Linux From Scratch

Versión 5.1.1

Gerard Beekmans

Linux From Scratch: Versión 5.1.1

por Gerard Beekmans

Traducido por el proyecto LFS-ES

Versión de la traducción: FINAL del 19 de Agosto de 2004

Este libro describe el proceso para la creación de un sistema Linux desde cero, usando solamente las fuentes del software necesario.

Dedicatoria

Este libro está dedicado a la querida esposa de Gerard Beekmans, *Beverly Beekmans*.

Tabla de contenidos

Prólogo

Prefacio

Después de haber usado diferentes distribuciones de Linux, nunca estuve satisfecho con ninguna de ellas. No me gustaba la forma en la que estaban organizados los guiones de arranque, o no me gustaba la configuración por defecto de ciertos programas, y cosas por el estilo. Llegué a darme cuenta de que si quería estar completamente satisfecho con algún sistema Linux, tenía que construir el mío propio desde cero, usando, idealmente, sólo el código fuente. Sin utilizar paquetes precompilados de ninguna clase. Sin la ayuda de un CD-ROM o disco de arranque que instalase utilidades básicas. Utilizaría mi sistema Linux actual para construir el mío por mi cuenta.

Esta, en su momento, idea descabellada se presentó muy difícil y algunas veces casi imposible. Después de sortear toda clase de problemas de dependencias, de compilación, etc., creé un sistema Linux hecho a medida y completamente funcional. Llamé a este sistema LFS, que significa Linux From Scratch (Linux Desde Cero).

¡Espero que paséis buenos momentos trabajando en vuestro LFS!

--
Gerard Beekmans
gerard@linuxfromscratch.org

Audiencia

Quién puede querer leer este libro

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 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. Sólo están ahí gastando 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; el espacio total de disco usado fue, aproximadamente, 8 MB. Con un repaso adicional para reducirlo, se podría llegar a 5 MB o menos. Intenta eso con una distribución normal.

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, a la barbacoa, o puedes comerla cruda.

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 como quieras.

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. Hasta que examines el nuevo parche y lo implementes por ti mismo no tienes garantía de que ese nuevo paquete se haya construido correctamente y realmente solucione el problema (de forma adecuada).

Hay muy buenas razones para construir tu propio sistema LFS aparte de las aquí listadas. Esta sección solo es 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.

A quién puede que no le interese leer el libro

Posiblemente algunos, por la razón que sea, sientan que no desean leer este libro. Si no deseas construir tu propio sistema Linux desde cero probablemente no quieras leer este libro. Nuestra meta es ayudarte a construir los fundamentos de un sistema completo y utilizable. 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). 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 .

Mientras decides lo que vas a leer, considera tu objetivo. Si deseas construir un sistema Linux mientras aprendes un poco en el camino, entonces este libro es tu mejor elección. Si tu objetivo es estrictamente educacional, y no tienes planes para tu sistema terminado, entonces el "De La Puesta En Marcha Al Indicador Del Bash CÓMO" es, probablemente, la mejor elección.

Podrás encontrar el "De La Puesta En Marcha Al Indicador De Bash CÓMO" en http://personal.telefonica.terra.es/web/aus/linux/p2b/power2bash.html y el original "From Power Up To Bash Prompt HOWTO" en http://axiom.anu.edu.au/~okeefe/p2b/ o en el sitio web de The Linux Documentation Project en http://www.tldp.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html.

Prerrequisitos

Este libro asume que sus lectores tienen un buen conocimiento sobre la utilización e instalación de software en Linux. Antes de que empieces a construir tu sistema LFS, deberías leer los siguientes CÓMOs:

- Software-Building-HOWTO (Construcción de Software CÓMO)

  Esta es una guía asequible sobre cómo construir e instalar las distribuciones de software Unix “genéricas” bajo Linux. Este CÓMO está disponible en http://www.tldp.org/HOWTO/Software-Building-HOWTO.html.

- The Linux Users' Guide (La Guía del Usuario de Linux)

  Esta guía cubre el uso de una amplia gama de software Linux. Está disponible en castellano en http://es.tldp.org/Manuales-LuCAS/GLUP/glup_0.6-1.1-html-1.1 y el original en inglés se encuentra en http://espc22.murdoch.edu.au/~stewart/guide/guide.html.

- The Essential Pre-Reading Hint (Receta de las lecturas previas esenciales)

  Esta es una receta del LFS escrita específicamente para los nuevos usuarios de Linux. Es básicamente un listado de enlaces a excelentes fuentes de información sobre un amplio rango de tópicos. Cualquier persona que intente instalar LFS debería, al menos, comprender muchos de los tópicos mencionados en esta receta. Está disponible en http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt

Tipografía

Para facilitar la comprensión se utilizan ciertas convenciones a lo largo del libro. Aquí hay unos ejemplos:

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

El texto con este estilo debe teclearse exactamente como aparece, a menos que se indique lo contrario. 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'
```

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

Énfasis

Este tipo de texto se utiliza con varios fines en el libro, principalmente para poner de relieve puntos importantes y para dar ejemplos de qué se debe teclear.

http://www.linuxfromscratch.org/

Este tipo de texto se usa para hipervínculos, tanto al propio libro como a páginas exteriores, direcciones de descarga, CÓMOs o sitios web.

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

Este tipo de secciones se usa principalmente al crear 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 EOF. Por lo tanto, generalmente la sección entera debe teclearse tal cual.

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 donar el dominio lfs-es.org y el servidor en el que se aloja.
- 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:

Actuales miembros del equipo del proyecto

- Gerard Beekmans <gerard@linuxfromscratch.org> -- Iniciador de Linux From Scratch, organizador del Proyecto LFS.
- Matthew Burgess <matthew@linuxfromscratch.org> -- Co-líder del proyecto LFS, mantenedor general de los paquetes del LFS, editor 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 LFS-Bootscripts.
- 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> -- Editor del libro LFS (XML).
- Alex Groenewoud <alex@linuxfromscratch.org> -- Editor del libro LFS.
- Mark Hymers <markh@linuxfromscratch.org> -- Mantenedor del CVS, creador del libro BLFS y anterior editor del libro BLFS.
- James Iwanek <iwanek@linuxfromscratch.org> -- Miembro del equipo de administración de sistemas.
- Nicholas Leippe <nicholas@linuxfromscratch.org> -- Mantenedor del Wiki.
- Anderson Lizardo <lizardo@linuxfromscratch.org> -- Creador y mantenedor de los guiones de generación del sitio web.
- Bill Maltby <bill@linuxfromscratch.org> -- Organizador del Proyecto LFS.
- Alexander Patrakov <alexander@linuxfromscratch.org> -- Editor del libro LFS (internacionalización/localización).
- 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.
- James Robertson <jwrober@linuxfromscratch.org> -- Mantenedor de Bugzilla, desarrollador del Wiki y editor del libro LFS.
- Greg Schafer <greg@linuxfromscratch.org> -- Mantenedor de las herramientas principales, anterior editor del libro LFS y co-creador del PLFS.
- Tushar Teredesai <tushar@linuxfromscratch.org> -- Editor del libro BLFS y mantenedor de los proyectos Hints y Patches.
- Jeremy Utley <jeremy@linuxfromscratch.org> -- Editor del libro LFS, mantenedor de Bugzilla, mantenedor de LFS-Bootscripts y co-administrador del servidor LFS.
- Zack Winkles <winkie@linuxfromscratch.org> -- Editor del libro LFS (Tecnologías emergentes) y co-mantenedor

de LFS-Bootscripts.

- 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.org> -- 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 alemás 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.
- Frank Mancuso <crash4o4@gameover.com> -- lfs.crash404.com.
- 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.org 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.
- Desconocido -- mirror.vtx.ch mirror
- 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.
- Desconocido -- linuxfromscratch.je-zi.de mirror
- Desconocido -- linuxfromscratch.tuxcenter.net mirror
- Hagen Herrschaft <hrx@hrxnet.de> -- de.linuxfromscratch.org.

- 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
- Desconocido -- lfs.mirror.if.itb.ac.id mirror

Australia

- Jason Andrade <jason@dstc.edu.au> -- au.linuxfromscratch.org.

Donaciones

- Dean Benson <dean@vipersoft.co.uk> por múltiples donaciones monetarias.
- DREAMWVR.COM por su anterior patrocinio al donar varios recursos a LFS y a los subproyectos relacionados.
- Hagen Herrschaft <hrx@hrxnet.de> por donar un sistema P4 a 2.2GHz, al que hemos llamado *lorien*.
- O'Reilly por donar libros sobre SQL y PHP.
- VA Software que, en nombre de Linux.com, donó una estación de trabajo VA Linux 420 (antes StartX SP2).
- Mark Stone por donar *shadowfax*, el primer servidor de linuxfromscratch.org , un P3 750 MHz con 512 MB RAM y dos discos SCSI de 9 GB. Cuando el servidor se movió lo rebautizamos como *belgarath*.
- Jesse Tie-Ten-Quee <highos@linuxfromscratch.org> por donar una regrabadora de CDs Yamaha CDRW 8824E.
- Innumerables personas en las diversas listas de LFS que están mejorando este libro al aportar sugerencias, enviar informes de errores y exponer sus críticas.

Anteriores miembros del equipo y colaboradores

- Timothy Bauscher <timothy@linuxfromscratch.org> -- Editor del libro LFS, mantenedor del proyecto Hints.
- Robert Briggs por donar originalmente los nombres de dominio *linuxfromscratch.org* y *linuxfromscratch.com*.
- Ian Chilton <ian@ichilton.co.uk> por mantener el proyecto Hints.
- Marc Heerdink <gimli@linuxfromscratch.org> -- Editor del libro LFS.
- Seth W. Klein <sklein@linuxfromscratch.org> -- Creador de las FAQ de LFS.
- Garrett LeSage <garrett@linuxart.com> -- Creador del logotipo original de LFS.
- Simon Perreault <nomis80@videotron.ca> -- Mantenedor del proyecto Hints.
- Geert Poels <Geert.Poels@skynet.be> -- Creador del logotipo original de BLFS, basado en el logotipo de Garrett LeSage.
- Frank Skettino <bkenoah@oswd.org> por el diseño inicial del antiguo sitio web - mira http://www.oswd.org.
- Jesse Tie-Ten-Quee <highos@linuxfromscratch.org> por hospedar temporalmente el servidor de

linuxfromscratch.org, por responder incontables preguntas en el IRC y tener grandes dosis de paciencia.

Estructura

Este libro se divide las siguientes partes:

Parte I - Introducción

En la Parte I se explica un poco sobre cómo se construye el sistema LFS, se listan los cambios realizados en el libro desde la última versión y se da consejos acerca de cómo buscar ayuda en caso de que encuentres problemas.

Parte II - Preparativos para la construcción

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

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 básico Linux obtenido será los cimientos sobre los 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

Cómo van a hacerse las cosas

Vas a construir tu sistema LFS utilizando una distribución ya instalada (como Debian, SuSE, Slackware, Mandrake o RedHat). El sistema Linux existente (el anfitrión) se utilizará como punto de inicio, pues necesitas herramientas tales como un compilador, un enlazador y un intérprete de comandos para construir el nuevo sistema. Normalmente, las herramientas necesarias están disponibles por defecto si seleccionas “desarrollo” como una de las opciones cuando instalas tu distribución.

En el Capítulo 2[p.11] crearás primero una nueva partición y un sistema de ficheros, el sitio donde se compilará e instalará tu nuevo sistema LFS. Después, en el Capítulo 3[p.15], descargarás todos los paquetes y parches necesarios para construir un sistema LFS y los guardarás en el nuevo sistema de ficheros. En el Capítulo 4[p.22] establecerás un buen entorno de trabajo.

En el Capítulo 5[p.28] se describe la instalación de una serie de paquetes que constituyen el entorno básico de desarrollo (o *herramientas principales*) utilizado para construir el sistema real en el Capítulo 6[p.69]. Varios de estos paquetes son necesarios para resolver dependencias circulares. Por ejemplo, para compilar un compilador necesitas un compilador.

Lo primero que se hará en el Capítulo 5[p.28] es construir en una primera fase las herramientas principales, compuestas por Binutils y GCC. Los programas de estos paquetes se enlazarán estáticamente para poder utilizarlos independientemente del sistema anfitrión. Lo segundo será construir Glibc, la librería C. Esta se construirá con los programas de las herramientas principales que acabamos de construir. Lo tercero es construir una segunda fase de las herramientas principales, esta vez enlazadas dinámicamente contra la recién construida Glibc. Todos los restantes paquetes del Capítulo 5[p.28] se construirán usando esta segunda fase de las herramientas principales y serán enlazados dinámicamente contra la nueva Glibc independiente del anfitrión. Cuando esto esté hecho, el proceso de instalación de LFS ya no dependerá de la distribución anfitriona, con la excepción del núcleo en ejecución.

Puede que pienses que “esto parece mucho trabajo para simplemente aislarme de mi distribución anfitriona”. Al principio del Capítulo 5[p.28] se da una explicación técnica completa, incluyendo algunas notas sobre las diferencias entre programas enlazados estática y dinámicamente.

En el Capítulo 6[p.69] construirás tu 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. La razón de que en realidad no reinicies sino que uses chroot, es que crear un sistema arrancable requiere un trabajo adicional que no es necesario aún. Pero la mayor ventaja es que chroot te permite seguir usando el sistema anfitrión mientras se construye tu LFS. Mientras esperas que se complete la compilación de un paquete, puedes simplemente cambiar a otra consola virtual o escritorio X y continuar usando tu ordenador como lo harías normalmente.

Para terminar la instalación, en el Capítulo 7[p.164] se configuran los guiones de arranque; el núcleo y el gestor de arranque se configuran en el Capítulo 8[p.174] y el Capítulo 9[p.180] tiene algunas indicaciones para ayudarte una vez que finalices el libro. Entonces, por fin, estarás preparado para reiniciar tu ordenador y entrar a tu nuevo sistema LFS.

Este es el proceso en pocas palabras. La información detallada sobre todos los pasos a dar se exponen en los capítulos y secciones a medida que avanzas en ellos. Si algo no está muy claro ahora, no te preocupes, pronto todo quedará en su sitio.

Por favor, lee atentamente el Capítulo 4[p.22], pues en él se explican varias cosas importantes que deberías tener en cuenta antes de que empieces a trabajar en el Capítulo 5[p.28] y los siguientes.

Historial de modificaciones

Esta es la versión FINAL del día 19 de Agosto de 2004 de la traducción al castellano de la versión 5.1.1 del libro Linux From Scratch publicado el 5 de Junio de 2004. Si este libro tiene más de tres meses de antigüedad es probable que haya disponible una versión más nueva y mejor. Para averiguarlo comprueba uno de los servidores alternativos listados en http://www.linuxfromscratch.org/.

A continuación hay una lista con los cambios realizados desde la anterior versión del libro, primero un resumen y después un registro detallado.

- Actualizado a:
  - autoconf-2.59
  - automake-1.8.4
  - coreutils-5.2.1
  - e2fsprogs-1.35
  - expect-5.41.0
  - file-4.09
  - gcc-3.3.3
  - gettext-0.14.1
  - glibc-2.3.3-lfs-5.1
  - grub-0.94
  - kbd-1.12
  - less-382
  - lfs-bootscripts-2.0.5
  - libtool-2.5.6
  - linux-2.4.26
  - man-pages-1.66
  - modutils-2.4.27
  - ncurses-5.4
  - perl-5.8.4
  - procps-3.2.1
  - psmisc-21.4
  - sed-4.0.9
  - shadow-4.0.4.1
  - tar-1.13.94
  - tcl-8.4.6
  - texinfo-4.7
  - util-linux-2.12a
- Añadidos:
  - iana-etc-1.00
  - inetutils-1.4.2-no_server_man_pages-1.patch
  - make_devices-1.2

    - mktemp-1.5 + mktemp-1.5-add-tempfile.patch

- Eliminados:

    - gcc-3.3.1-suppress-libiberty.patch
    - lfs-utils-0.5
    - MAKEDEV-1.7
    - man-1.5m2-manpath.patch
    - man-1.5m2-pager.patch
    - ncurses-5.3-etip-2.patch
    - ncurses-5.3-vsscanf.patch
    - perl-5.8.0-libc-3.patch
    - procps-3.1.11-locale-fix.patch
    - shadow-4.0.3-newgrp-fix.patch
    - zlib-1.1.4-vsnprintf.patch

- 2 de Junio de 2004 [matt]: Prólogo - Agradecimientos, añadido Thomas Reitelbach como traductor al alemán.
- 30 de Mayo de 2004 [matt]: Capítulo 6 - vim, corregido el comando opcional para invocar el banco de pruebas.
- 23 de Mayo de 2004 [matt]: Capítulo 6 - kbd, eliminada la ruta fija al directorio de las fuentes del núcleo.
- 19 de Mayo de 2004 [matt]: Capítulo 6 - mktemp, añadida la instrucción para instalar el envoltorio tempfile.
- 18 de Mayo de 2004 [manuel]: Capítulo 3 - Actualizada la lista de servidores del paquete Glibc. Corregidos varios errores textuales.
- 17 de Mayo de 2004 [winkie]: Capítulo 5 - Añadido “AUTOCONF=no” a la construcción de Glibc. Esto evita que autoconf nos cause problemas.
- 16 de Mayo de 2004 [jeremy]: Capítulo 9 - Añadido un breve párrafo en la página de reinicio del sistema para hablar sobre los paquetes que podría ser útil añadir antes de arrancar el nuevo sistema.
- 15 de Mayo de 2004 [matt]: Capítulo 6 - Añadido un claro aviso sobre lo necesario de personalizar make_devices.
- 14 de Mayo de 2004 [matt]: Capítulo 3 - Añadido el md5sum de glibc
- 14 de Mayo de 2004 [matt]: Capítulos 5 y 6 - Actualizado a glibc-2.3.3-lfs-5.1
- 11 de Mayo de 2004 [jeremy]: Prólogo - Actualizada la lista del personal activo en el proyecto.
- 9 de Mayo de 2004 [winkie]: Capítulo 6 - Eliminadas las entradas no utilizables o inservibles de `nsswitch.conf`.
- 7 de Mayo de 2004 [matt]: Añadidos los parches para lfs-xsl-0.9 de Manuel.
- 7 de Mayo de 2004 [matt]: Corregido un error en el README relativo a la invocación de `make`.
- 3 de Mayo de 2004: Publicada la versión LFS 5.1-pre2.
- 2 de Mayo de 2004 [matt]: Añadidas comillas en los comandos de chroot del Capítulo 6 (bug #818).
- 2 de Mayo de 2004 [matt]: Eliminada de la sección de estructura del prólogo la descripción de la ya no existente parte IV.
- 1 de Mayo de 2004 [jeremy]: Añadida la creación de los directorios /media y /srv, así como 2 directorios bajo /media para disquete y cdrom, como indica el FHS - Corrige los bugs #785 y #819.
- 14 de Abril de 2004 [jeremy]: Actualizado a lfs-bootscripts-2.0.3, no necesita cambios en el texto.
- 24 de Marzo de 2004 [jeremy]: Capítulo 7 - Actualizado al nuevo lfs-bootscripts-2.0.2, y todos los cambios necesarios en la configuración de los guiones de arranque.
- 21 de Febrero de 2004 [winkie]: Capítulo 6 - Sustituido Lfs-Utils por Iana-Etc y Mktemp.

- 27 de Febrero de 2004 [jeremy]: Actualizado a Procps-3.2.0
- 27 de Febrero de 2004 [jeremy]: Actualizado a Lfs-utils-0.5, corrige un posible ataque de enlace en iana-get.
- 27 de Febrero de 2004 [jeremy]: Alteradas las instrucciones de Findutils en el Capítulo 6 para cumplir el FHS.
- 26 de Febrero 2004 [jeremy]: Eliminada la creación del directorio /usr/etc para cumplir el FHS - Cerrado el bug 775.
- 26 de Febrero de 2004 [jeremy]: Actualizado a Linux-2.4.25.
- 23 de Febrero de 2004 [alex]: Capítulos 6 + 9 - Limpiadas las secciones sobre Revisión del chroot y Reiniciar.
- 22 de Febrero de 2004 [alex]: Movida la eliminación de símbolos del sistema final del Capítulo 9 al Capítulo 6.
- 22 de Febrero de 2004 [alex]: Capítulo 6 - Coreutils y E2fsprogs: Clarificados los prerrequisitos para ejecutar las pruebas.
- 19 de Febrero de 2004 [alex]: Capítulo 5 - Eliminación de Símbolos: Eliminado un “{,share/}” innecesario del comando **rm** de la documentación.
- 14 de Febrero de 2004 [jeremy]: Capítulo 6 - Actualizado a Less-382
- 14 de Febrero de 2004 [jeremy]: Capítulos 5 y 6 - Actualizado a Ncurses-5.4 y eliminadas las referencias al parche etip.
- 12 de Febrero de 2004 [jeremy]: Capítulo 6 - Eliminadas las rutas absolutas para los comandos pwconv y grpconv, pues /usr/sbin es parte de la ruta por defecto.
- 9 de Febrero de 2004 [alex]: Capítulo 6 - Movida la sección de instalación de lfs-bootscripts al Capítulo 7.
- 8 de Febrero de 2004 [matt]: Capítulo 6 - Actualizado a Man-pages-1.66.
- 7 de Febrero de 2004 [alex]: Capítulo 1 - Movidas las secciones de Convenciones y Agradecimientos al Prólogo.
- 7 de Febrero de 2004 [alex]: Capítulo 6 - Creación de dispositivos: Sustituido el guión MAKEDEV con el guión make_devices contribuido por Matthias Benkmann.
- 5 de Febrero de 2004 [alex]: Capítulo 6 - Simplificada la instalación final de las cabeceras del núcleo para copiarlas simplemente a partir del directorio de herramientas temporales.
- 4 de Febrero de 2004 [alex]: Capítulos 5 + 6 - Movido el montaje de proc y devpts a antes de entrar al chroot, eliminado Util-linux de las herramientas y añadido un pequeño guión arch para Perl.

Liberada la versión 5.1-pre1 el 1 de Febrero de 2004.

Recursos

FAQ

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

En http://www.escomposlinux.org/lfs-es/faq tienes una versión en castellano, aunque en el momento de la publicación de esta versión del libro se encontraba algo desfasada.

IRC

Varios miembros de la comunidad LFS ofrecen asistencia técnica en nuestro servidor IRC. Antes de utilizar este método de ayuda te pedimos que al menos compruebes si en las FAQ de LFS (ver arriba) o en los archivos de las listas de correo está la respuesta a tu problema. Puedes encontrar el servidor IRC en el puerto 6667 de *irc.linuxfromscratch.org* o *irc.linux-phreak.net*. El canal de soporte se llama #LFS-support.

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 ajenas 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

Servidor de noticias

Todas 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*.

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/. Tu también puedes añadir información para ayudar a otros.

Referencias

Si aún necesitas información más detallada sobre los paquetes, en esta página encontrarás apuntes útiles: http://www.109bean.org.uk/LFS-references.html.

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

- Cervera, España [126 Kbits] - http://www.macana-es.com
- Dattatec.com, Argentina [100 Mbits] - http://www.lfs-es.org

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).

Cómo buscar ayuda

Si tienes algún problema usando este libro, deberías consultar primero las FAQ (en castellano en http://www.escomposlinux.org/lfs-es/faq, y en inglés en http://www.linuxfromscratch.org/faq/), muy probablemente tu pregunta ya esté contestada aquí. Si no es así, deberías probar 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.

Si todo esto falla, encontrarás que mucha gente gente en el IRC y en las listas de correo (mira Capítulo 1 - Listas de correo[p.6]) estará dispuesta a ayudarte. Para ayudarles a diagnosticar y resolver tu problema, incluye toda la información relevante que sea posible en tu petición de ayuda.

Cosas que debes mencionar

Además de una breve explicación del problema que estás teniendo, las cosas esenciales que debes incluir en tu petición de ayuda son:

- La versión del libro que estás usando (que es 5.1.1),
- La distribución anfitriona (y su versión) que estás usando como base para crear el LFS.
- El paquete o la sección que te da problemas.
- El mensaje de error exacto o los síntomas que aparecen.
- Si te has desviado o no del libro.

(Ten en cuenta que decir que te has desviado del libro no implica que no vayamos a ayudarte. Después de todo, LFS se basa en la elección. Simplemente nos ayudará a detectar otras posibles causas de tu problema)

Problemas de configuración

Cuando algo vaya mal en la fase en que 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 decides pedir ayuda.

Problemas de compilación

Para ayudarnos a determinar la causa del problema, nos va a ser útil tanto la salida del terminal como el contenido de varios ficheros. Las salidas a terminal del guión ./configure y del comando make pueden ser útiles. No incluyas ciegamente todo el contenido pero, por otro lado, no incluyas demasiado poco. Por ejemplo, aquí hay parte de una salida a terminal 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 de la sección anterior desde donde pone

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

hasta el final. Esto no nos basta 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.

Hay un artículo excelente sobre cómo buscar ayuda en Internet, escrito por Eric S. Raymond. Está disponible en http://catb.org/~esr/faqs/smart-questions.html. Lee y sigue los consejos de este documento y tendrás muchas más posibilidades de obtener una respuesta, y también de que obtengas la ayuda que necesitas.

Problemas en los bancos de pruebas

Muchos paquetes proporcionan un banco de pruebas que, dependiendo de la importancia del paquete, te animaremos a ejecutar. 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 lo hemos investigado y anotado. Si nosotros ya sabemos de él, normalmente no hay necesidad de preocuparse.

Parte II. Preparativos para la construcción

Capítulo 2. Preparar una nueva partición

Introducción

En este capítulo se preparará la partición que contendrá el sistema LFS. Crearemos la partición, haremos un sistema de ficheros en ella, y la montaremos.

Crear una nueva partición

Para construir nuestro nuevo sistema Linux necesitaremos espacio: una partición de disco vacía. Si no tienes una partición libre, y no tienes sitio en ninguno de tus discos duros para crear una, entonces puedes construir LFS en la misma partición en la que tienes instalada tu distribución actual. Este proceso no es recomendable para tu primera instalación de LFS, pero si andas escaso de espacio en disco y te sientes valiente, echa un vistazo a la receta http://www.escomposlinux.org/lfs-es/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).

Para un sistema mínimo necesitas una partición de 1,3 GB más o menos. Esto es suficiente para almacenar todos los archivos de código fuente y compilar todos los paquetes. Pero si piensas usar el sistema LFS como tu sistema Linux principal, seguramente querrás instalar software adicional y necesitarás más espacio, posiblemente unos 2 o 3 GB.

Como casi nunca tenemos suficiente memoria RAM en nuestra máquina, 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 las cosas urgentes. La partición de intercambio para tu sistema LFS puede ser la misma que la de tu sistema anfitrión, por lo que no tienes que crear otra si tu sistema anfitrión ya utiliza una partición de intercambio.

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, que será algo como `hda5`. En este libro nos referiremos a ella como la partición LFS. Si (ahora) tienes además una partición de intercambio, recuerda también su denominación. Estos nombres se necesitarán posteriormente para el fichero `/etc/fstab`.

Crear un sistema de ficheros en la nueva partición

Ahora que tenemos una partición en blanco, podemos crear un sistema de ficheros en ella. El más usado en el mundo de Linux es el "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 crearemos un sistema de ficheros ext2, sin embargo encontrarás las instrucciones para otros sistemas de ficheros en http://www.escomposlinux.org/lfs-es/blfs-es-CVS/postlfs/filesystems.html (la versión original la tienes en http://www.linuxfromscratch.org/blfs/view/stable/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 (algo como `hda5`).

Si creas una (nueva) partición de intercambio (swap), también necesitas inicializarla (también conocido como formatearla, como hiciste anteriormente con **mke2fs**) ejecutando:

