Linux From Scratch

Linux From Scratch

Versión 5.0

Gerard Beekmans

Traducido por el proyecto LFS−ES

Versión de la traducción: FINAL del 14 de Diciembre de 2003

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

OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Dedicatoria

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

Índice

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 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 prévias 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

Estructura

Este libro se divide en las siguientes cuatro partes:

Parte I − Introducción

En la Parte I se explican algunas cosas importantes sobre cómo proceder durante la instalación y facilita información sobre el propio libro (versión, historial de modificaciones, reconocimientos, listas de correo asociadas y más cosas).

Parte II − Preparativos para la construcción

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

Parte III − Construcción del sistema LFS

La Parte III te guia 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.

Parte IV − Apéndices

La Parte IV se compone de dos apéndices. El primero es un listado alfabético de todos los paquetes instalados, mostrando para cada uno su localización oficial de descarga, su contenido y sus dependencias de instalación. En el segundo apéndice se listan, por orden alfabético, todos los programas y librerías instalados por estos paquetes, para que puedas encontrar fácilmente a qué paquete pertenece cierto programa o librería.

(Gran parte del primer apéndice está integrado en las partes II y III. Esto agranda algo el libro, pero creemos que facilta su lectura. De esta forma no tienes que dirigirte al apéndice mientras haces la instalación. Este ir y venir puede ser un fastido, sobre todo si estás leyendo la versión en texto plano del libro.)

I. Parte I − Introducción

Índice

Capítulo 1. Introducción

Cómo van a hacerse las cosas

Vas a construir el 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 3 crearás primero una nueva partición nativa de Linux y un sistema de ficheros, el sitio donde se compilará e instalará tu nuevo sistema LFS. Después, en el Capítulo 4, descargarás todos los paquetes y parches necesarios para construir un sistema LFS y los guardarás en el nuevo sistema de ficheros.

En el Capítulo 5 se describe la instalación de una serie de paquetes que formarán el equipo básico de desarrollo (o toolchain, conjunto de herramientas) utilizado para construir, en el Capítulo 6, el sistema real. Algunos 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 es construir un primer paso de 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 en el primer paso. Lo tercero es construir un segundo paso de las herramientas principales. Esta vez las herramientas serán enlazadas dinámicamente contra la recién construida Glibc. Todos los restantes paquetes del Capítulo 5 se construirán usando este segundo paso de las herramientas principales y 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 se da una explicación técnica completa, incluyendo algunas notas sobre las diferencias entre programas enlazados estática y dinámicamente.

En el Capítulo 6 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. Otra ventaja es que chroot te permite seguir usando el sistema anfitrión mientras se construye 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 se configuran los guiones de arranque. El núcleo y el gestor de arranque se configuran en el Capítulo 8 y el Capítulo 9 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 un resumen corto del proceso. La información detallada sobre los pasos que seguirás se expone en los capítulos y en las descripciones de los paquetes a medida que avances en ellos. Si algo no está muy claro

ahora, no te preocupes. Todo estará pronto en su sitio.

Por favor, lee con atención el Capítulo 2, ya que explica algunas cosas importantes que debes saber antes de comenzar a trabajar en el Capítulo 5 y posteriores.

Convenciones utilizadas en este libro

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 archivo, 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 externas (tales como direcciones de descarga, CÓMOs, sitios web, etc).

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

Este tipo de secciones se usa principalmente al crear archivos de configuración. El primer comando solicita al sistema que cree el archivo $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.

Versión del libro

Esta es la versión FINAL del día 14 de Diciembre de 2003 de la traducción al castellano de la versión 5.0 del libro Linux From Scratch publicado el 5 de Noviembre de 2003. Si este libro tiene más de dos meses de

antigüedad es probable que haya disponible una versión más nueva y mejor. Para encontrarlo comprueba uno de los servidores alternativos listados en http://www.linuxfromscratch.org/.

Historial de modificaciones

5.0 − 5 de Noviembre de 2003

- Actualizado a:
  - automake−1.7.6
  - bash−2.05b
  - binutils−2.14
  - e2fsprogs−1.34
  - file−4.04
  - findutils−4.1.20
  - gawk−3.1.3
  - gcc−3.3.1
  - gettext−0.12.1
  - glibc−2.3.2
  - glibc−2.3.2−sscanf−1.patch
  - grep−2.5.1
  - groff−1.19
  - gzip−1.3.5
  - less−381
  - lfs−bootscripts−1.12
  - libtool−1.5
  - linux−2.4.22
  - man−1.5m2
  - man−1.5m2−80cols.patch
  - man−1.5m2−manpath.patch
  - man−1.5m2−pager.patch
  - man−pages−1.60
  - modutils−2.4.25
  - procps−3.1.11
  - procps−3.1.11.patch
  - psmisc−21.3
  - sed−4.0.7
  - sysvinit−2.85
  - tar−1.13.25
  - texinfo−4.6
  - util−linux−2.12
  - vim−6.2
- Añadido:
  - bash−2.05b−2.patch
  - bison−1.875−attribute.patch
  - coreutils−5.0
  - coreutils−5.0−uname.patch
  - coreutils−5.0−hostname−2.patch
  - dejagnu−1.4.3

    - expect−5.39.0
    - expect−5.39.0.patch
    - gawk−3.1.3.patch
    - gcc−2.95.3
    - gcc−2.95.3−2.patch
    - gcc−2.95.3−no−fixinc.patch
    - gcc−2.95.3−returntype−fix.patch
    - gcc−3.3.1−no_fixincludes−2.patch
    - gcc−3.3.1−specs−2.patch
    - gcc−3.3.1−suppress−libiberty.patch
    - grub−0.93
    - grub−0.93−gcc33−1.patch
    - inetutils−1.4.2
    - lfs−utils−0.3
    - ncurses−5.3−etip−2.patch
    - ncurses−5.3−vsscanf.patch
    - perl−5.8.0−libc−3.patch
    - shadow−4.0.3−newgroup−fix.patch
    - tcl−8.4.4
    - zlib−1.1.4−vsnprintf.patch
- Eliminado:

    - bin86−0.16.3
    - fileutils−4.1
    - fileutils−4.1.patch
    - findutils−4.1−segfault.patch
    - findutils−4.1.patch
    - glibc−2.3.1−libnss.patch
    - glibc−2.3.1−root−perl.patch
    - gzip−1.2.4b.patch
    - lilo−22.2
    - netkit−base−0.17
    - sh−utils−2.0
    - sh−utils−2.0.patch
    - sh−utils−2.0−hostname.patch
    - tar−1.13.patch
    - textutils−2.1
    - vim−6.1.patch
- 2 de Noviembre de 2003 [alex]: Apéndice A − Comentadas todas las líneas "Última versión comprobada".
- 28 de Octubre de 2003 [greg]: Acortados los seds en las secciones "Bloquear Glibc" y "Reajustar las herramientas".
- 26 de Octubre de 2003 [greg]: Capítulo 6 − Glibc: Añadido un comando para crear /etc/ld.so.conf que coincide con Glibc en el Capítulo 5. Cerrado bug 700.
- 24 de Octubre de 2003 [alex]: Apéndice A − Cambiadas las dependencias a un formato mas conciso, basado en un mensaje de Tushar.
- 23 de Octubre de 2003 [gerard] Capítulo 9 − El final: Cambiado el fichero `/etc/lfs` por `/etc/lfs−release` para que sea mas consistente con otras distribuciones.
- 23 de Octubre de 2003 [alex]: Cambiadas muchas de las referencias a "Capítulo" por unas adecuadas referencias cruzadas "xref".

