, si estás en una situación en la que una configuración de IP estática no cubre los requisitos de tu red, instalar ahora paquetes como dhcpcd o ppp es también útil.

Una vez dicho esto, ¡vayamos a arrancar nuestra nueva instalación de LFS por primera vez!. Primero sal del entorno chroot:

```
logout
```

Desmonta los sistemas de ficheros virtuales:

```
umount $LFS/dev/pts
umount $LFS/dev/shm
umount $LFS/dev
umount $LFS/proc
umount $LFS/sys
```

Desmonta el sistema de ficheros del LFS:

```
umount $LFS
```

Si creaste varias particiones, desmonta las otras particiones antes de desmontar la principal, por ejemplo:

```
umount $LFS/usr
umount $LFS/home
umount $LFS
```

Ahora reinicia el sistema con:

```
shutdown -r now
```

Asumiendo que el gestor de arranque Grub fue configurado como se indicó anteriormente, el menú está establecido para que *LFS 6.0* arranque automáticamente.

Una vez terminado el reinicio, el sistema LFS está listo para su uso y se puede añadir más software.

9.4. ¿Y ahora, qué?

Gracias por leer el libro LFS. Esperamos que lo hayas encontrado útil y hayas aprendido algo sobre el proceso de creación del sistema.

Ahora que el sistema LFS está instalado, puede que te preguntes “¿Y ahora, qué?”. Para responder esta cuestión te hemos preparado una lista de recursos.

- Más Allá de Linux From Scratch

  El libro Más Allá de Linux From Scratch (BLFS) cubre los procesos de instalación de paquetes muy diferentes que están fuera del objetivo del Libro LFS. Puedes encontrar el proyecto BLFS en *http://www.linuxfromscratch.org/blfs/* y la traducción al castellano del libro en *http://www.lfs-es.com/blfs-es-CVS/*.

- Recetas de LFS

  Las recetas de LFS son una serie de documentos educacionales, suministrados por voluntarios a la comunidad LFS. Las recetas están disponibles en *http://www.linuxfromscratch.org/hints/list.html*. En *http://www.lfs-es.com/recetas/* puedes encontrar la traducción de un buen número de ellas, aunque muchas se encuentran desfasadas en el momento de publicar este libro.

- Listas de Correo

  Hay varias listas de correo sobre LFS a las que puedes suscribirte si necesitas ayuda, si quieres mantenerte al corriente de los últimos desarrollos, si quieres contribuir al proyecto y más. Para más información consulta el Capítulo 1 - Listas de correo.

- El Proyecto de Documentación de Linux (TLDP)

  El objetivo del Proyecto de Documentación de Linux es colaborar en todo lo relacionado con la creación y publicación de la documentación sobre Linux. El LDP ofrece una gran colección de CÓMOS, Guías y páginas de manual y puedes encontrarlo en *http://www.tldp.org/*. Su filial en castellano se encuentra en *http://es.tldp.org*.

Parte IV. Apéndices

Apéndice A. Acrónimos y términos

ABI Application Binary Interface - Interfaz de Aplicación Binaria

ALFS Automated Linux From Scratch - Linux From Scratch Automatizado

ALSA Advanced Linux Sound Architecture - Arquitectura Avanzada de Sonido en Linux

API Application Programming Interface - Interfaz de Aplicación para Programación

ASCII American Standard Code for Information Interchange - Código Americano Estándar para el Intercambio de Información

BIOS Basic Input/Output System - Sistema Básico de Entrada/Salida

BLFS Beyond Linux From Scratch - Más Allá de Linux From Scratch

BSD Berkeley Software Distribution - Distribución Berkeley de Software

chroot change root - cambio de raíz

CMOS Complementary Metal Oxide Semiconductor - Semiconductor Complementario de Oxido de Metal