```
mkswap /dev/yyy
```

Sustituye `yyy` por el nombre de la partición de intercambio.

Montar la nueva partición

Ahora que hemos creado un sistema de ficheros, queremos poder acceder a la partición. Para esto necesitamos montarla y debemos elegir un punto de montaje. En este libro asumimos que el sistema de ficheros se monta en `/mnt/lfs`, pero no importa el directorio que elijas.

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

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

Ahora 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 has decidido usar múltiples particiones para LFS (digamos que una para `/` y otra para `/usr`), móntalas de esta forma:

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

Por supuesto, sustituye `xxx` e `yyy` por los nombres de partición apropiados.

Deberías asegurarte que esta nueva partición no se monte con permisos muy restrictivos (como las opciones nosuid, nodev o noatime). Puedes usar 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 nos hemos hecho un sitio en el que trabajar, estamos preparados para descargar los paquetes.

Capítulo 3. Los materiales: paquetes y parches

Introducción

A continuación se muestra una lista con los paquetes que necesitas 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. A no ser que seas un experimentado constructor de LFS, te recomendamos encarecidamente que no pruebes con nuevas versiones, pues los comandos de construcción para una versión puede que no funcionen con otra más nueva. Igualmente, con frecuencia hay razones para no usar la última versión debido a problemas conocidos que aún no han podido solucionarse.

Todas las URLs, cuando es posible, apuntan a la página del proyecto 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.

No podemos garantizar que estas localizaciones de descarga estén siempre disponibles. En el caso de que una localización de descarga haya cambiado desde la publicación de este libro, prueba a buscar el paquete en google. Si no consigues resultados con esto, puedes consultar la página de erratas del libro en http://www.linuxfromscratch.org/lfs/print o, mejor aún, probar uno de los métodos alternativos de descarga listados en http://www.linuxfromscratch.org/lfs/packages.html.

Necesitarás guardar todos los paquetes y parches necesarios en algún sitio que esté disponible durante toda la construcción. También necesitarás un directorio de trabajo en el que desempaquetar las fuentes y construirlas. Un esquema que funciona bien es utilizar `$LFS/sources` para almacenar los paquetes y parches *y* como directorio de trabajo. De esta forma todo lo que necesitas se encontrará en la partición LFS y estará disponible durante todas las fases del proceso de construcción.

Puede que quieras ejecutar, como usuario *root* el siguiente comando antes de comenzar la sesión de descarga:

```
mkdir $LFS/sources
```

Y haz que tu usuario normal pueda escribir en este directorio (y activa también el bit sticky del mismo) pues, como suponemos, no realizarás la descarga como usuario *root*:

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

Todos los paquetes

Descarga u obtén por otros métodos los siguientes paquetes:

Autoconf (2.59) - 903 KB:
http://freshmeat.net/projects/autoconf/

Automake (1.8.4) - 644 KB:
http://freshmeat.net/projects/automake/

Bash (2.05b) - 1.910 KB:
http://freshmeat.net/projects/gnubash/

Binutils (2.14) - 10.666 KB:
http://freshmeat.net/projects/binutils/

Bison (1.875) - 796 KB:
http://freshmeat.net/projects/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/

Ed (0.2) - 182 KB:
http://freshmeat.net/projects/ed/

Expect (5.41.0) - 510 KB:
http://freshmeat.net/projects/expect/

File (4.09) - 356 KB: -- *(ver Nota 1 al pié de página.)*
http://freshmeat.net/projects/file/

Findutils (4.1.20) - 760 KB:
http://freshmeat.net/projects/findutils/

Flex (2.5.4a) - 372 KB:
ftp://ftp.gnu.org/gnu/non-gnu/flex/

Gawk (3.1.3) - 1.596 KB:
http://freshmeat.net/projects/gnuawk/

GCC (2.95.3) - 9.618 KB:
http://freshmeat.net/projects/gcc/

GCC-core (3.3.3) - 11.283 KB:
http://freshmeat.net/projects/gcc/

GCC-g++ (3.3.3) - 2.026 KB:
http://freshmeat.net/projects/gcc/

GCC-testsuite (3.3.3) - 1.051 KB:
http://freshmeat.net/projects/gcc/

Gettext (0.14.1) - 6.397 KB:
http://freshmeat.net/projects/gettext/

Glibc (2.3.3-lfs-5.1) - 13.101 KB: -- *(ver Nota 2 al pié de página.)*
http://freshmeat.net/projects/glibc/

Grep (2.5.1) - 545 KB:
http://freshmeat.net/projects/grep/

Groff (1.19) - 2.360 KB:
http://freshmeat.net/projects/groff/

Grub (0.94) - 902 KB:
ftp://alpha.gnu.org/pub/gnu/grub/

Gzip (1.3.5) - 324 KB:
ftp://alpha.gnu.org/gnu/gzip/

Iana-Etc (1.00) - 161 KB:
http://freshmeat.net/projects/iana-etc/

Inetutils (1.4.2) - 1.019 KB:
http://freshmeat.net/projects/inetutils/

Kbd (1.12) - 617 KB:
http://freshmeat.net/projects/kbd/

Less (382) - 259 KB:
http://freshmeat.net/projects/less/

LFS-Bootscripts (2.0.5) - 32 KB:
http://downloads.linuxfromscratch.org/

Libtool (1.5.6) - 2.602 KB:
http://freshmeat.net/projects/libtool/

Linux (2.4.26) - 30.051 KB:
http://freshmeat.net/projects/linux/

M4 (1.4) - 310 KB:
http://freshmeat.net/projects/gnum4/

Make (3.80) - 899 KB:
http://freshmeat.net/projects/gnumake/

Make_devices (1.2) - 20 KB:
http://downloads.linuxfromscratch.org/

Man (1.5m2) - 196 KB:
http://freshmeat.net/projects/man/

Man-pages (1.66) - 1.582 KB:
http://freshmeat.net/projects/man-pages/

Mktemp (1.5) - 69 KB:
http://freshmeat.net/projects/mktemp/

Modutils (2.4.27) - 229 KB:
http://freshmeat.net/projects/modutils/

Ncurses (5.4) - 2.019 KB:
http://freshmeat.net/projects/ncurses/

Net-tools (1.60) - 194 KB:
http://freshmeat.net/projects/net-tools/

Patch (2.5.4) - 182 KB:
http://freshmeat.net/projects/patch/

Perl (5.8.4) - 9.373 KB:
http://freshmeat.net/projects/perl/

Procinfo (18) - 24 KB:
http://freshmeat.net/projects/procinfo/

Procps (3.2.1) - 260 KB:
http://freshmeat.net/projects/procps/

Psmisc (21.4) - 375 KB:
http://freshmeat.net/projects/psmisc/

Sed (4.0.9) - 751 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 (1.13.94) - 1.025 KB:
ftp://alpha.gnu.org/gnu/tar/

Tcl (8.4.6) - 3.363 KB:
http://freshmeat.net/projects/tcltk/

Texinfo (4.7) - 1.385 KB:
http://freshmeat.net/projects/texinfo/

Util-linux (2.12a) - 1.814 KB:
http://freshmeat.net/projects/util-linux/

Vim (6.2) - 3.193 KB:
http://freshmeat.net/projects/vim/

Zlib (1.2.1) - 277 KB:
http://freshmeat.net/projects/zlib/

Tamaño total de estos paquetes: 134 MB

Nota

1) File (4.09) puede que no esté disponible cuando leas esto. Es sabido que en ocasiones los administradores de la localización principal de descarga eliminan las antiguas versiones cuando se libera una nueva. En esta localización de descarga alternativa puede que haya versiones antiguas disponibles: ftp://gaosu.rave.org/pub/linux/lfs/.

Nota

2) Cuando se escribió esto, los desarrolladores de Glibc habian decidido no poner paquetes de las nuevas versiones disponibles para descarga. Por tanto, el equipo de LFS responsable de las herramientas ha suministrado un paquete de las fuentes de Glibc extraido del CVS de Glibc y ha generado un paquete con ellas, incluyendo los parches que fuesen necesarios.

Este paquete está disponible por cortesía de los generosos sitios de réplica de LFS:

ftp://gaosu.rave.org/pub/linux/lfs/packages/conglomeration/glibc-2.3.3-lfs-5.1.tar.bz2
ftp://lfs.mirror.intermedia.com.sg/pub/lfs/lfs-packages/conglomeration/glibc-2.3.3-lfs-5.1.tar.bz2
http://packages.lfs-es.org/glibc/glibc-2.3.3-lfs-5.1.tar.bz2
http://mirror.averse.net/lfs-packages/glibc-2.3.3-lfs-5.1.tar.bz2
ftp://mirror.averse.net/pub/lfs-packages/glibc-2.3.3-lfs-5.1.tar.bz2
ftp://ftp.lfs-matrix.de/lfs-packages/conglomeration/glibc-2.3.3-lfs-5.1.tar.bz2
ftp://ftp.sg.linuxfromscratch.org/pub/lfs-packages/glibc-2.3.3-lfs-5.1.tar.bz2
http://ftp.sg.linuxfromscratch.org/glibc-2.3.3-lfs-5.1.tar.bz2

Si deseas comprobar la integridad del paquete, su resumen MD5 es `cd11fabdf5162ad68329e7b28b308278`, el cual puede verificarse usando **md5sum**.

Parches necesarios

Aparte de todos esos paquetes, también necesitarás varios parches. Estos corrigen pequeños errores en los paquetes que debería solucionar su desarrollador, o simplemente hacen pequeñas modificaciones para adaptarlos a nuestras necesidades. Necesitarás los siguientes:

Parche para Bash - 7 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/bash-2.05b-2.patch

Parche Attribute para Bison - 2 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/bison-1.875-attribute.patch

Parche Hostname para Coreutils - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/coreutils-5.2.1-hostname-1.patch

Parche Uname para Coreutils - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/coreutils-5.2.1-uname-1.patch

Parche Mkstemp para Ed - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/ed-0.2-mkstemp.patch

Parche Spawn para Expect - 6 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/expect-5.41.0-spawn-1.patch

Parche No-Fixincludes para GCC - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/gcc-3.3.3-no_fixincludes-1.patch

Parche Specs para GCC - 11 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/gcc-3.3.3-specs-1.patch

Parche para GCC-2 - 16 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/gcc-2.95.3-2.patch

Parche No-Fixincludes para GCC-2 - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/gcc-2.95.3-no-fixinc.patch

Parche Return-Type para GCC-2 - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/gcc-2.95.3-returntype-fix.patch

Parche No-Server-Man-Pages para Inetutls - 4 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/inetutils-1.4.2-no_server_man_pages-1.patch

Parche More-Programs para Kbd - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/kbd-1.12-more-programs-1.patch

Parche 80-Columns para Man - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/man-1.5m2-80cols.patch

Parche Tempfile para Mktemp - 3 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/mktemp-1.5-add-tempfile.patch

Parche Mii-Tool-Gcc33 para Net-tools - 2 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/net-tools-1.60-miitool-gcc33-1.patch

Parche Libc para Perl - 1 KB:
http://www.linuxfromscratch.org/patches/lfs/cvs/stable/perl-5.8.4-libc-1.patch

Aparte de los anteriores parches necesarios, hay una serie de parches opcionales creados por la comunidad LFS. Muchos de ellos 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 desees utilizar.

Capítulo 4. Últimos preparativos

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 nuestro ejemplo. Si la salida es errónea, siempre puedes establecer de nuevo la variable con:

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

Tener establecida esta variable significa que si se te indica que ejecutes un comando como **`mkdir $LFS/tools`**, puedes teclearlo literalmente. Tu intérprete de comandos sustituirá “$LFS” con “/mnt/lfs” (o aquello a lo que hayas establecido la variable) cuando procese la línea de comandos.

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

Creación del directorio $LFS/tools

Todos los programas que compilamos en el Capítulo 5[p.28] se instalarán bajo`$LFS/tools` para mantenerlos separados de los programas compilados en el Capítulo 6[p.69]. Los programas compilados aquí son sólo herramientas temporales y no formarán parte del sistema LFS final. Mantenerlos en un directorio aparte hace que más adelante podamos borrarlos fácilmente. Esto también ayuda a evitar que acaben en los directorios de producción de tu anfitrión (fácil de hacer en el Capítulo 5[p.28]), lo que podría ser muy mal asunto.

Puede que más tarde quieras explorar los binarios de tu sistema para ver de qué ficheros hacen uso o con cuales están enlazados. Para facilitar esta búsqueda puede que desees elegir un nombre inequívoco para el directorio en el que se guardan las herramientas temporales. En lugar del simple “tools” puedes usar algo como “herramientas-para-lfs”. Sin embargo, deberás tener cuidado de adaptar todas las referencias a “tools” a través del libro, incluidas aquellas que se encuentren en los parches, siendo el más destacable el parche Specs para GCC.

Crea el directorio necesario ejecutando lo siguiente:

```
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:

```
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 info antes de informar de lo que puedes pensar que es un error.

El enlace simbólico creado posibilita que compilemos nuestro conjunto de herramientas refiriéndonos siempre 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 próximo (cuando hagamos chroot a la partición LFS).

Añadir el usuario lfs

Si trabajas como *root*, un simple error puede dañar o incluso arruinar tu sistema. Por tanto te recomendamos que construyas los paquetes en este capítulo como un usuario sin privilegios. Por supuesto, puedes usar tu propio nombre de usuario, pero para asegurar un entorno de trabajo limpio crearemos un nuevo usuario llamado *lfs* y lo utilizaremos durante el proceso de instalación. Como *root*, ejecuta el siguiente comando para añadir el nuevo usuario:

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

El significado de las opciones:

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

Si quieres poder entrar como *lfs*, asígnale una contraseña al nuevo usuario:

```
passwd lfs
```

y concede al nuevo usuario *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*.

Configuración del entorno

Vamos a establecer un buen entorno de trabajo mediante la creación de dos nuevos ficheros de inicio para el intérprete de comandos **bash**. Mientras estás 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
```