- 22 de Octubre de 2003 [alex]: Capítulo 6 − Gawk y Shadow: Ajustado el texto. Y añadidas algunas marcas en varios sitios.
- 22 de Octubre de 2003 [alex]: Capítulo 6 − Entrar al entorno chroot: Eliminado el comando **`set +h`** pues no tiene sentido aquí: se rehará varias secciones mas adelante.
- 15 de Octubre de 2003 [greg]: Capítulo 9: Reescrito el comando strip. Incluidos párrafos sobre la eliminación de directorios del Capítulo 6.
- 14 de Octubre de 2003 [greg]: Capítulo 8 − Hacer el sistema LFS arrancable: Ampliados los detalles de Grub y añadido un aviso.
- 14 de Octubre de 2003 [alex]: Apéndice A − Actualizados los contenidos de Perl y Procps.
- 14 de Octubre de 2003 [alex]: Capítulos 4 y 5 − Añadida la sugerencia de usar $LFS/sources como directorio de trabajo y de almacenamiento.
- 13 de Octubre de 2003 [greg]: Capítulo 9 − Reiniciar el sistema: Reescritos los comandos umount.
- 11 de Octubre de 2003 [alex]: Modificados los espacios necesarios en disco y los SBUs según lo publicado por Bruce Dubbs.
- 11 de Octubre de 2003 [alex]: Capítulo 5 − Notas técnicas de las herramientas: Añadidas y cambiadas algunas marcas.
- 9 de Octubre de 2003 [gerard]: Actualizado a lfs−bootscripts−1.12.
- 9 de Octubre de 2003 [greg]: Realizado un reetiquetado interno para corregir un extraño problema de espacios en blanco en las páginas web generadas con tidy. Básicamente sustituir todas las apariciones de <para><screen> por <screen> (y las correspondientes etiquetas de cierre).
- 9 de Octubre de 2003 [alex]: Capítulo 6 − Red Básica: Movida una mitad a la sección Lfs−Utils y el resto a Perl.
- 8 de Octubre de 2003 [alex]: Capítulo 8 − Hacer el sistema LFS arrancable: Adaptado el estilo de los "screen" y reescritos algunos párrafos.
- 8 de Octubre de 2003 [alex]: Eliminadas una serie de entidades obsoletas.
- 7 de Octubre de 2003 [jeremy]: Añadidas notas en las pruebas de enlazado de los capítulos 5 y 6 indicando que la salida en blanco es mal asunto.
- 7 de Octubre de 2003 [alex]: Cambiadas las entidades de los parches para contener el nombre completo del fichero en vez de sólo el número de versión.
- 7 de Octubre de 2003 [jeremy]: Capítulo 1 − Añadida una nota sobre #LFS−support en el IRC.
- 7 de Octubre de 2003 [greg]: Prólogo: Añadida una nota sobre la Receta de Lecturas Previas Esenciales. Corregido el error 585.
- 6 de Octubre de, 2003 [alex]: Cambiado el estilo de las subsecciones Contenido en los Capítulos 5 y 6 y en el Apéndice A.
- 6 de Octubre de 2003 [greg]: Simplificados los seds en las secciones "Bloquear Glibc" y "Reajustar las herramientas". Reorganizada la sección "Cómo van ha hacerse las cosas".
- 5 de Octubre de 2003 [greg]: Capítulo 5: Añadida la nueva sección "Notas técnicas sobre las herramientas". Integrada la sección "Por qué usamos enlazado estático". Corregido el error 658.
- 4 de Octubre de 2003 [alex]: Correciones menores y adición de etiquetas.
- 4 de Octubre de 2003 [greg]: Capítulo 5 − Binutils Paso 1: Añadido un LDFLAGS extra para asegurar la reconstrucción estática de ld.
- 2 de Octubre de 2003 [greg]: Capítulo 6: Reañadido INSTALL=/tools/bin/install en el comando de ajuste del enlazador debido a problemas con anfitriones en los que existe un enlace simbólico ginstall. Esto hace superfluos los enlaces simbólicos "install", por lo que los eliminamos también.
- 2 de Octubre de 2003 [greg]: Capítulo 6 − Shadow: Activadas las contraseñas MD5. Corregido el error 600.
- 27 de Septiembre de 2003 [greg]: Capítulo 5 − Expect: modificada la instalación para que no se instalen los guiones redundantes. Capitulo 6 − Creación de los enlaces esenciales: Eliminados los enlaces redundantes. Capítulo 6 − man: Eliminado PATH. Corregido el error 574.
- 27 de Septiembre de 2003 [greg]: Añadido Tcl, Expect y DejaGnu al Apéndice A. Corregido el error 661.