COS Class Of Service - Clase De Servicio

CPU Central Processing Unit - Unidad Central de Procesamiento

CRC Cyclic Redundancy Check - Comprobación Cíclica de Redundancia

CVS Concurrent Versions System - Sistema Concurrente de Versiones

DHCP Dynamic Host Configuration Protocol - Protocolo Dinámico de Configuración del Anfitrión

DNS Domain Name Service - Servicio de Nombres de Dominio

EGA Enhanced Graphics Adapter - Adaptador Mejorado de Gráficos

ELF Executable and Linkable Format - Formato Ejecutable y Enlazable

EOF End Of File - Fin De Fichero

EQN equation - ecuación

EVMS Enterprise Volume Management System - Sistema Empresarial de Administración de Volúmenes

ext2 second extended file system - segundo sistema de ficheros extendido

FAQ Frequently Asked Questions - Cuestiones Preguntadas Frecuentemente

FHS Filesystem Hierarchy Standard - Estándar de la Jerarquía de Sistemas de Ficheros

FIFO First In, First Out - Primero en Entrar, Primero en Salir

FQDN Fully Qualified Domain Name - Nombre de Dominio Completamente Cualificado

FTP File Transfer Protocol - Protocolo de Transferencia de Ficheros

GB Gibabytes

GCC GNU Compiler Collection - Colección GNU de Compiladores

GID Group Identifier - Identificador de Grupo

GMT Greenwich Mean Time - Tiempo del Meridiano de Greenwich

GPG GNU Privacy Guard - Guardián GNU de Privacidad

HTML Hypertext Markup Language - Lenguaje de Marcas de Hipertexto

IDE Integrated Drive Electronics - Controlador Electrónico Integrado

IEEE Institute of Electrical and Electronic Engineers - Instituto de Ingenieros en Electricidad y Electrónica

IO Input/Output - Entrada/Salida

IP Internet Protocol - Protocolo de Internet

IPC Inter-Process Communication - Comunicación Entre Procesos

IRC Internet Relay Chat - Charlas en Internet

ISO International Organization for Standardization - Organización Internacional para la Estandarización

ISP Internet Service Provider - Proveedor de Servicios de Internet

KB Kilobytes

LED Light Emitting Diode - Diodo Emisor de Luz

LFS Linux From Scratch

LSB Linux Standards Base - Base de los Estándares en Linux

MB Megabytes

MBR Master Boot Record - Registro Maestro de Arranque

MD5 Message Digest 5 - Resúmen 5 del Mensaje

NIC Network Interface Card - Tarjeta de Interfaz de Red

NLS Native Language Support - Soporte para Lenguaje Nativo

NNTP Network News Transport Protocol - Protocolo de Red para Transporte de Noticias

NPTL Native POSIX Threading Library - Librería Nativa de Hilos POSIX

OSS Open Sound System - Sistema Abierto de Sonido

PCH Pre-Compiled Headers - Cabeceras Precompiladas

PCRE Perl Compatible Regular Expression - Expresión Regular Compatible de Perl

PID Process Identifier - Identificador del Proceso

PLFS Pure Linux From Scratch - Linux From Scratch Puro

PTY pseudo terminal

QA Quality Assurance - Control de Calidad

QOS Quality Of Service - Calidad Del Servicio

RAM Random Access Memory - Memoria de Acceso Aleatorio

RPC Remote Procedure Call - Llamada a Procedimiento Remoto

RTC Real Time Clock - Reloj de Tiempo Real

SBU Static Build Unit - Unidad Estática de Construcción

SCO The Santa Cruz Operation

SGR Select Graphic Rendition - Interpretación de la Selección Gráfica

SHA1 Secure-Hash Algorithm 1 - Algoritmo 1 de Tabla Segura

SMP Symmetric Multi-Processor - Multiprocesador Simétrico

TLDP The Linux Documentation Project - El Proyecto de Documentación Linux

TFTP Trivial File Transfer Protocol - Protocolo Trivial de Transferencia de Ficheros