Normalmente, cuando entras como usuario *lfs* el intérprete de comandos inicial es un intérprete *de ingreso* que lee `/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 último fichero sustituye el intérprete de comandos en ejecución por uno nuevo con un entorno completamente vacío, excepto para las variables HOME, TERM y PS1. Esto asegura que en nuestro entorno de construcción no aparezcan variables de entorno indeseadas o dañinas procedentes del sistema anfitrión. La técnica aquí usada es algo extraña, pero consige el objetivo de forzar 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 este último fichero:

```
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, nosotros queremos que las nuevas herramientas se utilicen tan pronto como las instalemos. Al desactivar esta característica, nuestros comandos "interactivos" (**make**, **patch**, **sed**, **cp**, etc) siempre usarán las versiones más nuevas durante el proceso de construcción.

Establecer la máscara de creación de ficheros a 022 nos asegura que los ficheros y directorios de nueva creación sólo pueden ser escritos por su propietario, pero legibles y ejecutables por cualquiera.

Por supuesto, la variable LFS deberás establecerla con el punto de montaje que hayas elegido.

La variable LC_ALL controla la localización de ciertos programas, haciendo que sus mensajes sigan las convenciones para un determinado país. Si tu 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 este capítulo puede causar problemas si sales del entorno chroot e intentas regresar más tarde. Estableciendo LC_ALL a "POSIX" (o su equivalente "C") nos aseguramos de que todo funcionará como se espera dentro del entorno chroot.

Añadimos `/tools/bin` al PATH para que, al avanzar en este capítulo, las herramientas que vamos instalando se vayan usando en el resto del proceso de construcción.

Finalmente, para tener todo preparado para construir nuestras herramientas temporales, carga el perfil recién creado:

```
source ~/.bash_profile
```

Sobre los SBUs

Bastante gente desea saber de antemano cuanto tiempo, aproximadamente, le llevará compilar e instalar cada paquete. Pero “Linux From Scratch” se construye sobre muchos sistemas diferentes, siendo imposible dar tiempos reales y precisos: el paquete más grande (Glibc) no tarda más de veinte minutos en un sistema rápido, pero puede tardar tres días en uno lento (no es broma). Así que en vez de dar tiempos reales hemos adoptado la idea de usar la *Static Binutils Unit (Unidad de Binutils Estático*, o abreviado, *SBU*).

Funciona de esta forma: el primer paquete que compilas en este libro es, en el Capítulo 5[p.28], Binutils enlazado estáticamente, y el tiempo que tarde en compilar este paquete es lo que llamamos “Unidad de Binutils Estático” o “SBU”. Todos los demás tiempos de compilación se expresarán relativamente a este tiempo.

Por ejemplo, considera un paquete en particular cuyo tiempo de compilación es de 4,4 SBUs. Esto significa que, si en tu sistema, el tiempo que se tarda en compilar e instalar el Binutils estático es de 10 minutos, sabes que 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, 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-3.3.3 que se sabe que es aproximadamente un 30% más lento.

Ten en cuenta también que los SBUs no funcionan bien en máquinas basadas en SMP (Multi-Procesadores Simétricos). Pero si eres tan afortunado de tener un multiprocesador, tienes la suerte de que tu sistema será tan rápido que eso no importe.

Si deseas ver tiempos reales para máquinas concretas, mira http://www.linuxfromscratch.org/~bdubbs/.

Sobre los bancos de pruebas

Muchos paquetes proporcionan un banco de pruebas. Ejecutar el banco de pruebas para un paquete recién construido es, generalmente, una buena idea, pues puede proporcionar una buena comprobación de que todo se ha compilado correctamente. Un banco de pruebas superado 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. Pero ten cuidado, los bancos de pruebas para GCC y Glibc pueden tardar bastante tiempo en completarse, sobre todo en hardware lento.

La experiencia nos ha mostrado que se gana poco ejecutando los bancos de pruebas en el Capítulo 5[p.28]. No hay escape del hecho de que el sistema anfitrión siempre ejerce su influencia sobre las pruebas en dicho capítulo, causando con frecuencia fallos raros e inexplicables. No solo eso, las herramientas construidas en el Capítulo 5[p.28] son temporales y descartables. A los lectores de este libro les recomendamos que *no* ejecuten los bancos de pruebas durante el Capítulo 5[p.28]. Las instrucciones para ejecutarlos se suministran todavía para el provecho de los comprobadores y desarrolladores, pero son estrictamente opcionales para el resto.

Un problema común al ejecutar los bancos de pruebas de Binutils y GCC es quedarse sin seudo-terminales (PTYs para abreviar). 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. Más adelante, en el Capítulo 5[p.28], trataremos este tema con mayor detalle.

En ocasiones los bancos de pruebas de los paquetes muestran falsos fallos. Puesdes consultar el Wiki de LFS en http://wiki.linuxfromscratch.org/ para consultar si estos fallos son normales. Esto se aplica a todas las pruebas del libro.

Capítulo 5. Construcción del sistema temporal

Introducción

En este capítulo compilaremos e instalaremos un sistema Linux mínimo. Este sistema contendrá sólo las herramientas necesarias para poder iniciar la construcción del sistema LFS definitivo en el siguiente capítulo, permitiendo un entorno de trabajo algo más amigable para el usuario que un entorno mínimo.

La construcción de este sistema minimalista se hará en dos etapas: primero construiremos un conjunto de herramientas independiente del sistema anfitrión (compilador, ensamblador, enlazador, librerías y unas pocas herramientas útiles), y después las usaremos 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, se espera que ya lo hayas desempaquetado (lo que se explicará pronto) como usuario *lfs* y hayas hecho 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 cuando parezca que hemos olvidado 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 muchos 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, lo que no es realmente un problema.

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. No te pierdas los recordatorios.

Notas técnicas sobre las herramientas

Esta sección intenta explicar 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. Eres libre de volver aquí en cualquier momento.

El principal objetivo del Capítulo 5[p.28] es proporcionar un entorno temporal sano 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[p.?]. 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*, que no debe confundirse con el enlazador estándar *ld* que es parte de Binutils. El enlazador dinámico lo suministra Glibc y su trabajo es encontrar y cargar las librerías compartidas necesarias para un programa, preparar el programa y ejecutarlo. Para la mayoría el nombre del enlazador dinámico será *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. Debes poder determinar el nombre del enlazador dinámico de tu plataforma 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[p.28]:

- 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 las librerías 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 Glibc incorrectamente configurado 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 su ensamblador y su enlazador en dos ubicaciones, `/tools/bin` y `/tools/$TARGET_TRIPLET/bin`. En realidad, 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: **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. Puedes saber cuál es el enlazador estándar que utilizará **gcc** ejecutando: **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[p.28] se construirán todos contra la nueva Glibc en `/tools` como debe ser.

Tras entrar en el entorno chroot en el Capítulo 6[p.69], 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, entonces procederemos a la construcción real del sistema LFS.

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.

Binutils-2.14 - Fase 1

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

```
Tiempo estimado de construcción:  1.0 SBU
Espacio requerido en disco:       170 MB
```

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

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 paquete se comporta mal si cambias sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Por tanto, si tienes definida cualquier variable de entorno que sobreescriba las optimizaciones por defecto, como CFLAGS y CXXFLAGS, te recomendamos que las desactives cuando construyas Binutils.

La documentación de Binutils recomienda construirlo en un directorio dedicado, fuera del directorio 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, tendrás que medir el tiempo que se tarda en construir este paquete, desde la compilación hasta la instalación. Para ello, envuelve los comandos dentro de un comando **time** de esta forma: **`time { ./configure ... && ... && ... && make install; }`**.

A continuación, prepara Binutils para su compilación:

```
../binutils-2.14/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 (también conocida como i18n), ya que no es necesaria para nuestros programas estáticos y *nls* suele causar problemas con el enlazado estático.

Continúa compilando 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. Y de todas formas no tendría sentido ejecutar las pruebas, pues los programas de esta primera fase pronto serán sustituidos por los de la segunda.

Instala el paquete:

```
make install
```

Ahora prepara al 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 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. Verás cómo estos preparativos se utilizan más tarde en este capítulo.

Aviso

No borres todavía 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 “Contenido de Binutils”[p.88].

GCC-3.3.3 - 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:       411.7 MB
```

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

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 paquete se comporta mal si cambias sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Por tanto, si tienes definida cualquier variable de entorno que sobreescriba las optimizaciones por defecto, como CFLAGS y CXXFLAGS, te recomendamos que las desactives cuando construyas GCC.

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

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

Prepara GCC para su compilación:

```
../gcc-3.3.3/configure --prefix=/tools \
    --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, intentamos minimizar la influencia del sistema anfitrión, así que esto es algo lógico de hacer.
- **`--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.

Continúa compilando 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, lo cual significa que es muy probable que se haya compilado correctamente.

La compilación se ha completado y 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. Y de todas formas no tendría sentido ejecutar las pruebas, pues los programas de esta primera fase pronto serán sustituidos.

Ahora instala el paquete:

```
make install
```

Como toque final crearemos 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 “Contenido de GCC”[p.90].

Cabeceras de Linux-2.4.26

```
Tiempo estimado de construcción:  0.1 SBU
Espacio requerido en disco:       192.5 MB
```

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 symlinks
```

Instala los ficheros de cabecera específicos de la plataforma:

```
mkdir /tools/include/asm
cp include/asm/* /tools/include/asm
cp -R include/asm-generic /tools/include
```

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

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

Glibc-2.3.3-lfs-5.1

El paquete Glibc contiene la librería C principal. Esta librería proporciona todas las rutinas básicas para la ubicación de memoria, busqueda 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:       734.2 MB
```

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

Instalación de Glibc

Se sabe que este paquete se comporta mal si cambias sus parámetros de optimización (incluyendo las opciones *-march* y *-mcpu*). Por lo tanto, si tienes definida cualquier variable de entorno que sobreescriba las optimizaciones por defecto, como CFLAGS y CXXFLAGS, te recomendamos que las desactives cuando construyas Glibc.

Básicamente, 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 directorio de las fuentes, en un directorio de construcción dedicado:

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

A continuación, prepara Glibc para su compilación:

```
../glibc-2.3.3-lfs-5.1/configure --prefix=/tools \
    --disable-profile --enable-add-ons=linuxthreads \
    --with-binutils=/tools/bin --with-headers=/tools/include \
    --without-gd --without-cvs
```

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=linuxthreads`**: Esto le indica a Glibc que utilice el añadido Linuxthreads como su librería de hilos.
- **`--with-binutils=/tools/bin`** y **`--with-headers=/tools/include`**: Estrictamente hablando, estas opciones no son necesarias, pero nos aseguran que nada vaya mal con respecto a las cabeceras del núcleo y los programas de Binutils que se usen durante la construcción de Glibc.
- **`--without-gd`**: Esto evita la construcción del programa **memusagestat**, el cual insiste extrañamente 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, pero no es realmente necesario en estos momentos. Lo usamos porque silencia un molesto pero inofensivo aviso sobre la ausencia del programa **autoconf**.

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.
*** Compruebe 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.

Compila el paquete:

```
make AUTOCONF=no
```

La compilación está completa. Como se mencionó 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 Glibc, hazlo con el siguiente comando:

```
make check
```

El banco de pruebas de Glibc depende en gran medida de ciertas funciones de tu sistema anfitrión, en particular del núcleo. Adicionalmente, en este capítulo algunas pruebas pueden verse afectadas adversamente por las herramientas existentes o el entorno del sistema anfitrión. Por supuesto, esto no será un problema cuando ejecutes el banco de pruebas de Glibc dentro del entorno chroot en el Capítulo 6[p.69]. En general, se espera que el banco de pruebas de Glibc pase siempre con éxito. Sin embargo, como se menciona anteriormente, bajo ciertas circunstancias algunos fallos son inevitables. Aquí hay una lista con las cuestiones más comunes a tener en cuenta:

- 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* o debido a otras rarezas del sistema de ficheros.
- 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`, 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.

En resumen, no te preocupes demasiado si ves fallos en el banco de pruebas de Glibc en este capítulo. La Glibc del Capítulo 6[p.69] es la que acabaremos usando al final, por lo que es la que realmente queremos ver pasar las pruebas (pero incluso ahí puede 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**, pero si lo haces, asegúrate de registrar la salida para que más tarde puedas revisar el fichero de registro y examinar el número total de errores.

Aunque es un mensaje inofensivo, la fase de instalación de Glibc se quejará de la ausencia del fichero `/tools/etc/ld.so.conf`. Evita este confuso aviso con:

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

Ahora instala el paquete:

```
make install
```

Nota

Nota específica de la traducción: Se ha comprobado que, bajo ciertos anfitriones y en condiciones no muy claras, el anterior comando puede fallar con en error similar a este:

```
make install
...
autoconf sysdeps/i386/elf/configure.in > sysdeps/i386/elf/configure.new
autoconf: Undefined macros
***BUG in Autoconf--please report** AC_FD_MSG
make[1] *** [sysdeps/i386/elf/configure] Error 1
make[1]: Leaving directory '/mnt/lfs/sources/glibc-2.3.3-lfs-5.1'
make: [all] Error 2
```

Si esto sucediese, repite la instalación sustituyendo el anterior comando por:

```
make AUTOCONF=no 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
```

Una alternativa al comando anterior 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 "Contenido de Glibc"[p.82].

Ajustar las herramientas

Ahora que hemos instalado las librerías de C temporales, queremos que todas las herramientas que compilemos en el resto de este capítulo se enlacen con ellas. Para conseguirlo, tenemos que ajustar el enlazador y el fichero de specs del compilador. Alguien podría decir que esto es *"magia negra"*, pero en realidad es muy simple.

Primero, instala el enlazador ajustado (lo ajustamos al final de la primera fase de Binutils) 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, los borraste accidentalmente o no tienes acceso a ellos, no te preocupes, no todo está perdido. Sólo 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 nuestro fichero de especificaciones de GCC para que apunte al nuevo enlazador dinámico. Un simple comando sed lo hará:

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

Recomendamos que copies y pegues lo anterior en lugar de intentar escribirlo. O puedes editar el fichero specs a mano si quieres: simplemente reemplaza "/lib/ld-linux.so.2" con "/tools/lib/ld-linux.so.2". Revisa visualmente el fichero de especificaciones para verificar que los cambios se han hecho realmente.

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 "Notas técnicas sobre las herramientas"[p.29].

Por último, 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-lib/*/*/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 (con las diferencias para la plataforma sobre el nombre del enlazador dinámico).:

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

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

Advierte especialmente 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. Necesitarás investigar y revisar tus pasos para encontrar el problema y corregirlo. No hay motivo para continuar hasta que lo hayas hecho. 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 "GCC-3.3.3 - Fase 1"[p.34] y corrige el enlace simbólico. Segundo, asegúrate de que tu PATH es correcto. Puedes comprobarlo ejecutando **echo $PATH** y comprobando 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 "Configuración del entorno"[p.25]. Tercero, algo pudo ir mal en el anterior arreglo del fichero specs. En este caso, repite el arreglo del fichero specs asegurándote de cortar y pegar los comandos como se recomendó.

Una vez estés seguro de que todo está bien, borra los ficheros de prueba:

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

Tcl-8.4.6

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

```
Tiempo estimado de construcción:  0.9 SBU
Espacio requerido en disco:       22.7 MB
```

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

Instalación de Tcl

Instalamos este paquete y los dos siguientes con el único propósito de poder ejecutar los bancos de pruebas de GCC y Binutils. Instalar tres paquetes sólo para realizar comprobaciones puede parecer demasiado trabajo, pero es muy tranquilizador, si no esencial, saber que nuestras herramientas más importantes funcionan adecuadamente. Aunque los bancos de pruebas no se ejecuten en este capítulo (nosotros recomendamos no ejecutarlos), estos paquetes son todavía necesarios para los bancos de pruebas del siguiente capítulo.

Prepara Tcl para su compilación:

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

Construye el paquete:

```
make
```

Si quieres comprobar los resultados, ejecuta: **`TZ=UTC make test`**. Sin embargo, se sabe que el banco de pruebas de Tcl a veces experimenta fallos bajo ciertas condiciones del anfitrión que aún no se comprenden por completo. Por tanto, 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 ejecutan correctamente. Más adelante, en el Capítulo 7[p.164], se dará más información sobre la variable de entorno TZ.

Instala el paquete:

```
make install
```

Aviso

No borres todavía el directorio de fuentes de `tcl8.4.6`, ya que el próximo paquete necesitará sus ficheros de cabecera internos.

Ahora crea un enlace simbólico necesario:

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

Contenido de Tcl

Programas instalados: tclsh (enlace a tclsh8.4), tclsh8.4

Librería instalada: libtcl8.4.so

Descripciones cortas

tclsh8.4 es el intérprete de comandos de TCL.

libtcl8.4.so es la librería TCL.

Expect-5.41.0

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, Tcl.

Instalación de Expect

Primero, aplica un parche que puede causar falsos fallos durante la ejecución del banco de pruebas de GCC:

```
patch -Np1 -i ../expect-5.41.0-spawn-1.patch
```

Ahora, 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
```

Si quieres 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 aún no se comprenden por completo. Por tanto, estos fallos del banco de pruebas no son una sorpresa y no se consideran críticos.

E instálalo:

```
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.

Contenido de Expect

Programa instalado: expect

Librería instalada: libexpect5.41.0.a

Descripción corta

expect “habla” con otros programas interactivos según un guión.

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:       6.1 MB
```

La instalación de Dejagnu depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Instalación de DejaGnu

Prepara DejaGnu para su compilación:

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

Construye e instala el paquete:

```
make install
```

Contenido de DejaGnu

Programa instalado: runtest

Descripción corta

runtest es el guión envoltorio que encuentra el intérprete de comando de expect adecuado y entonces ejecuta DejaGnu.

GCC-3.3.3 - Fase 2

```
Tiempo estimado de construcción:  11.0 SBU
Espacio requerido en disco:       332.7 MB
```

Reinstalación de GCC

Ahora están instaladas las herramientas necesarias para comprobar GCC y Binutils: Tcl, Expect y DejaGnu. Por lo que ahora podemos reconstruir 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 los pseudo-terminales (PTYs) suministrados por tu distribución anfitrión. Hoy en día, los PTYs se implementan normalmente mediante el sistema de ficheros *devpts*. Puedes comprobar rápidamente 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 que seas capaz de resolver este asunto. Puedes consultar el Wiki de LFS en http://wiki.linuxfromscratch.org/ para obtener información sobre cómo conseguir que funcionen los PTYs.

Esta vez construiremos los compiladores de C y C++, por lo que tendrás que desempaquetar los paquetes core y g++ (y también testsuite si quieres ejecutar las pruebas). Desempaquetándolos en tu directorio de trabajo todos ellos se desempaquetarán en un único subdirectorio `gcc-3.3.3/`.

Primero, corrige un problema y haz un ajuste esencial:

```
patch -Np1 -i ../gcc-3.3.3-no_fixincludes-1.patch
patch -Np1 -i ../gcc-3.3.3-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[p.69], 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 que usamos 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
```

Antes de comenzar con la construcción de GCC, recuerda desactivar cualquier variable de entorno que modifique las opciones de optimización por defecto.

Ahora, prepara GCC para su compilación:

```
../gcc-3.3.3/configure --prefix=/tools \
    --with-local-prefix=/tools \
    --enable-clocale=gnu --enable-shared \
    --enable-threads=posix --enable-__cxa_atexit \
    --enable-languages=c,c++
```

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 *generic*, que es erróneo.
- **`--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++.

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 recomendamos 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-3.3.3/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-3.3.3 en sistemas i686-pc-linux-gnu en http://gcc.gnu.org/ml/gcc-testresults/2004-01/msg00826.html.

Advierte que los resultados contienen:

```
* 1 XPASS (unexpected pass) for g++
* 1 FAIL (unexpected failure) for g++
* 24 XPASS's for libstdc++
```

El éxito inesperado (unexpected pass) de g++ se debe al uso de la opción *--enable-__cxa_atexit*. Aparentemente, no todas las plataformas soportadas por GCC tienen soporte para "__cxa_atexit" en sus librerías de C, así que no siempre se espera pasar esta prueba con éxito.

Los 24 éxitos inesperados para libstdc++ son consecuencia de usar la opción *--enable-clocale=gnu*. Esta opción, que es la elección correcta en los sistemas basados en Glibc versiones 2.2.5 y posteriores, activa en la librería C de GNU un soporte de locale que es superior al modelo por defecto *generic* (que puede ser aplicable si estuvieras usando Newlibc, Sun-libc u otra libc). El conjunto de pruebas para libstdc++ parece esperar el modelo *generic*, de aquí que no siempre se espere pasar estas pruebas.

Si tienes unos pocos fallos inesperados, con frecuencia puedes ignorarlos. Los desarrolladores de GCC normalmente los tienen en cuenta pero todavía no se han puesto a corregirlos. Un caso particular es la prueba de filebuf_members en el banco de pruebas de la librería estándar C++. Se ha observado que esta prueba falla en ciertos casos, pero se supera en otros. En resumen, a menos que tus resultados difieran mucho de los mostrados en la anterior URL, es seguro continuar adelante.

Finalmente, instala el paquete:

```
make install
```

En este punto se recomienda encarecidamente que se repitan las comprobaciones que realizamos anteriormente en este capítulo. Regresa a “Ajustar las herramientas”[p.40] y repite la pequeña prueba de compilación. Si los resultados son malos muy posiblemente se deba a que olvidaste aplicar el parche Specs de GCC mencionado arriba.

Los detalles sobre este paquete se encuentran en “Contenido de GCC”[p.90].

Binutils-2.14 - Fase 2

```
Tiempo estimado de construcción:  1.5 SBU
Espacio requerido en disco:       35,6 MB
```

Reinstalación de Binutils

Vuelve a crear un directorio dedicado para la construcción:

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

Ahora, prepara Binutils para su compilación:

```
../binutils-2.14/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 a través de los directorios de librerías del anfitrión.