- 26 de Septiembre de 2003 [jeremy]: Añadida una nueva solución para el problema de devpts.
- 24 de Septiembre de 2003 [greg]: Cambios varios para resolver el error 675.
- 22 de Septiembre de 2003 [greg]: Capítulo 8 − Creación del fichero /etc/fstab: Hacer que devpts se monte por defecto.
- 22 de Septiembre de 2003 [jeremy]: Añadido un parche a Net−tools para la compilación de mii−tool.
- 22 de Septiembre de 2003 [jwrober]: Capítulo 5 − Actualizada la página Por qué Estático para representar con más acierto las diferencias entre binarios enlazados estática y dinámicamente. Gracias a Ian Molton por sugerir esto. Corregido el error 602.
- 22 de Septiembre de 2003 [jeremy]: Eliminado el comando make de DejaGNU, pues no hace nada.
- 22 de Septiembre de 2003 [jeremy]: Eliminado el −k del make check de Tcl, pues no se espera que haya fallos.
- 22 de Septiembre de 2003 [jeremy]: Cambiada la referencia a la receta de man para apuntar al BLFS.
- 22 de Septiembre de 2003 [jeremy]: Añadida una nota para recordar que se monte devpts si se sale y se vuelve a entrar al chroot.
- 22 de Septiembre de 2003 [jeremy]: eliminado make check en Patch y Diffutils, pues estas pruebas no hacen nada.
- 22 de Septiembre de 2003 [greg]: Capítulo 5 − Establecer el entorno: Añadido unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD en .bash_profile para evitar problemas en la construcción.
- 20 de Septiembre de 2003 [greg]: Capítulo 5 − GCC Paso 2: Actualizado a gcc−3.3.1−specs−2.patch. Ncurses: Añadido −−enable−overwrite y su descripción.
- 19 de Septiembre de 2003 [jeremy]: Corregidos los comandos de bash para usar correctamente la opción −h.
- 19 de Septiembre de 2003 [jwrober]: Varias actualizaciones en la página de agradecimientos.
- 18 de Septiembre de 2003 [jeremy]: Capítulo 5 − GCC Paso 2 − Añadidos algunos comentarios sobre los 3 paquetes a desempaquetar.
- 17 de Septiembre de 2003 [greg]: Capítulo 6 − GCC−2.95.3: Añadidas unas notas sobre el razonamiento.
- 17 de Septiembre de 2003 [jwrober]: Actualizada la página de agradecimientos para que coincida con la del sitio web.
- 17 de Septiembre de 2003 [jeremy]: Actualizado File a 4.04
- 17 de Septiembre de 2003 [jeremy]: Capítulo 6 − Cambiadas 2 de las apariciones de exec bash −−login para incluir la directiva +h.
- 17 de Septiembre de 2003 [greg]: Capítulos 5 y 6 − Bloquear Glibc y Reajustar las herramientas: Hacer "make −C ld install" en lugar de "make −C ld install−data−local" para instalar un nuevo enlazador completo en vez de sólo los guiones de ld.
- 17 de Septiembre de 2003 [alex]: Normalizada la escritura de 'Tcl' y 'DejaGnu', siguiendo su propia documentación.
- 17 de Septiembre de 2003 [alex]: Correcta alfabetización de las dependencias.
- 16 de Septiembre de 2003 [alex]: Actualizadas finalmente las dependencias al nuevo Coreutils.
- 16 de Septiembre de 2003 [greg]: Capítulos 5 y 6 − Bloquear Glibc y reajustar las dependencias: Añadidas comprobaciones de seguridad.
- 16 de Septiembre de 2003 [greg]: Capítulos 5 y 6 − Binutils, GCC, y Glibc: Añadidas notas sobre los bancos de pruebas.
- 15 de Septiembre de 2003 [alex]: Corregidos varios errores ortográficos y algunas inconsistencias.
- 14 de Septiembre de 2003 [greg]: Capítulo 6 − Comando chroot revisado: Eliminado +h pues ya no es necesario.
- 14 de Septiembre de 2003 [greg]: Capítulo 6 − Creación de los enlaces simbólicos esenciales: Añadido el enlace simbólico /usr/lib/libgcc_s.so.1 para permitir la ejecución de gcc abi_check. Igualmente era necesario para el futuro NPTL.
- 13 de Septiembre de 2003 [jwrober]: Añadido texto del PLFS−hint a la página de creación de passwd y group en el Capítulo 6: error 596.

- 13 de Septiembre de 2003 [jwrober]: Actualizada la página "Cómo van a hacerse las cosas" para incluir más texto del PLFS−hint.
- 13 de Septiembre de 2003 [jwrober]: Mezclado "whoread" y "whonotread" dentro de la nueva página "audience".
- 13 de Septiembre de 2003 [greg]: Capítulo 2 − Añadida una nueva sección sobre los bancos de pruebas.
- 12 de Septiembre de 2003 [jeremy]: Capítulo 5 − Ncurses: Añadida la descripción para la opción −−without−ada de configure.
- 12 de Septiembre de 2003 [jeremy]: Capítulo 5 − Gawk: Añadido el banco de pruebas.
- 12 de Septiembre de 2003 [jeremy]: Capítulo 5 − Grep: Añadida la descripción a las opciones de configure por cortesía de Anderson Lizardo.
- 12 de Septiembre de 2003 [gerard]: Eliminada la creación del directorio /usr/lib/locale. Se crea durante el Capítulo 6 − Glibc, donde es más relevante.
- 11 de Septiembre de 2003 [jwrober]: Capítulo 5− GCC Paso 2: Corregido el texto del parche specs para hacerlo menos preciso, pero en realidad más correcto. Suministrado por Anderson Lizardo.
- 11 de Septiembre de 2003 [jwrober]: Capítulo 5 − Tcl: Correción gramatical en las intrucciones de instalación. Suministrado por Anderson Lizardo.
- 11de Septiembre de 2003 [jwrober]: Capítulo 5 − Bolquear Glibc: Pequeño cambio textual para /lib/ld.so.1. Suministrado por Anderson Lizardo.
- September 11th, 2003 [jeremy]: Añadida la configuración del gestor de arranque en el Capítulo 8, despues de la inclusión de Grub en el libro.
- 11 de Septiembre de 2003 [gerard]: Eliminados Bin86 y LILO, reemplazados por Grub.
- 11 de Septiembre de 2003 [jeremy]: Cambiadas a opcionales las pruebas de los paquetes que no pertenecen a las herramientas principales. Añadida una nota para usar el wiki para las pruebas fallidas.
- 11 de Septiembre de 2003 [jeremy]: Añadido un parche para Bison, extraido del CVS, para corregir los problemas de compilación con pwlib.
- 11 de Septiembre de 2003 [jeremy]: Añadido el parche de Greg para GCC que suprime la instalación de libbiberty y cambiado Binutils para permitir que su libbiberty permanezca.
- 11 de Septiembre de 2003 [jeremy]: Añadidas unas etiquetas de advertencia en el recordatorio de que no hay que borrar los directorios de las fuentes y de construcción de binutils en el capítulo 5.
- 11 de Septiembre de 2003 [jeremy]: Añadido el nuevo parche perl−libc−3 patch de Anderson Lizardo.
- 9 de Septiembre de 2003 [jwrober]: Corregido el enlace de descarga de findutils en la página de paquetes. Corregido el error 578.
- 9 de Septiembre de 2003 [jeremy]: Capítulo 6 − GCC 2.95.3: Eliminada la compilación de C++, añadido el parche return−type de Zack.
- 9 de Septiembre de 2003 [jeremy]: Capítulo 6 − Coreutils: Añadido el parche coreutils−5.0−hostname−2.patch, que suprime la compilación del binario hostname y también su comprobación.
- 9 de Septiembre de 2003 [jeremy]: Añadidas algunas notas sobre los tests fallidos de Glibc y Dejagnu.
- 9 de Septiembre de 2003 [jeremy]: Glibc − Añadidos los comandos en los Capítulos 5 y 6 para incluir las locales mínimas necesarias para las comprobaciones.
- 9 de Septiembre de 2003 [jeremy]: Capítulo 6 − Eliminado el movimiento de CFLAGS para Zlib a favor de una nota para añadir −fPIC.
- 8 de Septiembre de 2003 [matt]: Capítulo 5 − Corregido el comando rm para que borre la documentación innecesaria de /tools/share.
- 6 de Septiembre de 2003 [matt]: Capítulo 6 − Eliminada una referencia al directorio "static" en la introducción.
- 6 de Septiembre de 2003 [jeremy]: Capítulo 4 − Actualizada la localización de descarga de algunos paquetes.