TLS Thread-Local Storage - Almacenamiento Local de Hilos

UID User Identifier - Identificador de Usuario

umask user file-creation mask - máscara de creación de ficheros del usuario

USB Universal Serial Bus - Bus Serie Universal

UTC Coordinated Universal Time - Tiempo Universal Coordinado

UUID Universally Unique Identifier - Identificador Universalmente Unico

VC Virtual Console - Consola Virtual

VGA Video Graphics Array - Matríz de Gráficos de Vídeo

VT Virtual Terminal - Terminal Virtual

Apéndice B. Agradecimientos

Queremos agradecer a las siguientes personas y organizaciones su contribución al Proyecto LFS-ES:

- *Gerard Beekmans*, por crear el apasionante proyecto Linux From Scratch.
- *Red ECOLNET*, por prestarnos su apoyo incondicional desde el primer momento y facilitarnos los servicios de CVS, listas de correo y espacio web, que son vitales para realizar nuestro trabajo.
- *Alberto Ferrer*, por poner a nuestra disposición los servicios de hospedaje de Datta.
- *Al Equipo del LFS-ES*, por su dedicación e interés en conseguir que este proyecto funcione y que las traducciones tengan la mejor calidad posible.
- A todos aquellos que leen nuestras traducciones con interés, pues es para ellos para quienes las escribimos.

Queremos agradecer a las siguientes personas y organizaciones su contribución al Proyecto Linux From Scratch:

Miembros del equipo del proyecto

- *Gerard Beekmans* <gerard@linuxfromscratch.org> – Iniciador de Linux From Scratch, organizador del Proyecto LFS.
- *Christine Barczak* <theladyskye@linuxfromscratch.org> – Editora del libro LFS
- *Matthew Burgess* <matthew@linuxfromscratch.org> – Co-líder del proyecto LFS, mantenedor general de los paquetes del LFS, editor técnico del libro LFS.
- *Craig Colton* <meerkats@bellsouth.net> – Creador del logotipo para los proyectos LFS, ALFS, BLFS y Hints.
- *Nathan Coulson* <nathan@linuxfromscratch.org> – Mantenedor de los guiones de arranque de LFS.
- *Jeroen Coumans* <jeroen@linuxfromscratch.org> – Desarrollador del sitio web, mantenedor de las FAQ.
- *Bruce Dubbs* <bdubbs@linuxfromscratch.org> – Líder del equipo de calidad de LFS, editor del libro BLFS.
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – Mantenedor de los XML/XSL de LFS.
- *Jim Gifford* <jim@linuxfromscratch.org> – Escritor técnico de LFS, mantenedor de los parches.
- *Nicholas Leippe* <nicholas@linuxfromscratch.org> – Mantenedor del Wiki.
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Mantenedor de los guiones de generación del sitio web.
- *Scot Mc Pherson* <scot@linuxfromscratch.org> – Mantenedor de la pasarela NNTP de LFS.
- *Ryan Oliver* <ryan@linuxfromscratch.org> – Líder del equipo de pruebas, mantenedor de las herramientas principales y co-creador del PLFS.
- *Alexander Patrakov* <semzx@newmail.ru> – Anterior editor técnico del LFS
- *James Robertson* <jwrober@linuxfromscratch.org> – Mantenedor de Bugzilla, desarrollador del Wiki y editor técnico del libro LFS.

- *Tushar Teredesai* <tushar@linuxfromscratch.org> – Editor del libro BLFS y mantenedor de los proyectos Hints y Patches.
- *Jeremy Utley* <jeremy@linuxfromscratch.org> – Editor técnico del libro LFS, mantenedor de Bugzilla, mantenedor de LFS-Bootscripts y co-administrador del servidor LFS.
- *Zack Winkles* <zwinkles@gmail.com> – Anterior editor técnico del LFS.
- Innumerables personas de las diversas listas de correo de LFS y BLFS que han hecho que este libro sea posible mediante sus sugerencias, probando el libro y suministrando informes de errores, instrucciones y sus experiencias con la instalación de diversos paquetes.