Antes de comenzar con la construcción de Binutils, recuerda desactivar cualquier variable de entorno que modifique las opciones de optimización por defecto.

Compila el paquete:

```
make
```

La compilación está completa. Como se explica 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
```

No deberían haber fallos inesperados, los fallos esperados son correctos. Desafortunadamente, no hay un modo fácil para ver el sumario del resultado de las pruebas como lo había en el anterior paquete GCC. Sin embargo, si aquí ocurre un fallo es fácil de detectar. La salida mostrada contendrá algo como:

```
make[1]: *** [check-binutils] Error 2
```

Instala el paquete:

```
make install
```

Ahora, 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. Los volveremos a necesitar durante el siguiente capítulo en el estado en que se encuentran ahora.

Los detalles sobre este paquete se encuentran en “Contenido de Binutils”[p.88].

Gawk-3.1.3

El paquete Gawk contiene programas para manipular ficheros de texto.

```
Tiempo estimado de construcción:  0.2 SBU
Espacio requerido en disco:       16.9 MB
```

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

Instalación de Gawk

Prepara Gawk para su compilación:

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

Compila el paquete:

```
make
```

(Si insistes en comprobar los resultados, ejecuta: **`make check`**.)

E instálalo:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Gawk”[p.101].

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, Sed.

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 si haces esto tendrás que afrontar las consecuencias tu mismo: parchear los múltiples paquetes que todavía utilizan la vieja sintaxis. Por tanto, nosotros recomendamos usar las instrucciones mostradas arriba.

Compila el paquete:

```
make
```

(Si insistes en 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 ciertas plataformas. Sin embargo, normalmente no hay problemas sobre Linux.

E instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Coreutils”[p.92].

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, Make.

Instalación de Bzip2

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

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

Los detalles sobre este paquete se encuentran en “Contenido de Bzip2”[p.129].

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, Sed.

Instalación de Gzip

Prepara Gzip para su compilación:

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

Compila el paquete:

```
make
```

E instálalo:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Gzip”[p.139].

Diffutils-2.8.1

El paquete Diffutils contiene programas que mustran 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, Sed.

Instalación de Diffutils

Prepara Diffutils para su compilación:

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

Compila el paquete:

```
make
```

E instálalo:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Diffutils”[p.131].

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, Sed.

Instalación de Findutils

Prepara Findutils para su compilación:

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

Compila el paquete:

```
make
```

(Si insistes en comprobar los resultados, ejecuta: **`make check`**.)

E instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Findutils”[p.100].

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, Sed.

Instalación de Make

Prepara Make para su compilación:

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

Compila el programa:

```
make
```

(Si insistes en comprobar los resultados, ejecuta: **`make check`**.)

E instala el programa y su documentación:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Make”[p.143].

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, Texinfo.

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 los programas:

```
make
```

(Si insistes en comprobar los resultados, ejecuta: **`make check`**.)

E instala el paquete y su documentación:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Grep”[p.137].

Sed-4.0.9

El paquete Sed contiene un editor de flujos.

```
Tiempo estimado de construcción:  0.2 SBU
Espacio requerido en disco:       5.9 MB
```

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

Instalación de Sed

Prepara Sed para su compilación:

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

Compila el programa:

```
make
```

(Si insistes en comprobar los resultados, ejecuta: **`make check`**.)

E instala el paquete y su documentación:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Sed”[p.111].

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:       67.6 MB
```

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

Instalación de Gettext

Prepara Gettext para su compilación:

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

Compila los programas:

```
make
```

(Si insistes en 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 (pero en el proyecto Patches hay disponible un parche experimental para desactivar Java).)

E instala el paquete:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Gettext”[p.113].

Ncurses-5.4

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

```
Tiempo estimado de construcción:  0.7 SBU
Espacio requerido en disco:       27.8 MB
```

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

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 los programas y librerías:

```
make
```

E instálalos junto con su documentación:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Ncurses”[p.102].

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, Sed.

Instalación de Patch

Prepara Patch para su compilación (establecer la opción del preprocesador *-D_GNU_SOURCE* sólo es necesario en la plataforma PowerPC. Puedes omitirlo para otras arquitecturas):

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

Compila el programa:

```
make
```

E instálalo con su documentación:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Patch”[p.145].

Tar-1.13.94

El paquete Tar contiene un programa de archivado.

```
Tiempo estimado de construcción:  0.2 SBU
Espacio requerido en disco:       10.3 MB
```

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

Instalación de Tar

Prepara Tar para su compilación:

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

Compila los programas:

```
make
```

(Si insistes en comprobar los resultados, ejecuta: **make check**.)

Luego, instala el paquete y su documentación:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Tar”[p.156].

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:       16.3 MB
```

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

Instalación de Texinfo

Prepara Texinfo para su compilación:

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

Compila los programas:

```
make
```

(Si insistes en comprobar los resultados, ejecuta: **`make check`**.)

Luego, instala el paquete y su documentación:

```
make install
```

Los detalles sobre este paquete se encuentran en “Contenido de Texinfo”[p.121].

Bash-2.05b

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, Ncurses, Sed.

Instalación de Bash

Bash contiene varios errores conocidos. Corrígelos con el siguiente parche:

```
patch -Np1 -i ../bash-2.05b-2.patch
```

Ahora, prepara Bash para su compilación:

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

Compila el programa:

```
make
```

(Si insistes en comprobar los resultados, ejecuta: **`make tests`**.)

Luego instálalo junto con su documentación:

```
make install
```

Y 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 “Contenido de Bash”[p.126].

Util-linux-2.12a

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, Zlib.

Instalación de Util-linux

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

```
cp configure configure.backup
sed "s@/usr/include@/tools/include@g" configure.backup > 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
```

Ahora 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 “Contenido de Util-linux”[p.157].

Perl-5.8.4

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, Sed.

Instalación de Perl

Primero, corrige algunas rutas a la librería C fijadas en el código:

```
patch -Np1 -i ../perl-5.8.4-libc-1.patch
```

Perl insiste en usar el programa **arch** para averiguar el tipo de máquina. Crea un pequeño guión que imita a este comando:

```
echo "uname -m" > /tools/bin/arch
chmod 755 /tools/bin/arch
```

Ahora prepara Perl para su compilación (asegúrate de poner correctamente 'IO Fcntl POSIX', son todas 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
```

Y copia estas herramientas y sus librerías:

```
cp perl pod/pod2man /tools/bin
mkdir -p /tools/lib/perl5/5.8.4
cp -R lib/* /tools/lib/perl5/5.8.4
```

Los detalles sobre este paquete se encuentran en “Contenido de Perl”[p.119].

Stripping

Los pasos de esta sección son opcionales. Si tu partición LFS es pequeña, te encantará saber que puedes eliminar algunas cosas innecesarias. Los binarios y librerías que has construido contienen unos 130 MB de símbolos de depuración innecesarios. Elimina esos símbolos de esta forma:

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

El último de los comandos anteriores se saltará unos veinte 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 puedes eliminar la documentación:

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

Ahora necesitarás tener como mínimo 850 MB de espacio libre en tu sistema LFS para ser capaz de construir e instalar Glibc en la siguiente fase. Si puedes construir e instalar Glibc, también podrás construir e instalar el resto.

Parte III. Construcción del sistema LFS

Capítulo 6. Instalación de los programas del sistema base

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, creamos algunas cosas auxiliares y, después, comenzamos a instalar todos los paquetes uno por uno.

La instalación de todos estos programas es algo bastante sencillo, por lo que puedes pensar que, probablemente, sea más corto dar aquí las instrucciones genéricas de instalación y sólo explicar en profundidad la instalación de los paquetes que necesiten un método alternativo. Aunque estemos de acuerdo en eso, hemos elegido dar las instrucciones completas para todos y cada uno de los paquetes, simplemente para minimizar la posibilidad de errores.

La clave para aprender qué es lo que hace que un sistema Linux funcione es saber para qué se usa cada paquete y por qué el usuario (o el sistema) lo necesita. Por esta razón se muestra un sumario del contenido de cada paquete instalado seguido de unas concisas descripciones de cada programa y librería que instala.

Si piensas usar optimizaciones para la compilación durante este capítulo, mírate la receta de optimización en http://www.escomposlinux.org/lfs-es/recetas/optimization.html (el original 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. En resumen, 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, cuanto tardará aproximadamente en construirse, cuanto espacio de disco necesita durante el proceso de construcción, la localización oficial para la descarga del paquete (en caso de que simplemente quieras actualizar alguno de ellos) y qué otros paquetes necesita para construirlo correctamente. Tras las instrucciones de instalación 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.linuxfromscratch.org/blfs/view/cvs/introduction/important.html. Y para un método diseñado especialmente para LFS mira http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt.

Montar los sistemas de ficheros proc y devpts

Para que ciertos programas funcionen correctamente, los sistemas de ficheros *proc* y *devpts* deben estar disponibles dentro del entorno chroot. El sistema de ficheros proc es el pseudosistema de ficheros de información de procesos a trvés del cual el núcleo suministra información sobre el estado del sistema. Y el sistema de ficheros devpts es hoy en día el método más común para implementar los pseudoterminales (PTYs). Desde la versión 2,4 del núcleo, un sistema de ficheros se puede montar tantas veces y en tantos lugares como quieras, así que no hay problema en que estos sistemas de ficheros estén todavía montados en tu sistema anfitrión, sobre todo porque son sistemas de ficheros virtuales.

Primero hazte *root*, pues solo *root* puede montar sistemas de ficheros en sitios inusuales. Luego comprueba de nuevo que está establecida correctamente la variable de entorno LFS ejecutando **`echo $LFS`** y asegurandote de que muestra la ruta al punto de montaje de tu partición LFS, que es `/mnt/lfs` si seguiste nuestro ejemplo.

Ahora crea los puntos de montaje para estos sistemas de ficheros:

```
mkdir -p $LFS/{proc,dev/pts}
```

Monta el sistema de ficheros *proc* con:

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

Y monta el sistema de ficheros *devpts* con:

```
mount devpts $LFS/dev/pts -t devpts
```

Puede que este último comando falle con un error del tipo:

```
filesystem devpts not supported by kernel

sistema de ficheros devpts no soportado por el núcleo
```

La causa más probable es que el núcleo de tu sistema anfitrión fue compilado sin soporte para el sistema de ficheros devpts (puedes comprobar qué sistemas de ficheros soporta tu núcleo con, por ejemplo, **cat /proc/filesystems**). Se necesitan unos cuantos PTYs para ser capaz de ejecutar más adelante los bancos de pruebas de Binutils y GCC. Si tu núcleo no soporta devpts, no te preocupes, pues hay otro modo para conseguir que los PTYs funcionen dentro del entorno chroot. Pronto cubriremos esto en la sección Make_devices[p.76].

Recuerda que, si por alguna razón detienes tu trabajo en el LFS y más tarde lo continuas, es importante comprobar que estos sistemas de ficheros estén todavía montados dentro del entorno chroot, de otra forma seguramente tengas problemas.

Entrar al entorno chroot

Es hora de entrar en el entorno chroot para iniciar la construcción e instalar tu sistema LFS final. Siendo toavía *root* ejecuta el siguiente comando para entrar al paqueño 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 -- ya que lo que el intérprete de comandos piensa que es `/` en realidad es el valor de `$LFS`, que se le pasó al comando chroot.

Advierte que `/tools/bin` queda el último en el PATH. Esto significa que una herramienta temporal no volverá a usarse a partir de que se instale su versión final. Bueno, al menos cuando el intérprete de comandos no recuerda la localización de los binarios ejecutados; por esta razón se desactiva la tabla interna de rutas pasandole 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.

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 por parte de un usuario que no es de confianza.

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 (y el libro asume que lo haces) 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.

Creación de los directorios

Ahora vamos a crear una estructura en nuestro sistema de ficheros LFS. Crearemos un árbol de directorios. Usando los siguientes comandos se creará un árbol más o menos estándar:

```
mkdir -p /{bin,boot,dev/{pts,shm},etc/opt,home,lib,mnt,proc}
mkdir -p /{root,sbin,srv,tmp,usr/local,var,opt}
mkdir -p /media/{floppy,cdrom}
mkdir /usr/{bin,include,lib,sbin,share,src}
ln -s share/{man,doc,info} /usr
mkdir /usr/share/{doc,info,locale,man}
mkdir /usr/share/{misc,terminfo,zoneinfo}
mkdir /usr/share/man/man{1,2,3,4,5,6,7,8}
mkdir /usr/local/{bin,etc,include,lib,sbin,share,src}
ln -s share/{man,doc,info} /usr/local
mkdir /usr/local/share/{doc,info,locale,man}
mkdir /usr/local/share/{misc,terminfo,zoneinfo}
mkdir /usr/local/share/man/man{1,2,3,4,5,6,7,8}
mkdir /var/{lock,log,mail,run,spool}
mkdir -p /var/{tmp,opt,cache,lib/misc,local}
mkdir /opt/{bin,doc,include,info}
mkdir -p /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. Haremos dos cambios: uno para el directorio personal de *root*, y otro en los directorios de los ficheros temporales.

```
chmod 0750 /root
chmod 1777 /tmp /var/tmp
```

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 en la máscara de permisos 1777.

Nota de conformidad con FHS

Basamos nuestro árbol de directorios en el estándar FHS (disponible en http://www.pathname.com/fhs/). Además del árbol arriba creado, este estándar estipula la existencia de `/usr/local/games` y `/usr/share/games`, pero no nos gustan para un sistema 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 los directorios que pensamos que son necesarios.

Creación de los enlaces simbólicos escenciales

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
```

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:
EOF
```

Los grupos creados no son parte de ningún estándar, son los grupos que el guión **make_devices** utiliza en la siguiente sección. El LSB (Linux Standard Base) Recomienda sólo 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[p.28], 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 +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. Puesto que queremos usar nuestros binarios recién compilados tan pronto como sean instalados, desactivamos 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, por lo que si quieres que se produzca el registro, deberás crear los ficheros tú mismo. El paquete Shadow necesita detectar estos ficheros en sus ubicaciones adecuadas, así que los crearemos ahora con sus permisos correctos:

```
touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}
chmod 644 /var/run/utmp /var/log/{btmp,lastlog,wtmp}
```

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

Creación de los dispositivos con Make_devices-1.2

El paquete Make_devices contiene un guión para crear nodos de dispositivos.

```
Tiempo estimado de construcción:  1 SBU
Espacio requerido en disco:       160 KB
```

La instalación de Make_devices depende de: Bash, Bzip2, Coreutils.

Crear los dispositivos

Ten en cuenta que al desempaquetar el fichero `make_devices-1.2.bz2` no se crea un directorio al que debas entrar con **cd**, pues el fichero sólo contiene un guión del intérprete de comandos.

Instala el guión **make_devices**:

```
bzcat make_devices-1.2.bz2 > /dev/make_devices
chmod 754 /dev/make_devices
```

Los nodos de dispositivos son ficheros especiales: cosas que pueden generar o recibir datos. Usualmente corresponden a piezas físicas del hardware. Los nodos de dispositivos pueden crearse ejecutando comandos del tipo: **mknod -m modo nombre tipo mayor menor**. En dicho comando, *modo* es el usual triplete lectura/escritura/ejecución en octal, y *nombre* es el nombre del fichero de dispositivo a crear. Puede parecer sorprendente, pero el nombre del dispositivo en realidad es arbitrario, excepto que muchos programas requieren que dispositivos como `/dev/null` tengan su nombre usual. Los tres parámetros restantes le indican al núcleo a qué dispositivo se refiere en realidad el nodo. *tipo* es una letra, ya sea b o c, que indica si al dispositivo se accede en bloques (como un disco duro) o caracter a caracter (como la consola). Y *mayor* y *menor* son números que juntos identifican al dispositivo ante el núcleo. Una lista de los números de dispositivos asignados actualmente en Linux se puede encontrar en el fichero `devices.txt` dentro del subdirectorio `Documentation` de las fuentes del núcleo.

Advierte que una misma combinación mayor/menor se asigna normalmente tanto a un dispositivo de bloque como a uno de caracter. Estos son, sin embargo, dispositivos completamente diferentes que no pueden intercambiarse. Un dispositivo se identifica por el triplete tipo/mayor/menor, por lo que cuando se crea un nodo de dispositivo es importante elegir el *tipo* correcto del dispositivo.

Debido a que buscar los tripletes tipo/mayor/menor y usar manualmente **mknod** es tedioso y propenso a errores, se ha creado el guión `make_devices`. Este contiene una serie completa de comandos **mknod**, uno por cada dispositivo, completados con la asignación del nombre, permisos y grupo recomendado. Ha sido configurado para que sólo un grupo mínimo de dispositivos más comúnmente usados esté activado y el resto de líneas están comentadas. Deberías abrir con un editor `make_devices` y personalizarlo a tus necesidades. Esto se tomará su tiempo, pero es muy sencillo. Cuando estés satisfecho, ejecuta el guión para crear los ficheros de dispositivos:

Aviso

Si no editas correctamente **make_devices** para ajustarlo a la configuración de tu sistema (por ejemplo, el número de particiones) puede provocar errores en el arranque.

```
cd /dev
./make_devices
```

Si pudiste montar el sistema de ficheros devpts antes, en “Montar los sistemas de ficheros proc y devpts”[p.70], puedes continuar con la siguiente sección. Si no pudiste montar devpts, deberás crear crear en su lugar unos cuantos nodos de dispositivos ptyXX y ttyXX estáticos. Para hacer esto, abre en tu editor `make_devices`, ve a la sección “Pseudo-TTY masters” y activa algunos dispositivos ptyXX, con media docena hay suficiente para permitir que se ejecuten los bancos de pruebas, pero si piensas ejecutar un núcleo sin soporte para devpts problablemente necesitarás muchos más (cada xterm, conexión ssh o telnet y similares utilizan uno de estos pseudoterminales). En la sección “Pseudo-TTY slaves”, que está inmediatamente a continuación, activa los dispositivos ttyXX correspondientes. Cuando estés listo, vuelve a ejecutar **./make_devices** dentro de `/dev` para que crear los nuevos dispositivos.

Contenido de Make_devices

Guión instalado: make_devices

Descripción corta

make_devices es un guión que crea un grupo básico de nodos de dispositivo estáticos, que usualmente residen en el directorio `/dev`.

Cabeceras de Linux-2.4.26

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

Instalación de las cabeceras del núcleo

No compilaremos todavía un nuevo núcleo, lo haremos cuando terminemos la instalación de todos los paquetes. Pero las librerías instaladas en la siguiente sección necesitan referenciarse a los ficheros de cabecera del núcleo para saber cómo interactuar con el núcleo. En vez de desempaquetar de nuevo las fuentes del nucleo, crear el fichero de versión y los enlaces simbólicos, etc..., simplemente simplemente copiaremos de golpe las cabeceras a partir del directorio de las herramientas temporales:

```
cp -a /tools/include/{asm,asm-generic,linux} /usr/include
```

Unos pocos ficheros de cabecera del núcleo referencian el fichero de cabecera `autoconf.h`. Puesto que todavía no hemos configurado el núcleo, necesitamos crear este fichero por nuestra cuenta para evitar un fallo en la compìlación de Sysklogd. Crea un fichero `autoconf.h` vacío con:

```
touch /usr/include/linux/autoconf.h
```

Por qué copiamos las cabeceras del núcleo y no las enlazamos simbólicamente.

En el pasado, era una práctica común enlazar simbólicamente los directorios `/usr/include/{linux,asm}` a `/usr/src/linux/include/{linux,asm}`. Esta fue una *mala* práctica, como señala este extracto de un mensaje de Linus Torvalds a la lista de correo del núcleo Linux:

```
Sugeriría que la gente que compile núcleos nuevos debe:

 - no tener un sólo enlace simbólico a la vista (excepto el que crea la
   misma construcción del núcleo, el enlace simbólico llamado
   “linux/include/asm”, que sólo se usa para la compilación interna
   del propio núcleo).