- 5 de Septiembre de 2003 [jeremy]: Capítulo 5 − GCC Paso 2: Corregida la explicación del error de make check.
- 5 de Septiembre de 2003 [jeremy]: Capítulo 6 − Makedev: Cambiada la creación de dispositivos por defecto a generic−nopty, debido a que ahora utilizamos devpts por defecto.
- 5 de Septiembre de 2003 [jeremy]: Capítulo 6 − GCC: Corregida la frase para reflejar la supresión del enlace simbólico /usr/lib/cpp.
- 5 de Septiembre de 2003 [jeremy]: Corregido el parche libc de perl a −2. Cambiada la antigua estructura /stage1 por /tools.
- 5 de Septiembre de 2003 [matt]: Capítulo 6 − Actualizado el parche specs de gcc y actualizado a man−1.5m2.
- 4 de Septiembre de 2003 [jeremy]: Capítulo 6 − Creación de directorios: Eliminada la creación de /usr/tmp − Corregido el error 176.
- 4 de Septiembre de 2003 [jeremy]: Capíyulo 6 − Montar Proc: Añadido aquí el montaje del sistema de ficheros devpts en el chroot. Corregido el error 533.
- 4 de Septiembre de 2003 [jeremy]: Capítulo 6 − Montar Proc: Añadido un aviso al final relativo a la comprobación de que proc esté montado si paras y reinicias el proceso del lfs.
- 4 de Septiembre de 2003 [jeremy]: Capítulo 6 − Gzip: Modificado el texto para explicar mejor la razón que hay detrás del comando sed utilizado en la instalación de gzip. Corregido el error 551.
- 4 de Septiembre de 2003 [jeremy]: Capítulo 4 − Descarga de parches: Añadida una nota sobre el proyecto de parches de Tushar, y un enlace a la página de parches.
- 3 de Septiembre de 2003 [matt]: Corregido el problema de que util−linux no utilice las librerías y cabeceras instaladas en /stage1.
- 3 de Septiembre de 2003 [matt]: Eliminada la instrucción "rm /bin/pwd" en la instalación de las cabeceras del núcleo en el capítulo 6 pues el enlace todavía es necesario para instalar glibc.
- 2 de Septiembre de 2003 [alex]: Ajustados todos los SBU a los valores suministrados por Jeremy.
- 2 de Septiembre de 2003 [alex]: Finalmente renombramos /stage1 a /tools.
- 2 de Septiembre de 2003 [alex]: Unificados varios de los ficheros principales de la estructura del libro.
- 2 de Septiembre de 2003 [alex]: Listado alfabético de las descargas. Añadida una nota en las instrucciones de TCL.
- 2 de Septiembre de 2003 [alex]: Reescritas las secciones Organización, $LFS y SBU.
- 1 de Septiembre de 2003 [jeremy] − Capítulo 6 − Groff − Añadida una nota sobre la elección de A4 o letter para la variable PAGE.
- 1 de Septiembre de 2003 [jeremy] − Añadido en shadow el parche newgrp de Greg Schafer
- 31 de Agosto de 2003 [jeremy] − Capítulo 6 − Inetutils − añadidas las opciones −−disable−whois y −−disable−servers
- 31 de Agosto de 2003 [jeremy] − Añadidas las nuevas instrucciones de Greg para GCC 3.3.1 relativas al proceso fixincludes. Añadidas también mas explicaciones en las páginas "Bloquear" Glibc y GCC paso 2 sobre el proceso fixincludes.
- 31 de Agosto de 2003 [jeremy] − Añadido el usuario nobody a los ficheros passwd y group, para que coreutils pueda superar sus pruebas.
- 31 de Agosto de 2003 [alex]: Reescritos algunos párrafos, añadidas marcas olvidadas y depurado el historial de modificaciones.
- 31 de Agosto de 2003 [alex]: Encerradas entre paréntesis las frases "Última versión comprobada...". Varios pequeños retoques más.
- 30 de Agosto de 2003 [jeremy] − Actualizado el parche fix−includes para GCC 3.3.1
- 29 de Agosto de 2003 [jeremy] − Glibc − Actualizadas las instrucciones para el parche sscanf.
- 29 de Agosto de 2003 [jeremy] − Actualizado GCC a la versión 3.3.1, incluidas correciones basadas en la mini receta de Zack para GCC 3.3 y parches procedentes de sus documentos.
- 29 de Agosto de 2003 [alex]: Eliminados los ficheros obsoletos de Netkit−base, Fileutils, Sh−utils, y Textutils.

- 29 de Agosto de 2003 [alex]: Añadidas algunas marcas que faltaban y cambiados varios /static por /stage1.
- 29 de Agosto de 2003 [alex]: Capítulo 06 − Añadidas todas las líneas de texto faltantes en los make checks y reescritas otras líneas.
- 28 de Agosto de 2003 [matt] − Actualizado a linux−2.4.22, man−pages−1.60, expect−5.39.0, findutils−4.1.20 y tcl−8.4.4
- 28 de Agosto de 2003 [jeremy] − Nuevo parche bash−2.05b−2.patch que incluye los 7 parches de ftp.gnu.org
- 28 de Agosto de 2003 [alex]: Capítulo 06 − Reajustando las herramientas: Añadida una barra olvidada.
- 28 de Agosto de 2003 [alex]: Corregidos varios errores y añadidas algunas marcas olvidadas.
- 28 de Agosto de 2003 [alex]: Capítulo 06 − Binutils y GCC: Integrado el texto de la receta pure−lfs.
- 27 de Agosto de 2003 [jeremy] − Capítulo 06 − Inetutils: Añadido −−sysconfdir=/etc −−localstatedir=/var y movido el binario ping de /usr/bin a /bin.
- 27 de Agosto de 2003 [alex]: Capítulo 06 − Glibc: Integrado el texto de la receta pure−lfs.
- 26 de Agosto de 2003 [jeremy] − Capítulo 07 − Creación de /etc/hosts: Cambiado www.mydomain.org por <valor de HOSTNAME>.mydomain.org
- 26 de Agosto de 2003 [alex]: Capítulos 06 y 08 − Movida la instalación de las páginas de manual del núcleo del capítulo 6 al 8.
- 26 de Agosto de 2003 [jeremy] − Capítulo 04 − Montar la partición LFS: Añadido texto relativo al montaje con permisos demasiado restrictivos.
- 26 de Agosto de 2003 [jeremy] − Capítulo 06 − Creación de directorios: Añadida la creación del directorio /dev/shm.
- 26 de Agosto de 2003 [jeremy] − Capítulo 08 − Creación de fstab: Añadido el montaje del sistema de ficheros tmpfs en /dev/shm.
- 26 de Agosto de 2003 [jeremy] − Capítulo 08 − Instalación del núcleo: Añadido un recordatorio para compilar en el núcleo el soporte para tmpfs.
- 25 de Agosto de 2003 [alex]: Capítulo 06 − Reescrito el texto de la instalación de Shadow y Util−Linux mientras se corregían algunos errores.
- 25 de Agosto de 2003 [alex]: Capítulos 05 y 06 − Hecho que "Bloqueando" y "Reajustando" tengan un aspecto similar.
- 24 de Agosto de 2003 [alex]: Capítulo 04 − Mezclados los múltiples ficheros pequeños en un solo fichero. Puestos los paquetes y los parches en páginas separadas.
- 17 de Agosto de 2003 [alex]: Capítulo 05 − De Bash a Perl: Añadido el texto entre comandos. Añadida una sección sobre la eliminación de símbolos innecesarios para reducir el tamaño de las herramientas.
- 16 de Agosto de 2003 [alex]: Capítulo 05 − De Make a Texinfo: Añadido el texto entre comandos.
- 11 de Agosto de 2003 [alex]: Capítulo 05 − Desde Binutils Fase 1 a Findutils: varios pequeños ajustes en el texto. Se omiten los contenidos y dependencias en las segundas fases de Binutils y GCC.
- 11 de Agosto de 2003 [alex]: Capítulo 04 − Separados los paquetes core, g++, y testsuite de GCC.
- 11 de Agosto de 2003 [alex]: Capítulo 04 − Eliminada la mención del guión para wget.
- 9 de Agosto de 2003 [alex]: Capítulo 05 − Binutils Fase 2 y GCC Fase 2: integrada una parte del texto de la receta pure−lfs.
- 8 de Agosto de 2003 [alex]: Capítulo 05 − Tcl, Expect, y DejaGNU: añadido algo de texto.
- 6 de Agosto de 2003 [gerard]: Aplicado el parche de Alex Groenewoud que añade el Apéndice B, proporcionando una lista de todos los programas y librerías instalados, más referencias a las páginas de instalación.
- 30 de Julio de 2003 [gerard]: Capítulo 06 − Vim: Cambiado el modo en que se define de forma global la localización de `vimrc` y `gvimrc`.
- 30 de Julio de 2003 [gerard]: Capítulo 05 − Binutils Fase 2: eliminado el parche de la librerías; no hace falta con la actualización a binutils−2.14.