Traductores

- *Manuel Canales Esparcia* <macana@lfs-es.com> – Proyecto de traducción al castellano de LFS.
- *Johan Lenglet* <johan@linuxfromscratch.org> – Proyecto de traducción al francés de LFS.
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Proyecto de traducción al portugués de LFS.
- *Thomas Reitelbach* <tr@erdfunkstelle.de> – Proyecto de traducción al alemán de LFS.

Administradores de la red de réplicas

América del Norte

- *Scott Kveton* <scott@osuosl.org> – lfs.oregonstate.edu
- *Mikhail Pastukhov* <miha@xuy.biz> – lfs.130th.net.
- *William Astle* <lost@l-w.net> – ca.linuxfromscratch.org.
- *Jeremy Polen* <jpolen@rackspace.com> – us2.linuxfromscratch.org mirror.
- *Tim Jackson* <tim@idge.net> – linuxfromscratch.idge.net.
- *Jeremy Utley* <jeremy@linux-phreak.net> – lfs.linux-phreak.net.

América del Sur

- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.info mirror.
- *Andres Meggiotto* <sysop@mesi.com.ar> – lfs.mesi.com.ar.
- *Eduardo B. Fonseca* <ebf@aedsolucoes.com.br> – br.linuxfromscratch.org mirror.

Europa

- *Barna Koczka* <barna@siker.hu> – hu.linuxfromscratch.org.
- *UK Mirror Service* – linuxfromscratch.mirror.ac.uk.
- *Martin Voss* <Martin.Voss@ada.de> – lfs.linux-matrix.net.
- *Guido Passet* <guido@primerelay.net> – nl.linuxfromscratch.org mirror.

- *Bastiaan Jacques* <baafie@planet.nl> – lfs.pagefault.net
- *Roel Neefs* <lfs-mirror@linuxfromscratch.rave.org> – linuxfromscratch.rave.org.
- *Justin Knierim* <justin@jrknierim.de> – www.lfs-matrix.de
- *Stephan Brendel* <stevie@stevie20.de> – lfs.netservice-neuss.de mirror.
- *Antonin Sprinzl* <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org mirror.
- *Fredrik Danerklint* <fredan-lfs@fredan.org> – se.linuxfromscratch.org mirror.
- *Parisian sysadmins* <archive@doc.cs.univ-paris8.fr> – www2.fr.linuxfromscratch.org.
- *Alexander Velin* <velin@zadnik.org> – bg.linuxfromscratch.org.
- *Dirk Webster* <dirk@securewebservices.co.uk> – lfs.securewebservices.co.uk mirror
- *Thomas Skyt* <thomas@sofagang.dk> – dk.linuxfromscratch.org.
- *Simon Nicoll* <sime@dot-sime.com> – uk.linuxfromscratch.org.

Asia

- *Pui Yong* <pyng@spam.averse.net> – sg.linuxfromscratch.org mirror.
- *Stuart Harris* <stuart@althalus.me.uk> – lfs.mirror.intermedia.com.sg mirror

Australia

- *Jason Andrade* <jason@dstc.edu.au> – au.linuxfromscratch.org.

Un agradecimiento muy especial a nuestros donadores

- *Dean Benson* <dean@vipersoft.co.uk> por múltiples donaciones monetarias.
- *Hagen Herrschaft* <hrx@hrxnet.de> por donar un sistema P4 a 2.2GHz, que ahora corre bajo el nombre de Lorien.
- *VA Software* que, en nombre de *Linux.com*, donó una estación de trabajo VA Linux 420 (antes StartX SP2).
- Mark Stone por donar Belgarath, el servidor linuxfromscratch.org.

Índice

Paquetes

Programas

Librerías

Guiones

Otros