Y sí, esto es lo que yo hago. Mi /usr/src/linux todavía contiene los
ficheros de cabecera del antiguo 2.2.13, aunque no he ejecutado un
núcleo 2.2.13 desde hace _muucho_ tiempo. Pero esas fueron las cabeceras
con las que fue compilada Glibc, por lo que esas cabeceras son las que
coinciden con los ficheros objeto de la librería.

Y este es, de hecho, el entorno que se ha sugerido en, al menos, los
últimos cinco años. No sé por qué el asunto del enlace simbólico sigue
coleando, como un mal zombie. Casi cada distribución todavía tiene ese
enlace simbólico roto,  y la gente todavía recuerda que el código fuente
de linux debe ir en “/usr/src/linux” aunque no ha sido cierto desde
hace _muucho_ tiempo.
```

La parte relevante es donde Linus afirma que las cabeceras deberían ser con las que *fue compilada Glibc*. Estas son las cabeceras que deberías usar cuando más adelante compiles otros paquetes, pues son las que coinciden con los códigos de objetos de las librerías. Al copiar las cabeceras nos aseguramos de que permanecen disponibles si posteriormente actualizas el núcleo.

De paso, fíjate en que es perfectamente correcto tener las fuentes del núcleo en `/usr/src/linux`, mientras no tengas los enlaces simbólicos `/usr/include/{linux,asm}`.

Man-pages-1.66

El paquete Man-pages contiene alrededor de 1200 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, Make.

Instalación de Man-pages

Instala Man-pages ejecutando:

```
make install
```

Contenido de Man-pages

Ficheros instalados: diversas páginas de manual.

Descripción corta

Ejemplos de las *páginas de manual* incluidas son las que describen todas las funciones C y C++, los ficheros de dispositivo importantes y los ficheros de configuración importantes.

Glibc-2.3.3-lfs-5.1

El paquete Glibc contiene la librería C principal. Esta librería proporciona todas las rutinas básicas para la ubicación de memoria, busqueda 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, Texinfo.

Instalación de Glibc

El sistema de construcción de Glibc está muy bien autocontenido y se instalará perfectamente, incluso aunque nuestros ficheros de especificación 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.

Antes de comenzar a construir Glibc, recuerda desactivar las variables de entorno que puedan sobreescribir las opciones de optimización por defecto.

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

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

Ahora prepara Glibc para su compilación:

```
../glibc-2.3.3-lfs-5.1/configure --prefix=/usr \
    --disable-profile --enable-add-ons=linuxthreads \
    --libexecdir=/usr/lib --with-headers=/usr/include \
    --without-cvs
```

El significado de las nuevas opciones de configure:

- **`--libexecdir=/usr/lib`**: Esto cambia la localización del programa **pt_chown** de su ubicación por defecto `/usr/libexec` a `/usr/lib`. El uso de *libexec* no se considera acorde con el FHS pues el FHS ni siquiera lo menciona.
- **`--with-headers=/usr/include`**: Esto asegura que se utilicen las cabeceras del núcleo que hay en `/usr/include` para esta construcción. Si no se le pasa esta opción, entonces se utilizan las cabeceras que hay en `/tools/include`, lo que, por supuesto, no es ideal (aunque deberían ser idénticas). El uso de esta opción tiene la ventaja de que serás informado inmediatamente si te olvidaste de instalar los ficheros de cabecera del núcleo en `/usr/include`.

Compila el paquete:

```
make
```

Importante

El banco de pruebas para Glibc en este capítulo se considera *crítico*. No te lo saltes bajo ninguna circunstancia.

Comprueba los resultados:

```
make check
```

Las notas sobre el banco de pruebas que hay en “Glibc-2.3.3-lfs-5.1”[p.37] son aún más apropiadas aquí. Asegúrate de

consultarlas si tienes alguna duda.

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
```

E 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
```

Una alternativa al comando anterior 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
```

Por último, construye las páginas de manual de linuxthreads:

```
make -C ../glibc-2.3.3-lfs-5.1/linuxthreads/man
```

E instálalas:

```
make -C ../glibc-2.3.3-lfs-5.1/linuxthreads/man install
```

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. Esto se tratará en un capítulo posterior. 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 saber en qué zona horaria estás, ejecuta este 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/Canada/Eastern /etc/localtime
```

El significado de la opción:

- **`--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, por ejemplo, se arranca en modo de usuario único.

Por supuesto, reemplaza *Canada/Eastern* por el nombre de la zona horaria que te dió el guión **tzselect**.

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
```

Contenido de Glibc

Programas instalados: catchsegv, gencat, getconf, getent, glibcbug, 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], libc_nonshared.a, 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 especificas del sistema de ficheros.

getent obtiene entradas de una base de datos administrativa.

glibcbug crea un informe de fallos y lo envía a la dirección de correo electrónico de errores.

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 especificada.

lddlibc4 asiste a ld con los ficheros objeto.

locale es un programa Perl que le dice al compilador si debe activar (o desactivar) el uso de las locales POSIX para operaciones integradas.

localedef compila las especificaciones para locale.

mtrace ...

nscd es 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 es 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 en un servidor RPC.

sln se usa para hacer enlaces simbólicos. Está enlazado estáticamente, por lo que es útil para crear enlaces simbólicos a librerías dinámicas si, por alguna razón, el enlazador dinámico del sistema no funciona.

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 es el visualizador de la zona horaria.

zic es el compilador de la zona horaria.

ld.so es el programa de ayuda para las librerías compartidas ejecutables.

libBrokenLocale es usada por programas como Mozilla para resolver locales rotas.

libSegFault es un manejador de señales de violación de segmento. Intenta capturar estas señales.

libanl es una librería de búsqueda de nombres asíncrona.

libbsd-compat proporciona la portabilidad necesaria para ejecutar ciertos programas BSD en Linux.

libc es la librería principal de C, una colección de funciones usadas frecuentemente.

libcrypt es la librería criptográfica.

libdl es la librería de interfaz del enlazado dinámico.

libg es una librería en tiempo de ejecución de g++.

libieee es la librería de punto flotante IEEE.

libm es la librería matemática.

libmcheck contiene código ejecutado en el arranque.

libmemusage es usada por memusage para ayudar a recoger información sobre el uso de memoria de un programa.

libnsl es la librería de servicios de red.

libnss* son las librerías Name Service Switch (Interruptor del Servicio de Nombres). Contienen funciones para resolver el nombre de sistemas, de usuarios, de grupos, alias, servicios, protocolos y similares.

libpcprofile Código usado por el núcleo para rastrear el tiempo de CPU gastado en funciones, líneas de código fuente e instrucciones.

libpthread es 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.

Reajustar las herramientas

Ahora que hemos instalado las nuevas y finales librerías de C, es hora de ajustar de nuevo nuestro 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 anterior, incluso parace lo contario: antes lo guiamos de los directorios `/{,usr}/lib` del anfitrión a los nuevos `/tools/lib`, ahora lo guiamos de `/tools/lib` a los directorios `/{,usr}/lib` del LFS.

Primero ajustaremos 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[p.28], o los borraste accidentalmente o no tienes acceso a ellos, no te preocupes, no todo está perdido. 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`. El *INSTALL=/tools/bin/install* extra es necesario 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.

Ahora debes borrar los directorios de fuentes y de construcción de Binutils. (Esto es importante, pues deberías empezar la siguiente sección con un nuevo desempaquetado del paquete.)

Lo siguiente es corregir el fichero de especificaciones de GCC para que apunte al nuevo enlazador dinámico. Como antes, usaremos sed para hacerlo:

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

De nuevo te recomendamos que copies y pegues este comando. Como antes, 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 simbólico no sea `ld-linux.so.2`, *debes* sustituir `ld-linux.so.2` en el comando anterior por el nombre del enlazador dinámico para tu plataforma. Si es necesario, consulta “Notas técnicas sobre las herramientas”[p.29].

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]
```

Advierte especialmente que `/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. Necesitarás investigar y revisar tus pasos para encontrar el problema y corregirlo. No hay motivo para continuar hasta que esté hecho. Muy posiblemente algo fue mal durante el anterior arreglo del fichero de especificaciones.

Una vez estés seguro de que todo está bien, borra los ficheros de prueba:

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

Binutils-2.14

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

```
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, Texinfo.

Instalación de Binutils

Ahora es el momento apropiado para verificar que tus pseudo-terminales (PTYs) funcionan adecuadamente dentro del entorno chroot. Comprobaremos de nuevo rápidamente que todo está correcto realizando una simple prueba:

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

Si recibes el mensaje:

```
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.
```

Tu entorno chroot no está configurado para la correcta operación de PTY. En este caso no hay razón para ejecutar los bancos de pruebas de Binutils y GCC hasta que seas capaz de resolver este asunto. Regresa a “Montar los sistemas de ficheros proc y devpts”[p.70] y a la sección Make_devices[p.76] y lleva a cabo los pasos recomendados para corregir el problema.

Se sabe que este programa se comporta mal si cambias sus parámetros de optimización (incluyendo las opciones -march y -mcpu). Por tanto, si tienes definida cualquier variable de entorno que pueda sobreescribir las optimizaciones por defecto, como CFLAGS y CXXFLAGS, te recomendamos que las desactives o modifiques antes de construir Binutils.

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

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

A continuación, prepara Binutils para su compilación:

```
../binutils-2.14/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, por ejemplo, `/usr/i686-pc-linux-gnu`. Como sólo construimos programas para nuestro propio sistema, no necesitamos en `/usr` este 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

El banco de pruebas para Binutils de este capítulo se considera *crítico*. No te lo saltes bajo ninguna circunstancia.

Comprueba los resultados:

```
make check
```

Las notas sobre el banco de pruebas que hay en “Binutils-2.14 - Fase 2”[p.48] son aún más apropiadas aquí. Asegúrate de consultarlas en caso de que tengas alguna duda.

Instala el paquete:

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

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

```
cp ../binutils-2.14/include/libiberty.h /usr/include
```

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 traslada 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.

arcrea, modifica y extrae desde archivos. Un archivo es un fichero que almacena una colección de otros ficheros en una estructura que hace posible obtener el original de cada fichero individual (llamados miembros del archivo).

as es un ensamblador. Ensambla la salida de gcc dentro de ficheros objeto.

c++flit es usado por el enlazador para decodificar (demangling) símbolos de C++ y Java, guardando las funciones sobrecargadas para su clasificación.

gprof muestra los datos del perfil del gráfico de llamada.

ld es un enlazador. Combina un número de ficheros de objetos 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 se utiliza para traducir 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 cada uno de 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 por defecto sólo muestra 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 es la librería del Descriptor de Fichero Binario.

libopcodes es 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.

GCC-3.3.3

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 dendende de: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Instalación de GCC

Se sabe que este programa se comporta mal si cambias sus parámetros de optimización (incluyendo las opciones -march y -mcpu). Por esta razón, si tienes definida cualquier variable de entorno que pueda sobreescribir las optimizaciones por defecto, como CFLAGS o CXXFLAGS, recomendamos quitarlas o modificarlas cuando construyas GCC.

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

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

```
patch -Np1 -i ../gcc-3.3.3-no_fixincludes-1.patch
```

Ahora 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 directorio de las fuentes, en un directorio de construcción dedicado:

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

Ahora prepara GCC para su compilación:

```
../gcc-3.3.3/configure --prefix=/usr \
    --enable-shared --enable-threads=posix \
    --enable-__cxa_atexit --enable-clocale=gnu \
    --enable-languages=c,c++
```

Compila el paquete:

```
make
```

Importante

El banco de pruebas para GCC de este capítulo se considera *crítico*. No te lo saltes bajo ninguna circunstancia.

Comprueba los resultados, pero no pares en los errores (recordarás que hay varios conocidos):

```
make -k check
```

Las notas para el banco de pruebas que hay en “GCC-3.3.3 - Fase 2”[p.45] son aún más apropiadas aquí. Asegúrate de consultarlas si tienes alguna duda.

Ahora 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 “Reajustar las herramientas”[p.85] 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[p.28].

Contenido de GCC

Programas instalados: c++, cc (enlace a gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug y gcov

Librerías instaladas: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] y libsupc++.a

Descripciones cortas

cpp es el preprocesador de C. Lo usa el compilador para tener las sentencias #include, #define y similares expandidas en los ficheros fuente.

g++ es el compilador de C++.

gcc es el compilador de C. Se usa para traducir el código fuente de un programa a código ensamblador.

gccbug es un guión del interprete de comandos que ayuda a crear buenas notificaciones de errores.

gcov es 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++ es la librería estándar de C++. Contiene muchas funciones de uso frecuente.

libsupc++ proporciona rutinas de soporte para el lenguaje de programación c++.

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, Sed.

Instalación de Coreutils

Normalmente, el comando **uname** no funciona como debiera, ya que la opción *-p* siempre devuelve “unknown” (desconocido). Este parche corrige su funcionamiento en arquitecturas Intel:

```
patch -Np1 -i ../coreutils-5.2.1-uname-1.patch
```

No queremos que Coreutils instale su versión de **hostname**, pues es inferior a la versión suministrada por Net-tools. Evita su instalación aplicando este parche:

```
patch -Np1 -i ../coreutils-5.2.1-hostname-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 pronto en la construcción de LFS. Por tanto tenemos que preparar varias cosas antes de poder ejecutar las pruebas. Si decides no ejecutar el banco de pruebas, salta hasta “Instala el paquete”.

Para poder ejecutar el banco de pruebas al completo, se debe instalar el programa **su**. No nos preocupamos de instalar este pequeño programa en Capítulo 5[p.28] porque necesita privilegios de root, así que hazlo ahora:

```
make install-root
```

Crea un fichero de “tabla de sistemas de ficheros montados” con:

```
touch /etc/mtab
```

Y 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*:

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

A continuación ejecuta el resto como usuario *dummy*:

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

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

```
sed -i.bak '/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,install,ln,ls} /bin
mv /usr/bin/{mkdir,mknod,mv,pwd,rm,rmdir,sync} /bin
mv /usr/bin/{sleep,stty,su,test,touch,true,uname} /bin
mv /usr/bin/chroot /usr/sbin
```

Usaremos el programa **kill** del paquete Procps (que se instala más tarde en este capítulo como `/bin/kill`). Elimina el instalado por Coreutils:

```
rm /usr/bin/kill
```

Finalmente, crea dos enlaces simbólicos para cumplir con el FHS:

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

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, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami y yes

Descripciones cortas

basename elimina los directorios y los sufijos de los nombres de ficheros.

cat concatena ficheros en la salida estándar.

chgrp cambia el grupo 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 `/`. Dicho comando puede ser un interprete de comandos interactivo. En muchos sistemas solo *root* puede hacer esto.

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 indicados.

dir es 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 las líneas de 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, 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 (login name) del usuario actual.

ls lista el contenido de cada directorio indicado. Por defecto ordena los ficheros y subdirectorios alfabéticamente.

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 dispositivos de nodos con los nombres indicados. Un dispositivo de nodo 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 una utilidad parecida a finger. Muestra algo de información sobre un determinado usuario.

pr pagina o 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 fija una parada por el espacio de tiempo indicado.

sort ordena las líneas de los ficheros indicados.

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

stty establece o muestra los ajuste de línea del terminal.

su ejecuta un intérprete de comandos con un identificador de usuario y de grupo diferentes.

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 los ficheros indicados en orden inverso..

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/o 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.

uptime muestra cuanto tiempo hace que el sistema está en marcha, cuantos usuarios hay conectados y los índices de carga del sistema.

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.

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, Sed.

Instalación de Zlib

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 **configure** 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
```

Ahora, construye también la librería no compartida (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
```

Y 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 tiene más sentido 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 disponible 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
```

Ahora necesitamos corregir el enlace simbólico `/usr/lib/libz.so`, pues acabamos de mover el fichero al que apunta:

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

Contenido de Zlib

Librería instalada: libz[a,so]

Descripción corta

libz* contiene funciones de compresión y descompresión usadas por algunos programas.

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
```

Las dependencias de instalación de Mktemp no se han comprobado aún.

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** wrapper:

```
patch -Np1 -i ../mktemp-1.5-add-tempfile.patch
```

Ahora prepara Mktemp para su compilación:

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

El significado de las opciones 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
```

Ahora instálalo:

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

Contenido de Mktemp

Programas instaladados: mktemp, 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 compatibilidad hacia atrás.

Iana-Etc-1.00

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

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

Las dependencias de instalación de Iana-Etc no han sido comprobadas aún.

Instalación de Iana-Etc

Procesa los datos:

```
make
```

Ahora instálalos:

```
make install
```

Contenido de Iana-Etc

Ficheros instalados: protocols, services

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, Sed.

Instalación de Findutils

Prepara Findutils para su compilación:

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

La anterior directiva localstatedir cambia la localización de la base de datos de locate a /var/lib/misc, que cumple el FHS.

Compila el paquete:

```
make
```

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

Ahora instala el paquete:

```
make install
```

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 cumpla 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.

Gawk-3.1.3

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, Sed.

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`**.

Ahora instala el paquete:

```
make install
```

Contenido de Gawk

Programas instalados: awk (enlace a gawk), gawk, gawk-3.1.3, grcat, igawk, pgawk, pgawk-3.1.3 y pwcat

Descripciones cortas

gawk es un programa para manipular ficheros de texto. Es la implementación GNU de awk.

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.

pwcat vuelca la base de datos de contraseñas `/etc/passwd`.

Ncurses-5.4

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

```
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, Sed.

Instalación de Ncurses

Prepara Ncurses para su compilación:

```
./configure --prefix=/usr --with-shared --without-debug
```

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

Otorga permisos de ejecución a las librerías Ncurses:

```
chmod 755 /usr/lib/*.5.4
```

Ahora corrige una librería que no debería ser ejecutable:

```
chmod 644 /usr/lib/libncurses++.a
```

Mueve las librerías al directorio `/lib`, donde se espera encontrarlas:

```
mv /usr/lib/libncurses.so.5* /lib
```

Debido 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
```

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], libform_g.a, libmenu.[a,so], libmenu_g.a, libncurses++.a, libncurses.[a,so], libncurses_g.a, libpanel.[a,so] y libpanel_g.a

Descripciones cortas

captoinfo convierte una descripción de termcap en una descripción de terminfo.

clear limpia la pantalla si es posible.

infocmp compara o imprime en pantalla una descripción de terminfo.

infotocap convierte una descripción de terminfo en una descripción de termcap.

reset reinicializa un terminal a sus valores por defecto.

tack es el comprobador de acciones de terminfo. Se usa principalmente para verificar la que una entrada de la base de

datos de terminfo sea correctas.

tic es el compilador de entradas de descripciones de 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.

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.

Vim-6.2

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, Sed.

Alternativas a Vim

Si prefieres otro editor en vez de Vim, como Emacs, Joe, o Nano, mira en http://www.escomposlinux.org/lfs-es/blfs-es-CVS/postlfs/editors.html las instrucciones de instalación sugeridas (el original se encuentra en http://www.linuxfromscratch.org/blfs/view/stable/postlfs/editors.html).

Instalación de Vim

Primero 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
```

Ahora, prepara Vim para su compilación:

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

Compila el paquete:

```
make
```

Para comprobar los resultados, ejecuta: **`make test`**. Sin embargo, este banco de pruebas mostrará un montón de caracteres basura por pantalla que pueden hacer estragos en los ajustes del terminal actual. Por tanto la ejecución de este banco de pruebas es estríctamente opcional.

Ahora 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 simbolico:

```
ln -s vim /usr/bin/vi
```

Si vas a instalar el sistema X Window en tu sistema LFS, puede que quieras recompilar Vim después de instalar X. Vim tiene una bonita versión con interfaz gráfica que necesita X y algunas otras librerías instaladas. Para más información lee la documentación de Vim.