- 30 de Julio de 2003 [gerard]: Capítulo 05 − Binutils Fase 1: Añadido **`make configure-host`**.
- 30 de Julio de 2003 [gerard]: Actualizado a binutils−2.14, linux−2.4.21, expect−5.38.4, gawk−3.1.3, texinfo−4.6, util−linux−2.12, man−pages−1.58, lfs−utils−0.3, vim−6.2, gettext−0.12.1, automake−1.7.6, file−4.03, e2fsprogs−1.34, procps−3.1.11, psmisc−21.3
- 3 de Junio de 2003 [gerard]: Capítulo 06 − Gawk: Eliminado el borrado de `/bin/awk`. Ya no se crea este enlace simbólico.
- 21 de Mayo de 2003 [gerard]: Capítulo 06 − GCC−2.95.3: Añadido /opt/gcc−2.95.3/lib a /etc/ld.so.conf para que las librerías puedan ser usadas.
- 21 de Mayo de 2003 [gerard]: Capítulo 05 − Gzip: Simplificados los comandos.
- 21 de Mayo de 2003 [gerard]: Capítulo 05 − Bzip2: Simplificados los comandos.
- 21 de Mayo de 2003 [gerard]: Capítulo 06 − Shadow: Añadido el comando **`grpconv`** para complementar la activación de las contraseñas ocultas.
- 21 de Mayo de 2003 [winkie]: Capítulo 06 − Creación de Ficheros: Todos esos comandos **`ln`** se pueden hacer en unos pocos comandos ln largos.
- 21 de Mayo de 2003 [winkie]: Capítulo 05 − Instalación de Glibc: Crea un fichero ld.so.conf antes de construir Glibc, para prevenir un error inofensivo.
- 21 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Glibc: No ejecutamos más 'exec /stage1/bin/bash', ya que no tiene sentido al usar PLFS.
- 21 de Mayo de 2003 [winkie]: Capítulos 05 y 06 − Instalación de Coreutils: Sólo ejecutamos las comprobaciones no root en el Capítulo 05, luego ejecutamos todo en el Capítulo 06.
- 21 de Mayo de 2003 [winkie]: Capítulo 05 − Instalación de Expect: Sólo pasamos −−prefix=/stage1. Las demás opciones no son necesarias.
- 16 de Mayo de 2003 [gerard]: Capítulo 06: Net−tools: Cambiado **`make install`** por **`make update`**.
- 15 de Mayo de 2003 [timothy]: Capítulo 05: Instalación de Patch: Añadido **`CPPFLAGS=-D_GNU_SOURCE`** antes de **`./configure`** para corregir la construcción en plataformas PPC.
- 13 de Mayo de 2003 [gerard]: Capítulo 06: Después de **`exec /ruta/hacia/bash --login`**, ejecutamos **`set +h`** para mantener desactivada la opción de hashing. Corrige el error #531.
- 13 de Mayo de 2003 [gerard]: Capítulo 06 − Red Básica: Cambiadas las comillas simples por comillas dobles en el comando echo. Sin ellas, $(hostname) no se expandirá, haciendo que este comando no cumpla con su objetivo − que funcione la comprobación de perl del nombre de la máquina.
- 13 de Mayo de 2003 [winkie]: Eliminadas todas las apariciones de &&. Actualizada la sintaxis de los errores (bugs). Añadido "make check/test" donde es necesario durante el Capítulo 6.
- 13 de Mayo de 2003 [winkie]: Aplicado el parche "Cambio de dueño" para clarificar el texto. Corrige el error #511.
- 13 de Mayo de 2003 [winkie]: Aplicado el parche "Configuración de los componentes del sistema" para clarificar el texto. Corrige el error #510.
- 13 de Mayo de 2003 [gerard]: Capítulo 06: Eliminados TCL, Expect y DejaGNU. Ningún paquete los usa después de GCC en el Capítulo 6. Las versiones instaladas en /stage1/bin cumplen bien con su trabajo.
- 13 de Mayo de 2003 [winkie]: Capítulo 06 − Instalando Shadow: Ejecutamos 'touch /usr/bin/passwd' antes de instalar. No hacerlo provoca que Shadow piense que passwd estará en /bin.
- 13 de Mayo de 2003 [winkie]: Capítulo 06 − Instalando Procps: Eliminar el enlace /lib/libproc.so. Ningún paquete, además del propio Procps, usa o debiera usar esta librería.
- 13 de Mayo de 2003 [winkie]: Capítulo 06 − Instalando Net−tools: Ejecutamos "make config" antes de "make". Corrige los errores #462 y #497.
- 13 de Mayo de 2003 [gerard]: Capítulo 06 − Ncurses: Añadido el parche vsscanf.
- 12 de Mayo de 2003 [gerard]: Capítulo 05 − Gzip: Eliminado **`make check`**. No hace nada.