Configuración de Vim

Por defecto, **vim** se ejecuta en modo compatible con vi. Hay gente a la que le puede gustar esto, pero nosotros preferimos ejecutar **vim** en su propio modo (de otra forma no lo habríamos incluido en este libro, sino al **vi** original). Hemos incluido a continuación la opción “nocompatible” para resaltar el hecho de que se va a usar este comportamiento. Esto también les recuerda a aquellos que quieran cambiar al modo “compatible” que debería aparecer al principio, pues modifica otros ajuste y sobreescribe mucho de lo que le sigue a continuación. Crea un fichero de configuración por defecto de vim ejecutando lo siguiente:

```
cat > /etc/vimrc << "EOF"
" Begin /etc/vimrc

set nocompatible
set backspace=2
syntax on
```

```
" End /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. Y la opción *syntax on* activa la coloración semántica de **vim**.

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 es 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 es 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 es una versión restringida de view: no pueden ejecutarse comandos del intérprete de comandos y view no puede ser suspendido.

rvim es 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 es 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 es un guión que corrige un fichero y genera las sentencias de sintaxis necesarias para resaltar las palabras en vim.

vimtutor te 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.

M4-1.4

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, Sed.

Instalación de M4

Prepara M4 para su compilación:

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

Compila el paquete:

```
make
```

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

E instala el paquete:

```
make install
```

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.

Bison-1.875

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, Sed.

Instalación de Bison

Primero corregiremos un problema menos de compilación que Bison tiene con algunos paquetes. El parche está extraido de su CVS:

```
patch -Np1 -i ../bison-1.875-attribute.patch
```

Ahora prepara Bison para su compilación:

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

Compila el paquete:

```
make
```

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

Ahora instala el paquete:

```
make install
```

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 es 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 es la librería Yacc que contiene la implementación de yyerror compatible con Yacc y funciones principales. Esta librería normalmente no es muy útil, pero POSIX la solicita.

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, Sed.

Instalación de Less

Prepara Less para su compilación:

```
./configure --prefix=/usr --bindir=/bin --sysconfdir=/etc
```

El 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
```

Ahora instálalo:

```
make install
```

Contenido de Less

Programas instalados: less, lessecho y lesskey

Descripciones

less es un visor de ficheros o paginador. Muestra el contenido de un fichero con la posibilidad de recorrerlo, hacer búsquedas o saltar a marcas.

lessecho es necesario para expandir meta-caracteres, como * y ?, en los nombres de ficheros en sistemas Unix.

lesskey se usa para especificar los códigos de teclas usados por less.

Groff-1.19

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, Sed.

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 el siguiente comando es adecuado. Si vives en otro lugar, puede que prefieras cambiar *PAGE=letter* por *PAGE=A4*.

Prepara Groff para su compilación:

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

Compila el paquete:

```
make
```

Ahora instálalo:

```
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
```

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 formulas embebidas en los ficheros de entrada de troff a comandos que pueda entender troff.

eqn2graph convierte una ecuación EQN en una imagen.

grn es un preprocesador groff para ficheros gremlin.

grodvi es un controlador para groff que genera formatos dvi de TeX.

groff es 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 cual 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 es un controlador de groff para las impresoras Canon CAPSL (series LBP-4 y LBP-8 de impresoras láser)

grolj4 es un controlador para groff que produce salidas en el formato PCL5 adecuado para impresoras HP Laserjet 4.

grops transforma la salida de GNU troff en Postscript.

grotty transforma la salida de GNU troff en un formato adecuado para dispositivos tipo máquina de escribir.

gtbl es 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 bibliográficos en 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 es un preprocesador simple para groff.

neqn formatea ecuaciones para salida ASCII (Código Estándar Americano para Intercambio de Información).

nroff es 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 de troff o TeX a comandos que puedan ser entendidos por TeX o troff.

pic2graph convierte un diagrama PIC en una imagen.

pre-grohtml transforma la salida de GNU troff a HTML.

post-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 es la implementación GNU de soelim.

Sed-4.0.9

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, Texinfo.

Instalación de Sed

Prepara Sed para su compilación:

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

Compila el paquete:

```
make
```

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

Ahora instala el paquete:

```
make install
```

Contenido de Sed

Programa instalado: sed

Descripción corta

sed se usa para filtrar y transformar ficheros de texto en una sola pasada.

Flex-2.5.4a

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, Sed.

Instalación de Flex

Prepara Flex para su compilación:

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

Compila el paquete:

```
make
```

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

Ahorqa 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
```

Contenido de Flex

Programas instalados: flex, flex++ (enlace a flex) y lex

Librería instalada: libfl.a

Descripciones cortas

flex es una herramienta para generar programas capaces de reconocer patrones de texto. El reconocimiento de patrones es muy útil en muchas aplicaciones. A partir de un conjunto de reglas de búsqueda flex genera un programa que busca esos patrones. La razón para usar flex es porque es mucho más fácil establecer las reglas de búsqueda que escribir un programa real que busque el texto.

flex++ invoca una versión de flex usada exclusivamente por analizadores C++.

lbfl.a es la librería flex.

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, Sed.

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, sobre unos 7 SBUs.

Ahora instala el paquete:

```
make install
```

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, project-id, team-address, trigger, urlget, user-email 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 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 de textual cuya forma gramatical depende de un número.

xgettext extrae las líneas de mensajes traducibles de los ficheros de los ficheros fuente indicados, para hacer la primera plantilla de traducción.

libasprintf defina 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 es una librería privada que contiene rutinas comunes utilizadas por diversos programas de gettext. No está indicada para uso general.

libgettextpose usa 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 es una librería privada que contiene rutinas comunes utilizadas por diversos programas de gettext. No está indicada para uso general.

Net-tools-1.60

El paquete Net-tools contiene programas para el trabajo en red básico.

```
Tiempo estimado de construcción:  0.1 SBU
Espacio requerido en disco:       9.4 MB
```

La instalación de Net-tools depende de: Bash, Binutils, Coreutils, GCC, Glibc, Make.

Instalación de Net-tools

Si no sabes qué contestar a todas las preguntas que se hacen durante la etapa **make config**, entonces basta con aceptar los valores por defecto, ya que será lo correcto en la mayoría de los casos. Lo que se pregunta aquí es una serie de cuestiones relativas al tipo de protocolos de red que tienes activados en tu núcleo. Las respuestas por defecto activarán las herramientas de este paquete para trabajar con los protocolos más comunes, como TCP, PPP y algunos otros. En realidad, todavía necesitarás activar esos protocolos en el núcleo. Lo que estás haciendo aquí es, simplemente, ordenar a los programas que sean capaces de usar esos protocolos, pero corre por cuenta del núcleo dejarlos disponibles para el sistema.

Primero corrige un pequeño problema de sintaxis en las fuentes del programa **mii-tool**:

```
patch -Np1 -i ../net-tools-1.60-miitool-gcc33-1.patch
```

Ahora prepara Net-tools para su compilación (si piensas aceptar todas las opciones por defecto, puedes saltarte las preguntas ejecutando en su lugar **yes "" | make config**):

```
make config
```

Compila el paquete:

```
make
```

Ahora instálalo:

```
make update
```

Contenido de Net-tools

Programas instalados: arp, dnsdomainname (enlace a hostname), domainname (enlace a hostname), hostname, ifconfig, nameif, netstat, nisdomainname (enlace a hostname), plipconfig, rarp, route, slattach y ypdomainname (enlace a hostname)

Descripciones cortas

arp se usa para manipular la caché ARP del núcleo, usualmente para añadir o borrar una entrada o volcar la caché completa.

dnsdomainname muestra el nombre del dominio DNS (Servidor de Nombres de Dominio) del sistema.

domainname muestra o establece el nombre del dominio NIS/YP del sistema.

hostname muestra o establece el nombre del sistema actual.

ifconfig es la utilidad principal usada para configurar las interfaces de red.

nameif nombra interfaces de red basándose en las direcciones MAC.

netstat se usa para mostrar las conexiones de red, tablas de encaminamiento y estadísticas de las interfaces.

nisdomainname hace lo mismo que domainname.

plipconfig se usa para afinar los parámetros del dispositivo PLIP, para mejorar su rendimiento.

rarp se usa para manipular la tabla RARP del núcleo.

route se usa para manipular la tabla de encaminamiento IP.

slattach conecta una interfaz de red a una línea serie. Esto permite usar líneas de terminales normales para crear enlaces punto a punto con otras computadoras.

ypdomainname hace lo mismo que domainname.

Inetutils-1.4.2

El paquete Inetutils contiene programas para el trabajo en red básico.

```
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, Sed.

Instalación de Inetutils

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 seguiente parche corregirá esta situación:

```
patch -Np1 -i ../inetutils-1.4.2-no_server_man_pages-1.patch
```

Ahora 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 solo se consideran seguros en redes de confianza. Puedes encontrar mas información en http://www.escomposlinux.org/lfs-es/blfs-es-CVS/basicnet/inetutils.html (el original se encuentra en http://www.linuxfromscratch.org/blfs/view/stable/basicnet/inetutils.html). Ten en cuenta que para muchos de estos servidores hay disponibles mejores sustitutos.

Compila el paquete:

```
make
```

Instálalo:

```
make install
```

Mueve el programa **ping** al lugar indicado por el FHS:

```
mv /usr/bin/ping /bin
```

Contenido de Inetutils

Programas instalados: ftp, ping, rcp, rlogin, rsh, talk, telnet y tftp

Descripciones cortas

ftp es el programa para transferencia de ficheros de ARPANET.

ping envía paquetes de petición de eco e informa cuanto tardan las respuestas.

rcp copia ficheros de forma remota.

rlogin permite la entrada remota al sistema.

rsh ejecuta un intérprete de comandos remoto.

talk permite hablar con otro usuario.

telnet es una interfaz de usuario para el protocolo TELNET.

tftp es un programa para la transferencia trivial de ficheros.

Perl-5.8.4

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, Sed.

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 piensas que puedes vivir con los (razonables) valores que autodetecta por defecto, entonces prepara Perl para su compilación con:

```
./configure.gnu --prefix=/usr -Dpager="/bin/less -isR"
```

El 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
```

Si decides ejecutar el banco de pruebas, primero debes crear 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
```

Por último, instala el paquete:

```
make install
```

Contenido de Perl

Programas instalados: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.4 (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: (demasiadas para nombrarlas)

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.

en2cxsconstruye 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.

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 es el Procedimiento de Verificación de la Instalación de Perl. Puede usarse para verificar que perl y sus librerías se han instalado correctamente.

piconv es 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 es la 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 es un programa que fuerza diagnósticos de avisos exhaustivos en perl.

xsubpp convierte el código XS de Perl en código C.

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, Sed.

Instalación de Texinfo

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 de tu árbol de TeX si, por ejemplo, piensas instalar más tarde un paquete de 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
```

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 e 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 varios otros formatos: 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.

Autoconf-2.59

El paquete Autoconf contiene programas para generar guiones del interprete de comandos que puedan configurar automáticamente el codigo 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, Sed.

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
```

Contenido de Autoconf

Programas instalados: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate e ifnames

Descripciones cortas

autoconf es una herramienta para generar guiones del intérprete de comandos que automáticamente configuran paquetes de código fuente, adaptándolos a muchas clases de sistemas tipo UNIX. Los guiones de configuración 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 hecha una mano cuando hay que enfrentarse a muchos guiones de configuración generados por autoconf. El programa ejecuta autoconf y autoheader (cuando es necesario) repetidamente para recrear los guiones de configuración de autoconf y las plantillas de configuración de las cabeceras en un árbol de directorios dado.

autoscan puede ayudar en la creación de ficheros `configure.in` para los paquetes de software. Este programa 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 ese 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 puede ser de ayuda cuando se escribe un `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.

Automake-1.8.4

Automake genera ficheros Makefile.in, pensados para usar 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, Sed.

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
```

Contenido de Automake

Programas instalados: acinstall, aclocal, aclocal-1.8, automake, automake-1.8, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, symlink-tree, ylwrap

Descripciones cortas

acinstall es un guión que instala ficheros M4 de estilo aclocal.

aclocal genera ficheros `aclocal.m4` basados en el contenido de ficheros `configure.in`.

automake es una 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 mas alto nivel. Mediante la exploración de los `configure.in` automáticamente encuentra cada `Makefile.am` apropiado y genera el correspondiente `Makefile.in`.

compile es una envoltura (wrapper) para compiladores.

config.guess es un guión que intenta averiguar el triplete canónico para la construcción, anfitrión o arquitectura objeto dada.

config.sub es un guión con subrutinas para la validación de configuraciones.

depcomp es un 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 es un guión que instala un programa, guión o fichero de datos.

mdate-sh es un guión que imprime la fecha de modificación de un fichero o directorio.

missing es un guión que actúa como sustituto común de programas GNU no encontrados durante una instalación.

mkinstalldirs es un guión que genera una árbol de directorios.

py-compile compila un programa Python.

symlink-tree es un guión para crear un árbol de enlaces simbólicos de un árbol de directorios.

ylwrap es un envoltorio para lex y yacc.

Bash-2.05b

El paquete Bash contiene la "Bourne-Again SHell".

```
Approximate build time:  1.2 SBU
Required disk space:     27 MB
```

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

Instalación de Bash

Bash contiene un número de errores que pueden hacer que en ocasiones no se comporte como es de esperar. Corrige este comportamiento con el parche siguiente:

```
patch -Np1 -i ../bash-2.05b-2.patch
```

Ahora prepara Bash para su compilación:

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

Compila el paquete:

```
make
```

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

Instala el paquete:

```
make install
```

Ahora lanza el programa **bash** recién compilado (sustituyendo al que estabasejecutando hasta ahora):

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

Advierte que los parámetros utilizados hacen de él una intáncia de acceso interactivo (por lo que lee /etc/profile, si existe, y el primero que encuentre de ~/.bash_profile, ~/.bash_login o ~/.profile) y continúa desactivando las tablas internas para que los nuevos programas sean encontrados en cuanto estén disponibles.

Contenido de Bash

Programas instalados: bash, sh (enlace a bash) y bashbug

Descripciones cortas

bash es un intérprete de comandos ampliamente usado. Realiza muchos tipos de expansiones y sustituciones en una línea de comando dada antes de ejecutarla, lo que hace de este intérprete una herramienta poderosa.

bashbug es un guión que ayuda al usuario en la composición y envío de informes de errores relacionados con bash, en un formato estándar.

sh es un 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.

File-4.09

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, Zlib.

Instalación de File

Prepara File para su compilación:

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

Compila el paquete:

```
make
```

Ahora instálalo:

```
make install
```

Contenido de File

Programa instalado: file

Librería instalada: 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. La primera prueba que tenga éxito determina el resultado.

libmagic contiene rutinas para reconocimiento de números mágicos, usados por el programa file.

Libtool-1.5.6

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, Sed.

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
```

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.

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, Make.

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
```

Ahora 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}
mv /usr/bin/{bzip2recover,bzless,bzmore} /bin
ln -s bzip2 /bin/bunzip2
ln -s bzip2 /bin/bzcat
```

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 y sus derivados ejecutan grep 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 LZ77/LZ78, como **gzip**.

bzip2recover intenta recuperar datos de ficheros bzip2 dañados.

bzless ejecuta less sobre ficheros comprimidos con bzip2.

bzmore ejecuta more sobre ficheros comprimidos con bzip2.

libbz2* es la librería que implementa la compresión sin pérdidas por ordenación de bloques, usando el algoritmo de Burrows-Wheeler.

Diffutils-2.8.1

El paquete Diffutils contiene programas que mustran 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, Sed.

Instalación de Diffutils

Prepara Diffutils para su compilación:

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

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

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.

Ed-0.2

El paquete Ed contiene un editor de líneas austero.

```
Tiempo estimado de construcción:  0.1 SBU
Espacio requerido en disco:       3.1 MB
```

La instalación de Ed depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Instalación de Ed

Nota

Ed no es algo que utilice mucha gente. Se instala aquí porque puede que lo use el programa patch si te encuentras con algún parche basado en ed. Esto no suele ocurrir porque ahora se prefieren los parches basados en diff.

Generalmente, Ed usa la función *mktemp* para crear ficheros temporales en `/tmp`, pero esta función tiene una vulnerabilidad de seguridad (ver "Temporary Files" en http://en.tldp.org/HOWTO/Secure-Programs-HOWTO/avoid-race.html). Aplica el siguiente parche para hacer que Ed use *mkstemp* en su lugar, una forma segura para crear ficheros temporales.

```
patch -Np1 -i ../ed-0.2-mkstemp.patch
```

Ahora prepara Ed para su compilación:

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

El significado de la opción de configure:

- **`--exec-prefix=""`**: Esto fuerza que los programas se instalen en el directorio `/bin`. Es útil tener a mano estos programas cuando la partición `/usr` no está disponible.

Compila el paquete:

```
make
```

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

Instala el paquete:

```
make install
```

Contenido de Ed

Programas instalados: ed y red (enlace a ed)

Descripciones cortas

ed es un editor de líneas de texto. Se usa para crear, mostrar, modificar o cualquier otra manipulación de ficheros de texto.

red es un ed restringido: sólo puede editar ficheros del directorio actual y no puede ejecutar comandos del intérprete de comandos.

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, Sed.

Instalación de Kbd

Kbd no instala algunas de sus utilidades (**setlogcons**, **setvesablank** y **getunimap**) por defecto. Primero activa la compilación de estas utilidades:

```
patch -Np1 -i ../kbd-1.12-more-programs-1.patch
```

Ahora prepara Kbd para compilarlo:

```
./configure
```

Compila el paquete:

```
make
```

Ahora instálalo:

```
make install
```

Configuración del teclado

Nada es más molesto que usar Linux teniendo cargado un mapa de teclado incorrecto. Si tienes un teclado estándar de US (EEUU), te puedes saltar esta sección. El mapa de teclado US es el mapa por defecto si no lo cambias.

Para asignar un mapa de teclado por defecto, crea el enlace simbólico `/usr/share/kbd/keymaps/defkeymap.map.gz` ejecutando los siguientes comandos:

```
ln -s ruta/mapa/teclado /usr/share/kbd/keymaps/defkeymap.map.gz
```

Reemplaza `ruta/mapa/teclado` por la ruta y el nombre del fichero de tu mapa de teclado. Por ejemplo, si tienes un teclado español, puedes poner `/usr/share/kbd/keymaps/i386/qwerty/es.map.gz`.

La segunda opción para configurar la disposición de tu teclado es compilar el mapa de teclado directamente en el núcleo. Esto asegurará que tu teclado siempre funcione como se espera, incluso cuando has arrancado en modo de rescate (pasando `init=/bin/sh' al núcleo) y los guiones de arranque que normalmente se encargan de cargar el mapa de teclado adecuado no se hayan ejecutado.

Cuando en el Capítulo 8[p.174] estés preparado para compilar el núcleo, ejecuta el siguiente comando para parchear el mapa de teclado actual dentro de las fuentes (deberás repetir este comando siempre que desempaquetes un nuevo núcleo):

```
loadkeys -m /usr/share/kbd/keymaps/defkeymap.map.gz > \
    [ruta al directorio de fuentes]/linux-2.4.26/drivers/char/defkeymap.c
```

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 es 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/VGA de la consola.

setkeycodes carga las entradas de la tabla de correspondencia de código de exploración (scan code) a código de tecla del núcleo.

setleds establece los LEDs y las opciones del teclado. Mucha gente encuentra útil tener el bloqueo numérico (Num Lock) activado por defecto, setleds +num hace esto.

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 (no animados, sólo una pantalla en blanco).

showconsolefont muestra la actual fuente de pantalla de la consola EGA/VGA.

showkey examina los códigos de exploración (scan codes) y los códigos de tecla enviados por el teclado.

unicode_start pone el teclado y la consola en modo Unicode.

unicode_stop revierte el teclado y la consola del modo Unicode.

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:       48.4 MB
```

La instalación de E2fsprogs depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo.

Instalación de E2fsprogs

Se recomienda construir E2fsprogs fuera del árbol de las fuentes:

```
mkdir ../e2fsprogs-build
cd ../e2fsprogs-build
```

Prepara E2fsprogs para su compilación:

```
../e2fsprogs-1.35/configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs
```

El significado de las opciones de configure es:

- **`--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 le pasáramos esta opción al configure de E2fsprogs, los programas se colocarían en el directorio `/usr`, que no es lo que queremos.
- **`--enable-elf-shlibs`**: Esto crea las librerías compartidas utilizadas por algunos de los programas de este paquete.

Compila el paquete:

```
make
```

Si compruebas los resultados, primero asegurate de que existe un fichero mtab con **`touch /etc/mtab`** para evitar que fallen unas sesenta pruebas, y (si todavía no existe) falsea la presencia de un paginador antiguo con **`ln -s /tools/bin/cat /bin/more`** para evitar que falle una prueba. Luego ejecuta **`make check`**.

Comienza la instalación del paquete:

```
make install
```

Instala también las librerías compartidas:

```
make install-libs
```

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 es 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 de sistemas de ficheros ext3, la versión con registro de transacciones del sistema de ficheros ext2.

compile_et es 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 es 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`.

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.

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 es la rutina para mostrar errores comunes.

libe2p es usada por dumpe2fs, chattr, y lsatt.

libext2fs contiene rutinas para permitir a los programas de nivel de usuario manipular un sistema de ficheros ext2.

libss es usada por debugfs.

libuuid contiene rutinas para generar identificadores únicos para objetos que pueden estar accesibles más allá del sistema local.

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, Texinfo.

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`**.

Ahora instala el paquete:

```
make install
```

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.

Grub-0.94

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, Sed.

Instalación de Grub

Se sabe que este programa se comporta mal si cambias sus parámetros de optimización (incluyendo las opciones -march y -mcpu). Por esta razón, si tienes definida cualquier variable de entorno que pueda sobreescribir las optimizaciones por defecto, como CFLAGS o CXXFLAGS, recomendamos quitarlas o modificarlas cuando construyas Grub.

Prepara Grub para su compilación:

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

Compila el paquete:

```
make
```

Ahora instálalo:

```
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`.

Contenido de Grub

Programas instalados: grub, grub-install, grub-md5-crypt, grub-terminfo y mbchk

Descripciones cortas

grub es el intérprete de comandos de 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.

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, Sed.

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.

```
cp gzexe.in{,.backup}
sed 's%"BINDIR"%/bin%' gzexe.in.backup > 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
```

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 se utiliza para crear ficheros ejecutables autodescomprimibles.

gzip comprime los ficheros indicados usando codificación Lempel-Ziv (LZ77).

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 gzip 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 sobre formato .Z (compress) al formato .gz (gzip).

Man-1.5m2

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, Sed.

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 caractéres:

```
patch -Np1 -i ../man-1.5m2-80cols.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/' 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%#&%' src/man.conf.in
```

Ahora prepara Man para su compilación:

```
./configure -default -confdir=/etc
```

El significado de las opciones de configure:

- **`-default`**: Esto le indica al guión configure que seleccione un conjunto sensible de opciones por defecto. Por ejemplo: sólo las páginas de manual en inglés, sin catálogos de mensajes, man sin el setuid, manejo de páginas de manual comprimidas, compresión de páginas capturadas, creación de páginas capturadas cuando el directorio apropiado exista y seguir el FHS poniendo las páginas capturadas bajo /var/cache/man (suponiendo que este directorio exista).
- **`-confdir=/etc`**: Esto le indica al programa **man** que busque el fichero de configuración `man.conf` en el directorio `/etc`.

Compila el paquete:

```
make
```

Por último, instálalo:

```
make install
```

Nota

Si deseas desactivar las secuencias de escape SGR, debes editar el fichero `man.conf` y añadir el argumento *-c* a NROFF.

Puede que quieras mirar la página http://www.escomposlinux.org/lfs-es/blfs-es-CVS/postlfs/compressdoc.html del libro BLFS (el original se encuentra en http://www.linuxfromscratch.org/blfs/view/stable/postlfs/compressdoc.html), que se ocupa de las cuestiones de formateado y compresión de las páginas de manual.

Contenido de Man

Programas instalados: apropos, makewhatis, man, man2dvi, man2html y whatis

Descripciones cortas

apropos busca una cadena en la base de datos de whatis y muestra las descripciones cortas de los comandos del sistema que contengan dicha cadena.

makewhatis construye la base de datos de 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 de 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 de whatis y muestra las descripciones cortas de los comandos del sistema que contengan la palabra clave dada como una palabra completa.

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, Sed.

Instalación de Make

Prepara Make para su compilación:

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

Compila el paquete:

```
make
```

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

Ahora instala el paquete:

```
make install
```

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.

Modutils-2.4.27

El paquete Modutils contiene programas para manejar módulos del núcleo.

```
Tiempo estimado de construcción:  0.1 SBU
Tiempo estimado de construcción:  2.9 MB
```

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

Instalación de Modutils

Prepara Modutils para su compilación:

```
./configure
```

Compila el paquete:

```
make
```

Instálalo:

```
make install
```

Contenido de Modutils

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.

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, Sed.

Instalación de Patch

Prepara Patch para su compilación (establecer la opción del preprocesador *-D_GNU_SOURCE* sólo es necesario en la plataforma PowerPC, en otras plataformas puedes ignorarlo):

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

Compila el paquete:

```
make
```

Ahora instálalo:

```
make install
```

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. Usar parches en vez de un nuevo paquete completo para mantener actualizado tu código fuente puede ahorrarte un montón de tiempo de descarga.

Procinfo-18

El paquete Procinfo contiene programas para mostrar información del sistema.

```
Tiempo estimado de construcción:  0.1 SBU
Espacio requerido en disco:       0.2 MB
```

La instalación de Procinfo depende de: Binutils, GCC, Glibc, Make, Ncurses.

Instalación de Procinfo

Compila Procinfo:

```
make LDLIBS=-lncurses
```

Significado del parámetro de make:

- **`LDLIBS=-lncurses`**: Esto le indica a Procinfo que use la librería `libncurses` en lugar de la obsoleta `libtermcap`.

Instala el paquete:

```
make install
```

Contenido de Procinfo

Programas instalados: lsdev, procinfo y socklist

Descripciones cortas

lsdev lista los dispositivos presentes en tu sistema y que IRQs (Peticiones de Interrupción) y puertos IO (entrada/salida) usan.

procinfo muestra algunos datos del sistema contenidos en el sistema de ficheros virtual proc.

socklist lista todos los conectores de red (sockets) abiertos, enumerando su tipo, número de puerto y otros datos específicos.

Procps-3.2.1

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, Ncurses.

Instalación de Procps

Compila Procps:

```
make
```

Instálalo:

```
make install
```

Elimina un enlace simbólico molesto:

```
rm /lib/libproc.so
```

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 cuanto tiempo hace que el sistema está en ejecución, cuantos 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 que usuarios hay actualmente en el sistema, en que terminal y desde cuando.

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.

Psmisc-21.4

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, Sed.

Instalación de Psmisc

Prepara Psmisc para compilarlo:

```
./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
```

Ahora instálalo:

```
make install
```

No hay razón para que los programas `pstree` y `pstree.x11` residan en `/bin`. Pot 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
```

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.

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, Sed.

Instalación de Shadow

Shadow fija la ruta al binario **passwd** dentro del propio binario, pero lo hace de forma incorrecta. Si antes de instalar Shadow no existe un binario **passwd**, este asume erróneamente que el binario estará en `/bin/passwd`, pero luego lo instala como `/usr/bin/passwd`. Esto resultará en errores sobre que no se puede encontrar `/bin/passwd`. Para corregir este fallo de Shadow crearemos un fichero temporal `passwd` para que en su código se fije la ruta correcta:

```
touch /usr/bin/passwd
```

Ahora, 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
```

Compila el paquete:

```
make
```

E instálalo:

```
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*, queremos utiliza el más seguro método de encriptación de contraseñas *MD5*, que además permite contraseñas de más de 8 caractéres. También queremos 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. Haremos ambas cosas modificando el fichero de configuración correspondiente mientras lo copiamos a su destino (posiblemente sea mejor cortar y pegar esto que intentar teclearlo):

```
sed -e 's%#MD5_CRYPT_ENAB.no%MD5_CRYPT_ENAB yes%' \
    -e 's%/var/spool/mail%/var/mail%' \
    etc/login.defs.linux > /etc/login.defs
```

Mueve algunos enlaces mal ubicados a sus lugares correctos:

```
mv /bin/sg /usr/bin
mv /bin/vigr /usr/sbin
```

Y 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 este directorio 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
```

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*, pues 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**.

Establecer la contraseña de root

Elige una contraseña para el administrador (root) y establécela mediante:

```
passwd root
```

Contenido de Shadow

Programas instalados: chage, chfn, chpasswd, chsh, dpasswd, 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

Descripciones cortas

chage cambia 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 sirve 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.

dpasswd cambia las contraseñas de acceso telefónico de un usuario.

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 un grupo a partir de un 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 (shadow group file) 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 se usa para establecer una nueva sesión con el sistema.

logoutd es un demonio que refuerza las restricciones en base a horas y puertos de acceso.

mkpasswd encripta una contraseña dada usando para ello una perturbación también dada.

newgrp se usa para cambiar el identificador de grupo actual durante una sesión de acceso.

newusers crea o actualiza un grupo de cuentas de usuario de una sola vez.

passwd cambia las contraseñas de las cuentas de usuarios y grupos.

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 con el identificador de grupo del grupo indicado.

useradd crea un nuevo usuario con el nombre especificado o actualiza la información por defecto de un nuevo usuario.

userdel borra una cuenta de usuario.

usermod modifica el nombre, identificador (UID), intérprete de comandos, grupo inicial, directorio personal, etc, de un usuario.

vigr puede usarse para editar los ficheros `/etc/group` y `/etc/gshadow`.

vipw puede usarse para editar los ficheros `/etc/passwd` y `/etc/shadow`.

libmisc...

libshadow contiene funciones usadas por la mayoría de los programas de este paquete.

Sysklogd-1.4.1

El paquete Sysklogd contiene programas para grabar 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, Make.

Instalación de Sysklogd

Compila Sysklogd:

```
make
```

Ahora instálalo:

```
make install
```

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
```

Contenido de Sysklogd

Programas instalados: klogd y syslogd

Descripciones cortas

klogd es un demonio del sistema que intercepta y registra los mensajes del núcleo.

syslogd registra los mensajes que los programas del sistema ofrecen. Cada mensaje registrado contiene como mínimo un campo con la fecha y el nombre de la máquina y, normalmente, también el nombre del programa, pero eso depende de lo confiable que sea el demonio de registros.

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, Make.

Instalación de Sysvinit

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 la fuente 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):

```
cp src/init.c{,.backup}
sed 's/Sending processes/& started by init/g' \
    src/init.c.backup > src/init.c
```

Compila Sysvinit:

```
make -C src
```

E instálalo:

```
make -C src install
```

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 tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# Fin de /etc/inittab
EOF
```

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. Pero primero anota en `/var/log/wtmp` que el sistema se va a cerrar.

init es el padre de todos los procesos. Lee sus comandos desde `/etc/inittab`, el cual normalmente le indica que guiones ejecutar en cada nivel de ejecución y cuantos procesos getty iniciar.

killall5 envía una señal a todos los procesos, excepto a los procesos de su propia sesión -- por tanto no puede matar el intérprete de comandos en el que se esté ejecutando el guión desde el que fue llamado.

last muestra los últimos usuarios conectados (y desconectados), buscando hacia atrás en el fichero `/var/log/wtmp`. También puede mostrar los inicios y paradas del sistema y los cambios del 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 pare 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, como figura en el último registro de nivel de ejecución de `/var/run/utmp`.

shutdown provoca la caída del sistema de una forma segura, enviando señales a todos los procesos y notificando a todos los usuarios conectados.

sulogin permite el ingreso del superusuario al sistema. Suele ser invocado por init cuando el sistema entra en el modo monousuario.

telinit le indica a init en qué nivel de ejecución debe entrar.

utmpdump muestra el contenido de un fichero de acceso dado en un formato comprensible por el usuario.

wall envía un mensaje a todos los usuarios conectados.

Tar-1.13.94

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, Sed.

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`**.

Ahora instala el paquete:

```
make install
```

Contenido de Tar

Programas instalados: rmt y tar

Descripciones cortas

rmt es utilizado para manipular remotamente una unidad de cinta magnética mediante una comunicación de conexión entre procesos.

tar se usa para almacenar y extraer ficheros de un archivo, también conocido como paquete tar (tarball).

Util-linux-2.12a

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, Zlib.

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:

```
cp hwclock/hwclock.c{,.backup}
sed 's%etc/adjtime%var/lib/hwclock/adjtime%' \
    hwclock/hwclock.c.backup > hwclock/hwclock.c
mkdir -p /var/lib/hwclock
```

Instalación de Util-linux

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.

Ahora instala el paquete:

```
make HAVE_KILL=yes HAVE_SLN=yes install
```

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, parse.bash, parse.tcsh, pg, pivot_root, ramsize (enlace a rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (enlace a rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (enlace a swapon), swapon, test.bash, test.tcsh, 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 /bin/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 filtra avances de línea inversos de la entrada.

colcrt filtra la salida de nroff para terminales a los que les faltan ciertas características como el sobrerefresco o semilíneas.

colrm filtra 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 se utilizaba para ajustar 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 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) con 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 es un filtro para paginar texto pantalla a pantalla. Pero less es mucho mejor.

mount monta el sistema de ficheros de un dispositivo dado en el directorio indicado (ocultando el contenido de dicho directorio) del árbol de sistemas de ficheros.

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.

rdev muestra y establece el dispositivo raíz y otras cosas en una imagen de arranque.

readprofile lee la información de los 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 escrito de una sesión de terminal, o de todo lo impreso en el 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 es 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 se usa para ajustar los parámetros de la línea de impresión.

ul es 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.

GCC-2.95.3

```
Tiempo estimado de construcción:  1.5 SBU
Tiempo estimado de construcción:  130 MB
```

Instalación de GCC

Se sabe que este programa se comporta mal si cambias sus parámetros de optimización (incluyendo las opciones -march y -mcpu). Por esta razón, si tienes definida cualquier variable de entorno que pueda sobreescribir las optimizaciones por defecto, como CFLAGS o CXXFLAGS, recomendamos quitarlas o modificarlas cuando construyas GCC.

Esta es una versión antigua de GCC que vamos a instalar con el propósito de compilar el núcleo Linux en el Capítulo 8[p.174]. Esta versión es la recomendada por los desarrolladores del núcleo cuando necesitas una estabilidad absoluta. Las versiones posteriores de GCC no han sido lo suficientemente probadas para compilar el núcleo Linux. Usar una versión posterior podría funcionar, pero recomendamos sumarse al aviso de los desarrolladores del núcleo y utilizar esta versión para compilar tu núcleo.

Nota

No instalaremos aquí el compilador C++ y las librerías. Sin embargo, puede haber razones por las que quisieras instalarlos. Puedes encontrar mas información en http://www.escomposlinux.org/lfs-es/blfs-es-CVS/general/gcc2.html (el original se encuentra en http://www.linuxfromscratch.org/blfs/view/stable/general/gcc2.html).

Instalaremos esta versión antigua de GCC dentro del prefijo no estándar `/opt` para evitar interferencias con el GCC del sistema instalado en `/usr`.

Aplica los parches y haz un pequeño ajuste:

```
patch -Np1 -i ../gcc-2.95.3-2.patch
patch -Np1 -i ../gcc-2.95.3-no-fixinc.patch
patch -Np1 -i ../gcc-2.95.3-returntype-fix.patch
echo timestamp > gcc/cstamp-h.in
```

La documentación de GCC recomienda construir GCC fuera del directorio de las fuentes, en un directorio de construcción dedicado

```
mkdir ../gcc-2-build
cd ../gcc-2-build
```

Compila e instala el compilador:

```
../gcc-2.95.3/configure --prefix=/opt/gcc-2.95.3 \
    --enable-shared --enable-languages=c \
    --enable-threads=posix
make bootstrap
make install
```

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 de qué compilador se usó y con qué librería C. 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 todos los programas y librerías. Puedes encontrar información sobre otras formas de optimizar tu sistema en la receta http://www.escomposlinux.org/lfs-es/recetas/optimization.html (el original se encuentra en http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt).

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.

Si piensas seguir adelante y ejecutar strip, es necesario 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 “Entrar al entorno chroot”[p.71], 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
```

Ahora puedes usar strip con tranquilidad en 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, no hay peligro.

Si realmente andas corto de espacio en disco, puede que quieras utilizar *--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.

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`, puede que quieras borrar el directorio por completo para ganar espacio. Antes de borrar realmente el directorio, sal del chroot y reentra con el anterior comando. Igualmente, antes de eliminar `/tools` es posible que quieras empaquetarlo y guardarlo en lugar seguro, en caso de que quieras construir pronto 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[p.28], excepto por el cambio de la ruta de */tools* a */usr*. El libro BLFS expone un método ligeramente diferente para instalar Tcl, mira http://www.linuxfromscratch.org/blfs/.

Es posible que también quieras mover los paquetes y parches de `/sources` a una localización más normal, como `/usr/src/packages`, y eliminar el directorio, o simplemente borrar por completo el directorio si has quemado su contenido en un CD.

Capítulo 7. Configurar los guiones de arranque del sistema

Introducción

En este capítulo instalaremos los guiones de arranque y los configuraremos adecuadamente. Muchos de estos guiones funcionarán sin necesidad de modificarlos, pero algunos necesitan ficheros de configuración adicionales, pues manejan información dependiente del hardware.

Hemos elegido guiones de inicio al estilo System-V simplemente porque son ampliamente utilizados y nos sentimos cómodos con ellos. Si prefieres probar alguna otra cosa, Marc Heerdink ha escrito una receta sobre los guiones de arranque al estilo BSD, que puedes encontrar en http://www.escomposlinux.org/lfs-es/recetas/bsd-init.html (la versión original se encuentra en http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt). Y si quieres algo más radical, busca “depinit” en las listas de correo de LFS.

Si decides usar algun otro estilo de guiones de inicio, puedes saltarte este capítulo y pasar al Capítulo 8[p.174].

LFS-Bootscripts-2.0.5

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, Coreutils.

Instalación de LFS-Bootscripts

La instalación de los guiones de arranque es muy simple:

```
make install
```

Contenido de LFS-bootscripts

Guiones instalados: checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysklogd y template

Descripciones cortas

El guión **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).

El guión **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.

El guión **functions** contiene funciones usadas por diferentes guiones, como el chequeo de errores y de estado.

El guión **halt** se encarga de cerrar el sistema.

Los guiones **ifdown** e **ifup** ayudan al guión network con los dispositivos de red.

El guión **loadkeys** carga el mapa que especifiques como apropiado para tu modelo de teclado.

El guión **localnet** establece el nombre de máquina usado por el sistema (hostname) y activa el dispositivo de red "loopback".

El guión **mountfs** monta todos los sistemas de ficheros que no estén marcados como *noauto* o que no se monten a través de la red.

El guión **mountkernfs** se usa para montar los sistemas de ficheros suministrados por el núcleo, como /proc.

El guión **network** activa las interfaces de red, como las tarjetas de red, y establece la puerta de enlace por defecto (gateway) cuando es necesario.

El guión **rc** es 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.

El guión **reboot** se encarga de reiniciar el sistema.

El guión **sendsignals** se asegura de que todos los procesos terminen antes de parar o reiniciar el sistema.

El guión **setclock** fija el reloj del núcleo a la hora local en caso de que el reloj del ordenador no esté fijado a la hora GMT.

El guión **static** suministra la funcionalidad necesaria para asignar una dirección IP estática a una interfaz de red.

El guión **swap** activa y desactiva las particiones y ficheros de intercambio (swap).

El guión **sysklogd** lanza y detiene los demonios de registro de eventos del sistema y del núcleo.

El guión **template** es una plantilla que puedes utilizar para crear tus propios guiones de arranque para otros demonios.

¿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 <inserte 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 deseemos 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 enlaces simbólicos apuntan a los guiones situados en el directorio /etc/rc.d/init.d, que son los que realmente se ejecutan. Tanto los enlaces de parada como los de inicio apuntan al mismo guión. Esto se debe a que se pueden ejecutar usando parámetros como start, stop, restart, reload o status. Cuando se encuentra un enlace que comienza por K se ejecuta el guión con el parámetro stop. Y cuando comienza por S, con el parámetro start.

Hay una excepción. Los enlaces que comienzan por S en los directorios rc0.d y rc6.d no inician nada. Todos estos guiones se ejecutan con el parámetro *stop* para parar algo. Es evidente que cuando quieres apagar o reiniciar el sistema, no quieres ejecutar nada, sólo quieres pararlo.

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 se necesita reiniciar el servicio para que actualice su configuración.
- *status*: Dice si el servicio se está ejecutando y con qué identificador de proceso (PID).

Por supuesto, puedes modificar el proceso de inicio para adecuarlo a tus necesidades (después de todo es tu sistema LFS). Lo aquí expuesto es tan sólo un ejemplo de cómo hacer las cosas de una manera correcta (claro que aunque a nosotros esta manera nos parezca bien, puede que tú la odies).

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), y la convierte a la hora local mediante el fichero `/etc/localtime` (si el reloj interno del ordenador utiliza GMT) o no (si el reloj interno de la computadora ya está puesto a la hora local). No hay manera de detectar automáticamente si el reloj utiliza GMT o no, así que necesitamos configurarlo nosotros mismos.

Si el reloj interno del ordenador no utiliza GMT hay que cambiar el valor de la variable *UTC* a *0* (cero).

Para ello vamos a crear el fichero `/etc/sysconfig/clock` mediante la ejecución del siguiente comando:

```
cat > /etc/sysconfig/clock << "EOF"
# Inicio de /etc/sysconfig/clock

UTC=1

# Fin de /etc/sysconfig/clock
EOF
```

Para más información sobre la hora en LFS tienes una buena explicación en http://www.escomposlinux.org/lfs-es/recetas/time.html (la versión original se encuentra en http://www.linuxfromscratch.org/hints/downloads/files/time.txt). En ella se explican conceptos como las zonas horarias, UTC, y la variable de entorno TZ.

¿Necesito el guión loadkeys?

Si piensas compilar el mapa del teclado directamente en el núcleo durante el Capítulo 8[p.174] (mira Kbd[p.133]), entonces no es necesario que ejecutes el guión loadkeys, pues el núcleo establecerá el mapa del teclado por ti. Puedes ejecutarlo de todas maneras si quieres, no te causará ningún problema. Incluso puede ser beneficioso que mantengas el guión en el caso de que uses diversos núcleos y no estés seguro de que el mapa de teclado está compilado dentro de todos ellos.

Si has decidido que no necesitas o no quieres usar el guión loadkeys, elimina el enlace simbólico `/etc/rc.d/rcsysinit.d/S70loadkeys`

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. Si quieres habilitar esta marca de tiempo periódica debes editar el guión `sysklogd` y realizar los cambios necesarios. Para más información utiliza el comando **`man syslogd`**.

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.

Puedes crear el fichero /etc/sysconfig/network y configurar el nombre de tu máquina ejecutando:

```
echo "HOSTNAME=lfs" > /etc/sysconfig/network
```

Debes substituir “lfs” por el nombre de tu máquina. No debes escribir el FQDN (nombre completo de la máquina, incluido su dominio). Esta información la escribiremos más tarde en el fichero `/etc/hosts`

Creación del fichero /etc/hosts

Si se va a configurar una tarjeta de red, debes decidir 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, muchos de nosotros 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     Entre 172.16.0.0 y 172.31.0.0
        C     Entre 192.168.0.0 y 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 tu servidor de nombres de dominio).

Aunque no vayas a configurar la tarjeta de red necesitas un FQDN, ya que algunos programas lo necesitan para funcionar correctamente.

Si no vas a configurar la 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 <valor de HOSTNAME>.example.org <valor de HOSTNAME> localhost

# Fin de /etc/hosts (versión sin tarjeta de red)
EOF
```

Si vas a configurar la tarjeta de red crea el fichero `/etc/hosts` ejecutando:

```
cat > /etc/hosts << "EOF"
# Inicio de /etc/hosts (versión con tarjeta de red)

127.0.0.1 localhost
192.168.1.1 <valor de HOSTNAME>.example.org <valor de HOSTNAME>

# Fin de /etc/hosts (versión con tarjeta de red)
EOF
```

Por supuesto, debes cambiar 192.168.1.1 y <valor de HOSTNAME>.example.org a tu gusto (o a lo que te indique el administrador de la red/sistema si él te asigna una dirección IP y está planeado que esta máquina se conecte a una red ya existente).