- 12 de Mayo de 2003 [winkie]: Capítulo 05 − Instalación de Texinfo: Ya no instalamos los datos de texmf. Ningún paquete los usa.
- 12 de Mayo de 2003 [winkie]: Capítulos 05 y 06 − Instalación de Ncurses: En el Capítulo 6, la creación de los enlaces se actualizó para incluir libcurses.*, y libncurses++.a tiene sus permisos cambiados a 644. En el capítulo 5 no se necesitan las libcurses.*, así que las eliminamos.
- 12 de Mayo de 2003 [gerard]: Capítulo 06 − Red Básica: Añadido $(hostname) a /etc/hosts, de no hacerlo, falla la comprobación de Perl por el nombre de la máquina.
- 12 de Mayo de 2003 [gerard]: Capítulo 06 − Instalación de GCC: No eliminamos /usr/include/libiberty.h ya que nunca se instala.
- 12 de Mayo de 2003 [winkie]: Actualizado a findutils−4.1.7, gzip−1.3.5 y tar−1.13.25.
- 12 de Mayo de 2003 [winkie]: Capítulo 05 − Instalación de Perl: Añadidos comandos extra para construir ciertos módulos dentro de Perl para evitar problemas al hacer el "make check" de Coreutils. Corrige en parte el error #528.
- 12 de Mayo de 2003 [winkie]: Capítulo 05 − Instalación de Gzip: Nada en el Capítulo 6 necesita el comando uncompress, por tanto no deberíamos crearlo.
- 12 de Mayo de 2003 [winkie]: Capítulo 05 − Instalación de Bzip2: Ejecutar "make" implica "make check", no hay razón para ejecutarlo manualmente.
- 12 de Mayo de 2003 [winkie]: Capítulo 05 − Instalación de Lfs−Utils: Eliminado. El único paquete que comprueba la existencia de mktemp antes de instalarse es GCC, y solamente para gccbug.
- 11 de Mayo de 2003 [gerard]: Capítulo 06 − GCC−2.95.3: Añadida la opción −−enable−threads=posix para completar la adición del compilador de C++.
- 11 de Mayo de 2003 [gerard]: Capítulo 06 − GCC−2.95.3: Añadida la opción −−enable−languages=c,c++ para corregir un error de esa versión de gcc con respecto a −Wreturn−type. Corrige el error #525
- 11 de Mayo de 2003 [gerard]: Capítulo 05 − Bash: Eliminada la opción de configure −−without−bash−malloc.
- 11 de Mayo de 2003 [gerard]: Actualizado a gcc−3.2.3−specs−4.patch.
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Red Básica: Añadida una sección. Creamos un fichero /etc/hosts básico. Creamos /etc/services y /etc/protocols desde IANA. Corrige los errores #359 y #515.
- 11 de Mayo de 2003 [winkie]: Actualizado a lfs−utils−0.2.2. Esto añade dos ficheros necesarios para una correcta configuración de la red.
- 11 de Mayo de 2003 [winkie]: Eliminado Netkit−base 0.17. Añadido Inetutils 1.4.2. Corrige el error #490.
- 11 de Mayo de 2003 [winkie]: Añadido lfs−utils−0.2.1. Corrige el error #493.
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Ncurses: Corregidos los enlaces simbólicos. No más cosas extrañas.
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Procps: Eliminada XSCPT="" y su correspondiente párrafo. Esta variable ya no es necesaria.
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Ncurses: Pasamos −−without−debug al guión configure. Parece que lo hemos olvidado en algún momento.
- 11 de Mayo de 2003 [timothy]: Capítulos 5 y 6 − Instalación de Bzip2, Instalación de Zlib: Modificados los comandos de instalación, de acuerdo con el error #524.
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Glibc: También instalamos las páginas de manual de linuxthreads. Esto quedó olvidado en algún momento.
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Grep: Añadida la opción −−with−included−regex para evitar que Grep use las expresiones regulares de Glibc, que tienen algunos errores .
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Coreutils: Corregida en parte la funcionalidad del comando uname, mediante un parche.

- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Net−tools: Volvemos al viejo "make install" en vez de "make update". Ahora funciona correctamente.
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de GCC: Después de la instalación, eliminamos /usr/include/libiberty.h. No se usa fuera del árbol de compilación de GCC.
- 11 de Mayo de 2003 [winkie]: Actualizado a Bash 2.05b y añadido un parche.
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Zlib: Aplicado un parche que corrige un desbordamiento de memoria en gzprintf().
- 11 de Mayo de 2003 [winkie]: Capítulo 06 − Configuración de los componentes del sistema: Movida la creación de btmp, wtmp, lastlog y utmp justo después de Shadow, para que sean detectados en las ubicaciones correctas.
- 10 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Automake: Ejecutamos "make" antes de instalar. Las nuevas versiones de Automake lo necesitan.
- 10 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Vim: Eliminado el parche. No se necesita con GCC 3.2.1 y posteriores.
- 10 de Mayo de 2003 [winkie]: Capítulo 06 − Creando el fichero mtab: Eliminado. Montar /proc crea /etc/mtab por nosotros.
- 10 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Make: Eliminada la modificación de /usr/bin/make. Ya no se instala con propietario y/o permisos extraños.
- 10 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de Glibc: Ahora, /etc/localtime es un fichero en vez de un enlace. El método del enlace no sirve en sistemas en los que /usr reside en una partición separada.
- 10 de Mayo de 2003 [winkie]: Capítulo 06 − Instalación de E2fsprogs: Eliminados los comandos install−info de e2fsprogs. El objetivo "make install" se encarga por nosotros.
- 10 de Mayo de 2003 [gerard]: Eliminadas todas las variables CFLAGS y LDFLAGS donde no sean esenciales (es decir, excepto para compilar binutils y gcc estáticamente y al compilar zlib con −fPIC).
- 10 de Mayo de 2003 [gerard]: Capítulo 05 − Binutils (fase 1, fase 2, bloquear glibc y ajustar las herramientas): Cambiada la opción tooldir a /stage1 (del mismo modo, tooldir=/usr en el Capítulo 6).
- 10 de Mayo de 2003 [gerard]: Capítulo 05 − Cabeceras del Núcleo: Eliminado el uso de **`cp -H`** porque algunas versiones de cp no conocen la opción **`-H`**.
- 10 de Mayo de 10th, 2003 [gerard]: Nuevo gcc−3.2.3−specs−3.patch.
- 10 de Mayo de 2003 [gerard]: Capítulo 06 − Ajustando las herramientas: Hecho más independiente de la arquitectura.
- 10 de Mayo de 2003 [gerard]: Capítulo 05 − Bloqueando Glibc: Hecho más independiente de la arquitectura.
- 7 de Mayo de 2003 [gerard]: Eliminados los parches GCC_No_Debug. Ya no asumimos que se descargaron los paquetes gcc−core y gcc−g++, así que se añade la opción −−enable−languages adecuada.
- 7 de Mayo de 2003 [gerard]: Eliminado Capítulo 6 − Glibc−Fase2. Ya no se necesita con la integración de Pure−LFS.
- 7 de Mayo de 2003 [gerard]: Vuelta a la versión 2.5.4a de flex. Las versiones más nuevas siguen sin funcionar correctamente.
- 5 de Mayo de 2003 [gerard]: Eliminada la instalación de zlib en el Capítulo 5 (se incluyó por error).
- 5 de Mayo de 2003 [gerard]: Corregidos varios errores introducidos durante la integración con Pure−LFS.
- 2 de Mayo de 2003 [gerard]: Actualizado a: automake−1.7.4, e2fsprogs−1.33, file−4.02, flex−2.5.31, gawk−3.1.2, gcc−3.2.3, glibc−2.3.2, grep−2.5.1, groff−1.19, less−381, libtool−1.5, man−1.5l, man−pages−1.56, modutils−2.4.25, procps−3.1.8, sed−4.0.7, sysvinit−2.85, texinfo−4.5 y util−linux−2.11z
- 2 de Mayo de 2003 [gerard]: Eliminados fileutils−4.1, sh−utils−2.0 y textutils−2.1 (todos reemplazados por coreutils−5.0).