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, debes eliminar los enlaces simbólicos a `network` de todos los directorios de los niveles de ejecución (`/etc/rc.d/rc*.d`)

Configuración de la puerta de enlace por defecto

Si estás conectado a una red puede que necesites establecer cual es la puerta de enlace por defecto (un nodo en tu red que suministra el ascceso a otras redes) para esa máquina. Para ello, se deben añadir los valores apropiados al fichero /etc/sysconfig/network ejecutando lo siguiente:

```
cat >> /etc/sysconfig/network << "EOF"
GATEWAY=192.168.1.2
GATEWAY_IF=eth0
EOF
```

Debes cambiar los valores de GATEWAY y GATEWAY_IF por los que correspondan en tu red. GATEWAY contiene la dirección IP de la puerta de enlace por omisión, y GATEWAY_IF la interfaz de red por la que es accesible dicha dirección IP.

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 con el nombre ifconfig.xyz, donde xyz corresponde con 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 que actualizas el fichero /etc/sysconfig/rc, asignando a la variable network_devices la nueva localización.

Ahora, en este directorio se crean nuevos ficheros. El siguiente comando crea un fichero ifconfig.eth0 de ejemplo:

```
cat > /etc/sysconfig/network-devices/ifconfig.eth0 << "EOF"
ONBOOT=yes
SERVICE=static
IP=192.168.1.1
NETMASK=255.255.255.0
BROADCAST=192.168.1.255
EOF
```

Por supuesto, 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 el NIC (Interfaz de Tarjeta de Red) correspondiente durante el arranque del sistema. Si contiene cualquier otro valor, el guión network ignorará el 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 puedes permitir otros métodos de asignación IP. Comúnmente podría usarse si necesitas DHCP, explicado en el libro BLFS.

Creación del fichero /etc/resolv.conf

Si vas a estar conectado a Internet, entonces lo más probable es que necesites algún tipo de resolución de nombres DNS para resolver los nombres de dominio de Internet a direcciones IP. Esto se consigue mejor colocando la dirección IP de tu servidor DNS, facilitado por tu ISP (Proveeder de Servicios de Internet) o administrador de red, en `/etc/resolv.conf`. Crea el fichero ejecutando lo siguiente:

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

nameserver <dirección IP de tu servidor de nombres>
```

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

Por supuesto, sustituye <dirección IP de tu 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). La dirección IP pueder ser incluso un enrutador de tu red local.

Capítulo 8. Hacer el sistema LFS arrancable

Introducción

Este capítulo hará arrancable el sistema LFS. Trataremos 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.

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 del   opciones        volcado   orden de
# archivos     montaje    sist. de                            chequeo
#                         ficheros

/dev/xxx       /          fff        defaults        1         1
/dev/yyy       swap       swap       pri=1           0         0
proc           /proc      proc       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
```

Por supuesto, reemplaza `xxx`, `yyy` y `fff` por los valores apropiados para tu sistema, por ejemplo `hda2`, `hda5` y `reiserfs`. Para ver todos los detalles de los seis campos de esta tabla, consulta **man 5 fstab**.

Cuando se añada una partición reiserfs, 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. Hay 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 puedes considerar añadir al fichero `fstab`. Un ejemplo es la línea que debe ponerse si pretendes utilizar dispositivos USB:

```
usbfs       /proc/bus/usb usbfs defaults    0       0
```

Esta opción sólo funcionará si se tiene el soporte pertinente compilado dentro del núcleo.

Linux-2.4.26

El paquete Linux contiene el núcleo y los ficheros de cabecera.

```
Tiempo estimado de construcción:  Todas las opciones por defecto: 4.20 SBU
Espacio requerido en disco:       Todas las opciones por defecto: 181 MB
```

La instalación de Linux depende de: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Instalación del núcleo

Construir el núcleo comprende varios pasos: configuración, compilación e instalación. Si no te gusta la forma en que se configura el núcleo en este libro, mira los métodos alternativos en el fichero `README` del árbol de las fuentes del núcleo.

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.

Configura el núcleo mediante una interfaz de menús:

```
make menuconfig
```

Puede que **`make oldconfig`** sea más adecuado en algunas situaciones. Lee el fichero `README` para más detalles.

Si lo deseas, puedes saltarte la configuración del núcleo simplemente copiando el fichero de configuración del núcleo, `.config`, de tu sistema anfitrión (asumiendo que esté disponible) al directorio `linux-2.4.26`. Sin embargo, no recomendamos esta opción. Te será 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.

Comprueba las dependencias y crea los ficheros de información de las dependencias:

```
make CC=/opt/gcc-2.95.3/bin/gcc dep
```

Compila la imagen del núcleo:

```
make CC=/opt/gcc-2.95.3/bin/gcc bzImage
```

Compila los controladores que han sido configurados como módulos:

```
make CC=/opt/gcc-2.95.3/bin/gcc modules
```

Si planeas usar los módulos del núcleo necesitarás un fichero `/etc/modules.conf`. La información relativa a los módulos y a la configuración del núcleo en general puedes encontrarla en la documentación del núcleo, que se encuentra en el directorio `/usr/src/linux-2.4.26/Documentation`. La página de manual de modules.conf y el kernel-CÓMO en http://es.tldp.org/COMO-INSFLUG/COMOs/Kernel-Como/ (el original en inglés se encuentra en http://www.tldp.org/HOWTO/Kernel-HOWTO.html) puede que también sean de interés para ti.

Instala los módulos:

```
make CC=/opt/gcc-2.95.3/bin/gcc modules_install
```

Si tienes muchos módulos y muy poco espacio, puede que quieras considerar eliminarles los símbolos y comprimirlos. Para muchos dicha compresión no evita el problema, pero si realmente estás presionado por el espacio, mira

http://www.linux-mips.org/archives/linux-mips/2002-04/msg00031.html.

Como nada está completo sin su documentación, construye las páginas de manual que vienen con el núcleo:

```
make mandocs
```

E instala dichas páginas:

```
cp -a Documentation/man /usr/share/man/man9
```

La compilación del núcleo ha terminado, pero para completar la instalación se necesitan algunos pasos más. 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
```

`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
```

`.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-lfskernel
```

Es importante notar 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 querrás ejecutar **`chown -R 0:0`** sobre el directorio `linux-2.4.26` para asegurar que todos los ficheros son propiedad de *root*.

Contenido de Linux

Ficheros instalados: el núcleo y las cabeceras del núcleo.

Descripciones cortas

El *núcleo* es el corazón de tu 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 de tu 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.

Las *cabeceras del núcleo* definen la interfaz a los servicios proporcionados por el núcleo. Las cabeceras en tu directorio del sistema `include` deben *siempre* ser aquellas contra las que se compiló Glibc y, por tanto, *nunca* deben reemplazarse al actualizar el núcleo.

El fichero `System.map` es un listado de direcciones y símbolos. Mapea los puntos de entrada y direcciones de todas las funciones y estructuras de datos del núcleo.

Hacer el sistema LFS arrancable

Tu nuevo y brillante sistema LFS está casi completo. Una de las últimas cosas por hacer es asegurarte de que puedes arrancarlo. 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. En realidad deberías estar familiarizado 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 que puedas recuperar tu ordenador si, por cualquier motivo, quedase inutilizable (no arrancable).

Anteriormente, compilamos e instalamos el gestor de arranque Grub en preparación de 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 por si acaso. 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 iniciaremos 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. Esto significa que la partición `hda1` es (hd0,0) para Grub, y `hdb2` es (hd1,1). Al contrario que Linux, Grub no considera los dispositivos CD-ROM como discos duros, por lo que si, por ejemplo, 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`

Primero, indícale a Grub donde 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 quieres. Por ejemplo, puede que estés usando 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 (Master Boot Record, Registro Maestro de Arranque) de `hda`:

```
setup (hd0)
```

Si todo está bien, Grub informará que ha encontrado sus ficheros en `/boot/grub`. Todo lo que hay allí es para él:

```
quit
```

Ahora necesitamos crear el fichero “menu.lst”, que define 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 5.1.1
root (hd0,3)
kernel --no-mem-option /boot/lfskernel root=/dev/hda4
EOF
```

Nota

Por defecto, Grub automáticamente le pasará al núcleo el argumento “mem=xxx” en su línea de comandos. Sin embargo, en ocasiones Grub detecta una cantidad errónea de memoria que puede crear problemas en ciertas situaciones. Es mejor desactivar esta función y dejar que el núcleo determine por si mismo la cantidad de memoria, de aquí el uso de *--no-mem-option*.

Puede que también quieras añadir una entrada para tu 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
```

Igualmente, 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

El final

¡Bien hecho! Has terminado de instalar tu sistema LFS. Puede que haya sido un proceso largo pero esperamos que haya merecido la pena. 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 5.1.1 > /etc/lfs-release
```

Registrarse

¿Quieres registrarte como usuario de LFS ahora que has terminado el libro? Visita http://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 el sistema LFS ahora...

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 nestra nueva instalación de LFS por primera vez!. Primero sal del entorno chroot:

```
logout
```

Luego desmonta los sistemas de ficheros virtuales:

```
umount $LFS/dev/pts
umount $LFS/proc
```

Y desmonta el sistema de ficheros del LFS:

```
umount $LFS
```

Si al comienzo decidiste crear varias particiones, necesitas desmontar las otras particiones antes de desmontar la principal, por ejemplo:

```
umount $LFS/usr
umount $LFS/home
umount $LFS
```

Ahora reinicia tu 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 5.1.1* arranque automáticamente.

Una vez hayas reiniciado, tu sistema LFS está listo para su uso y puedes empezar a añadir los programas que desees.

Y ahora, ¿qué?

Gracias por leer el libro LFS. Esperamos que lo hayas encontrado útil y te recompense el tiempo empleado.

Ahora que has terminado de instalar tu sistema LFS, 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.escomposlinux.org/lfs-es/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.escomposlinux.org/lfs-es/recetas/ puedes encontrar la traducción de un buen número de ellas, aunque se encuentran algo 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. Para más información consulta Capítulo 1 - Listas de correo[p.6].

- 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.

Índice de paquetes y ficheros importantes

Pquetes

Programas

bunzip2:	Bzip2-1.0.2[p.129]	descripción[p.129]
bzcat:	Bzip2-1.0.2[p.129]	descripción[p.129]
bzcmp:	Bzip2-1.0.2[p.129]	descripción[p.129]
bzdiff:	Bzip2-1.0.2[p.129]	descripción[p.129]
bzgrep:	Bzip2-1.0.2[p.129]	descripción[p.129]
bzip2:	Bzip2-1.0.2[p.129]	descripción[p.129]
bzip2recover:	Bzip2-1.0.2[p.129]	descripción[p.130]
bzless:	Bzip2-1.0.2[p.129]	descripción[p.130]
bzmore:	Bzip2-1.0.2[p.129]	descripción[p.130]
c++filt:	Binutils-2.14[p.87]	descripción[p.88]
c2ph:	Perl-5.8.4[p.119]	descripción[p.119]
cal:	Util-linux-2.12a[p.157]	descripción[p.158]
captoinfo:	Ncurses-5.4[p.102]	descripción[p.102]
cat:	Coreutils-5.2.1[p.91]	descripción[p.92]
catchsegv:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.82]
cfdisk:	Util-linux-2.12a[p.157]	descripción[p.158]
chage:	Shadow-4.0.4.1[p.150]	descripción[p.151]
chattr:	E2fsprogs-1.35[p.135]	descripción[p.136]
chfn:	Shadow-4.0.4.1[p.150]	descripción[p.151]
chgrp:	Coreutils-5.2.1[p.91]	descripción[p.92]
chkdupexe:	Util-linux-2.12a[p.157]	descripción[p.158]
chmod:	Coreutils-5.2.1[p.91]	descripción[p.92]
chown:	Coreutils-5.2.1[p.91]	descripción[p.92]
chpasswd:	Shadow-4.0.4.1[p.150]	descripción[p.151]
chroot:	Coreutils-5.2.1[p.91]	descripción[p.92]
chsh:	Shadow-4.0.4.1[p.150]	descripción[p.151]
chvt:	Kbd-1.12[p.133]	descripción[p.134]
cksum:	Coreutils-5.2.1[p.91]	descripción[p.92]
clear:	Ncurses-5.4[p.102]	descripción[p.102]
cmp:	Diffutils-2.8.1[p.131]	descripción[p.131]
code:	Findutils-4.1.20[p.100]	descripción[p.100]
col:	Util-linux-2.12a[p.157]	descripción[p.158]
colcrt:	Util-linux-2.12a[p.157]	descripción[p.158]
colrm:	Util-linux-2.12a[p.157]	descripción[p.158]
column:	Util-linux-2.12a[p.157]	descripción[p.158]
comm:	Coreutils-5.2.1[p.91]	descripción[p.92]
compile:	Automake-1.8.4[p.124]	descripción[p.124]
compile_et:	E2fsprogs-1.35[p.135]	descripción[p.136]
config.charset:	Gettext-0.14.1[p.113]	descripción[p.113]
config.guess:	Automake-1.8.4[p.124]	descripción[p.124]
config.rpath:	Gettext-0.14.1[p.113]	descripción[p.113]
config.su:	Automake-1.8.4[p.124]	descripción[p.124]
cp:	Coreutils-5.2.1[p.91]	descripción[p.92]
cpp:	GCC-3.3.3[p.89]	descripción[p.90]
csplit:	Coreutils-5.2.1[p.91]	descripción[p.92]
ctrlaltdel:	Util-linux-2.12a[p.157]	descripción[p.158]
cut:	Coreutils-5.2.1[p.91]	descripción[p.92]
cytune:	Util-linux-2.12a[p.157]	descripción[p.158]
date:	Coreutils-5.2.1[p.91]	descripción[p.92]
dd:	Coreutils-5.2.1[p.91]	descripción[p.92]
ddate:	Util-linux-2.12a[p.157]	descripción[p.158]
deallocvt:	Kbd-1.12[p.133]	descripción[p.134]
debugfs:	E2fsprogs-1.35[p.135]	descripción[p.136]
depcomp:	Automake-1.8.4[p.124]	descripción[p.124]
depmod:	Modutils-2.4.27[p.144]	descripción[p.144]
df:	Coreutils-5.2.1[p.91]	descripción[p.92]
diff:	Diffutils-2.8.1[p.131]	descripción[p.131]
diff3:	Diffutils-2.8.1[p.131]	descripción[p.131]
dir:	Coreutils-5.2.1[p.91]	descripción[p.93]
dircolors:	Coreutils-5.2.1[p.91]	descripción[p.93]
dirname:	Coreutils-5.2.1[p.91]	descripción[p.93]
dmesg:	Util-linux-2.12a[p.157]	descripción[p.158]
dnsdomainname:	Net-tools-1.60[p.115]	descripción[p.115]

whoami:	Coreutils-5.2.1[p.91]	descripción[p.95]
write:	Util-linux-2.12a[p.157]	descripción[p.159]
xargs:	Findutils-4.1.20[p.100]	descripción[p.100]
xgettext:	Gettext-0.14.1[p.113]	descripción[p.114]
xsubpp:	Perl-5.8.4[p.119]	descripción[p.120]
xtrace:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
xxd:	Vim-6.2[p.104]	descripción[p.105]
yacc:	Bison-1.875[p.107]	descripción[p.107]
yes:	Coreutils-5.2.1[p.91]	descripción[p.95]
ylwrap:	Automake-1.8.4[p.124]	descripción[p.125]
ypdomainname:	Net-tools-1.60[p.115]	descripción[p.116]
zcat:	Gzip-1.3.5[p.139]	descripción[p.139]
zcmp:	Gzip-1.3.5[p.139]	descripción[p.139]
zdiff:	Gzip-1.3.5[p.139]	descripción[p.139]
zdump:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
zegrep:	Gzip-1.3.5[p.139]	descripción[p.139]
zfgrep:	Gzip-1.3.5[p.139]	descripción[p.139]
zforce:	Gzip-1.3.5[p.139]	descripción[p.139]
zgrep:	Gzip-1.3.5[p.139]	descripción[p.139]
zic:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
zless:	Gzip-1.3.5[p.139]	descripción[p.140]
zmore:	Gzip-1.3.5[p.139]	descripción[p.140]
znew:	Gzip-1.3.5[p.139]	descripción[p.140]
zsoelim:	Groff-1.19[p.109]	descripción[p.110]

Librerías

ld.so:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libanl:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libasprintf:	Gettext-0.14.1[p.113]	descripción[p.114]
libbfd:	Binutils-2.14[p.87]	descripción[p.88]
libblkid:	E2fsprogs-1.35[p.135]	descripción[p.136]
libBrokenLocale:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libbsd-compat:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libbz2*:	Bzip2-1.0.2[p.129]	descripción[p.130]
libc:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libcom_err:	E2fsprogs-1.35[p.135]	descripción[p.136]
libcrypt:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libdl:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libe2p:	E2fsprogs-1.35[p.135]	descripción[p.136]
libext2fs:	E2fsprogs-1.35[p.135]	descripción[p.136]
libfl.a:	Flex-2.5.4a[p.112]	descripción[p.112]
libform*:	Ncurses-5.4[p.102]	descripción[p.103]
libg:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libgcc*:	GCC-3.3.3[p.89]	descripción[p.90]
libgettextlib:	Gettext-0.14.1[p.113]	descripción[p.114]
libgettextpo:	Gettext-0.14.1[p.113]	descripción[p.114]
libgettextsrc:	Gettext-0.14.1[p.113]	descripción[p.114]
libiberty:	Binutils-2.14[p.87]	descripción[p.88]
libieee:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libltdl:	Libtool-1.5.6[p.128]	descripción[p.128]
libm:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libmagic:	File-4.09[p.127]	descripción[p.127]
libmcheck:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libmemusage:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libmenu*:	Ncurses-5.4[p.102]	descripción[p.103]
libmisc:	Shadow-4.0.4.1[p.150]	descripción[p.152]
libncurses*:	Ncurses-5.4[p.102]	descripción[p.103]
libnsl:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.83]
libnss*:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.84]
libopcodes:	Binutils-2.14[p.87]	descripción[p.88]
libpanel*:	Ncurses-5.4[p.102]	descripción[p.103]
libpcprofile:	Glibc-2.3.3-lfs-5.1[p.80]	descripción[p.84]
libproc:	Procps-3.2.1[p.147]	descripción[p.148]

Guiones

Otros

/var/log/btmp:	Creación de los ficheros de contraseñas, grupos y registro[p.75]	
/var/log/lastlog:	Creación de los ficheros de contraseñas, grupos y registro[p.75]	
/var/log/wtmp:	Creación de los ficheros de contraseñas, grupos y registro[p.75]	
/var/run/utmp:	Creación de los ficheros de contraseñas, grupos y registro[p.75]	
kernel headers:	Linux-2.4.26[p.176]	descripción[p.177]
manual pages:	Man-pages-1.66[p.79]	descripción[p.79]