- 2 de Mayo de 2003 [gerard]: Añadidos binutils−2.13.2−libc.patch, coreutils−5.0, dejagnu−1.4.3, expect−5.38, gawk−3.1.2, gcc−2.95.3 y tcl−8.4.2
- 2 de Mayo de 2003 [gerard] − Integrado el nuevo método de instalación de la receta Pure LFS (LFS Puro) escrita por Greg Schafer y Ryan Oliver.

Liberada la versión 4.1 el 28 de Abril de 2003

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 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*. 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*.

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
- Dattaelite.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 (ver arriba).

Pero si necesitas contactar personalmente con Gerard Beekmans, manda un mensaje a gerard@linuxfromscratch.org

Y si necesitas contactar directamente con el coordinador del Proyecto LFS−ES, envía tu mensaje a macana@lfs−es.org

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> −− 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.
- 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.
- 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.
- Scot Mc Pherson <scot@linuxfromscratch.org> −− Mantenedor de la pasarela NNTP de LFS.
- Ryan Oliver <ryan@linuxfromscratch.org> −− Líder del equipo de pruebas 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, 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 y mantenedor de Bugzilla.
- 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.

Administradores de la red de réplicas

- Jason Andrade <jason@dstc.edu.au> −− au.linuxfromscratch.org.
- William Astle <lost@l−w.net> −− ca.linuxfromscratch.org.
- Baque <baque@cict.fr> −− lfs.cict.fr.
- Stephan Brendel <stevie@stevie20.de> −− lfs.netservice−neuss.de.
- Ian Chilton <ian@ichilton.co.uk> −− us.linuxfromscratch.org, linuxfromscratch.co.uk.
- Fredrik Danerklint <fredan−lfs@fredan.org> −− se.linuxfromscratch.org
- David D.W. Downey <pgpkeys@aeternamtech.com> −− lfs.learnbyexample.com
- Eduardo B. Fonseca <ebf@aedsolucoes.com.br> −− br.linuxfromscratch.org
- Hagen Herrschaft <hrx@hrxnet.de> −− de.linuxfromscratch.org
- Tim Jackson <tim@idge.net> −− linuxfromscratch.idge.net
- Barna Koczka <barna@siker.hu> −− hu.linuxfromscratch.org
- Roel Neefs −− linuxfromscratch.rave.org
- Simon Nicoll <sime@dot−sime.com> −− uk.linuxfromscratch.org
- Ervin S. Odisho <ervin@activalink.net> −− lfs.activalink.net
- Guido Passet <guido@primerelay.net> −− nl.linuxfromscratch.org
- Mikhail Pastukhov <miha@xuy.biz> −− lfs.130th.net
- Jeremy Polen <jpolen@rackspace.com> −− us2.linuxfromscratch.org
- UK Mirror Service −− linuxfromscratch.mirror.co.uk

- Thomas Skyt <thomas@sofagang.dk> −− dk.linuxfromscratch.org
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> −− at.linuxfromscratch.org
- Dag Stenstad <dag@stenstad.net> por proporcionar no.linuxfromscratch.org y Ian Chilton por mantenerlo.
- Adimistradores de sistemas parisienses<archive@doc.cs.univ−paris8.fr> −− www2.fr.linuxfromscratch.org.
- Jesse Tie−Ten−Quee <highos@linuxfromscratch.org> por proporcionar y mantener el servidor linuxfromscratch.org.
- Alexander Velin <velin@zadnik.org> −− bg.linuxfromscratch.org.
- Martin Voss <Martin.Voss@ada.de> −− lfs.linux−matrix.net.
- Pui Yong <pyng@spam.averse.net> −− sg.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 responder incontables preguntas en el IRC y tener grandes dosis de paciencia.

Capítulo 2. Información importante

Sobre $LFS

Por favor, lee los siguientes párrafos con atención. En este libro la variable LFS se usará frecuentemente. $LFS deberá sustituirse en todo momento por el directorio en el que está montada la partición que contiene el sistema LFS. Cómo crear y dónde montar la partición se explicará con todo detalle en el Capítulo 3. Por el momento asumiremos que la partición LFS está montada en `/mnt/lfs`.

Cuando se te indique que ejecutes un comando como **`./configure --prefix=$LFS/tools`**, en realidad debes ejecutar **`./configure --prefix=/mnt/lfs/toools`**.

Es importante hacer esto donde quiera que aparezca, ya sea en comandos introducidos en un intérprete de comandos, o al crear o editar un archivo.

Una posible solución es establecer la variable de entorno LFS. De este modo $LFS puede introducirse literalmente, en lugar de substituirlo por /mnt/lfs. Esto se consigue ejecutando:

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

Ahora, cuando las instrucciones sean ejecutar un comando como **`./configure --prefix=$LFS/tools`** puedes introducir eso literalmente. Tu intérprete de comandos substituirá "$LFS" por "/mnt/lfs" al procesar la línea de comando (es decir, cuando pulses Enter después de haber tecleado el comando).

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 mas 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)* (abreviado, *SBU*).

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

Por ejemplo, el tiempo que tarda en construirse la versión estática de GCC es 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 el GCC estático. 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−3.3.1 que se sabe que es aproximadamente un 30% mas lento.

Ten en cuenta tambien 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.

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 (la librería C) −− 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.

A medida que avances a través del libro y encuentres los comandos para ejecutar los diferentes bancos de pruebas, te indicaremos la importancia relativa de dicho banco de pruebas para que puedas decidir por tí mismo si vas a ejecutarlo o no.

> **Nota:** 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, trataremos este tema con mayor detalle.

Cómo buscar ayuda

Si tienes algún problema usando este libro, y tu problema no aparece en las FAQ (en castellano en http://www.escomposlinux.org/lfs−es/faq, y en inglés en http://www.linuxfromscratch.org/faq), encontrarás que la mayoría de la gente en el Internet Relay Chat (IRC) y en las listas de correo estará dispuesta a ayudarte (puedes encontrar una introducción a las listas de correo de LFS en Capítulo 1 − Listas de correo). Para facilitarnos la tarea de identificar 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, debes incluir lo siguiente en tu petición:

- La versión del libro que estás usando (que es 5.0),
- La distribución anfitrión (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, la razón de ser de LFS es la posibilidad de 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 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.

II. Parte II − Preparativos para la construcción

Índice

Capítulo 3. Preparación de 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 del LFS, pero si andas escaso de espacio en el 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 1,2 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 sobre 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) (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-5.0/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
```

Después 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 tienes decidido usar mútiples 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 4. 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 (2.57) − 792 KB:
http://freshmeat.net/projects/autoconf/

Automake (1.7.6) − 545 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 (5.0) − 3,860 KB:
http://freshmeat.net/projects/coreutils/

DejaGnu (1.4.3) − 1,775 KB:
http://freshmeat.net/projects/dejagnu/

Diffutils (2.8.1) − 762 KB:
http://freshmeat.net/projects/diffutils/

E2fsprogs (1.34) − 3,003 KB:
http://freshmeat.net/projects/e2fsprogs/

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

Expect (5.39.0) − 508 KB:
http://freshmeat.net/projects/expect/

File (4.04) − 338 KB: (*) Ver Nota al Pié
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 (3.3.1) − 10,969 KB:
http://freshmeat.net/projects/gcc/

GCC−g++ (3.3.1) − 2,017 KB:
http://freshmeat.net/projects/gcc/

GCC−testsuite (3.3.1) − 1,033 KB:
http://freshmeat.net/projects/gcc/

Gettext (0.12.1) − 5,593 KB:
http://freshmeat.net/projects/gettext/

Glibc (2.3.2) − 13,064 KB:
http://freshmeat.net/projects/glibc/

Glibc−linuxthreads (2.3.2) − 211 KB:
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 (0.93) − 870 KB:
ftp://alpha.gnu.org/pub/gnu/grub/

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

Inetutils (1.4.2) − 1,019 KB:
http://freshmeat.net/projects/inetutils/

Kbd (1.08) − 801 KB:
http://freshmeat.net/projects/kbd/

Less (381) − 259 KB:
http://freshmeat.net/projects/less/

LFS−Bootscripts (1.12) − 25 KB:
http://downloads.linuxfromscratch.org/lfs−bootscripts−1.12.tar.bz2

Lfs−Utils (0.3) − 221 KB:
http://www.linuxfromscratch.org/~winkie/downloads/lfs−utils/

Libtool (1.5) − 2,751 KB:
http://freshmeat.net/projects/libtool/

Linux (2.4.22) − 28,837 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

MAKEDEV (1.7) − 8 KB:
http://downloads.linuxfromscratch.org/MAKEDEV−1.7.bz2

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

Man−pages (1.60) − 627 KB:
http://freshmeat.net/projects/man−pages/

Modutils (2.4.25) − 215 KB:
http://freshmeat.net/projects/modutils/

Ncurses (5.3) − 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 (5.8.0) − 10,765 KB:
http://freshmeat.net/projects/perl/

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

Procps (3.1.11) − 242 KB:
http://freshmeat.net/projects/procps/

Psmisc (21.3) − 259 KB:
http://freshmeat.net/projects/psmisc/

Sed (4.0.7) − 678 KB:
http://freshmeat.net/projects/sed/

Shadow (4.0.3) − 760 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 (1.13.25) − 1,281 KB:
ftp://alpha.gnu.org/gnu/tar/

Tcl (8.4.4) − 3,292 KB:
http://freshmeat.net/projects/tcltk/

Texinfo (4.6) − 1,317 KB:
http://freshmeat.net/projects/texinfo/

Util−linux (2.12) − 1,814 KB:
http://freshmeat.net/projects/util−linux/

Vim (6.2) − 3,193 KB:
http://freshmeat.net/projects/vim/

Zlib (1.1.4) − 144 KB:
http://freshmeat.net/projects/zlib/

Tamaño total de estos paquetes: 134 MB

> **Nota:** File (4.04) puede que no esté disponible cuando leas esto. Es sabido que la localización principal de descarga elimina las antiguas versiones cuando se libera una nueva. Por favor, consulta la sección correspondiente en Apéndice A para una localización de descarga alternativa.

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/5.0/bash−2.05b−2.patch

Parche Attribute para Bison − 2 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/bison−1.875−attribute.patch

Parche Hostname para Coreutils − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−hostname−2.patch

Parche Uname para Coreutils − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−uname.patch

Parche Mkstemp para Ed − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/ed−0.2−mkstemp.patch

Parche Spawn para Expect − 6 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/expect−5.39.0−spawn.patch

Parche Libexecdir para Gawk − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gawk−3.1.3−libexecdir.patch

Parche No−Fixincludes para GCC − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−no_fixincludes−2.patch

Parche Specs para GCC − 10 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−specs−2.patch

Parche que suprime Libiberty para GCC − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−suppress−libiberty.patch

Parche para GCC−2 − 16 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−2.patch

Parche No−Fixincludes para GCC−2 − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−no−fixinc.patch

Parche Return−Type para GCC−2 − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−returntype−fix.patch

Parche Sscanf para Glibc − 2 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/glibc−2.3.2−sscanf−1.patch

Parche Gcc33 para Grub − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/grub−0.93−gcc33−1.patch

Parche More−Programs para Kbd − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/kbd−1.08−more−programs.patch

Parche 80−Columns para Man − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−80cols.patch

Parche Manpath para Man − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−manpath.patch

Parche Pager para Man − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−pager.patch

Parche Etip para Ncurses − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−etip−2.patch

Parche Vsscanf para Ncurses − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−vsscanf.patch

Parche Mii−Tool−Gcc33 para Net−tools − 2 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/net−tools−1.60−miitool−gcc33−1.patch

Parche Libc para Perl − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/perl−5.8.0−libc−3.patch

Parche Locale para Procps − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/procps−3.1.11−locale−fix.patch

Parche Newgrp para Shadow − 1 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/shadow−4.0.3−newgrp−fix.patch

Parche Vsnprintf para Zlib − 10 KB:
http://www.linuxfromscratch.org/patches/lfs/5.0/zlib−1.1.4−vsnprintf.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 los parches, que se encuentra en http://www.linuxfromscratch.org/patches, y elegir cualquier parche adicional que desees utilizar.

Capítulo 5. Construir un 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.

La construcción de este sistema minimalista se hará en dos etapas: primero construiremos un conjunto de herramientas independiente del sistema anfitrión (compilador, ensamblador, enlazador y librerías), 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. Puesto que los paquetes compilados aquí son puramente temporales, no queremos que estos ficheros contaminen el futuro sistema LFS.

La clave para aprender qué es lo que hace funcionar un sistema Linux es saber para qué se usa cada paquete y por qué el usuario o el sistema lo necesita. Por esta razón se facilita un breve resumen del contenido de cada paquete antes de las propias instrucciones de instalación. Para ver una descripción corta de cada programa de un paquete, consulta la sección correspondiente en el Apéndice A.

Las instrucciones de construcción asumen que estás usando el intérprete de comandos bash. También se asume que ya has desempaquetado el código fuente del paquete y has ejecutado un **`cd`** para cambiarte al directorio de los fuentes desempaquetadas antes de empezar con los comandos de instalación.

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 este como en el siguiente capítulo, pero a veces sólo es necesario en uno de ellos. Por tanto, no te preocupes cuando parezca que hemos olvidado las instrucciones para uno de los parches descargados.

Durante la instalación de muchos paquetes verás aparecer en pantalla todo tipo de 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.

Excepto si se indica lo contrario, normalmente debes borrar las fuentes y directorios de construcción despues de instalar cada paquete, por motivos de limpieza y para liberar espacio.

Antes de continuar, asegúrate de que la variable de entorno LFS tiene el valor correcto ejecutando lo siguiente:

```
echo $LFS
```

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

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 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. 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, facilitar el máximo valor educacional. En otras palabras, se pueden usar técnicas mas 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" será, por ejemplo: *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:

- 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 cual es el enlazador dinámico a usar.

Se instala primero Binutils debido a que tanto GCC como Glibc realizan varias pruebas de capacidades sobre el ensamblador y el enlazador en sus respectivas fases **`./configure`** para determinar qué características deben activarse o desactivarse. Esto es mas 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 cual 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 los ficheros specs de **gcc** para que apunten 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 los ficheros specs de **gcc** estaremos seguros de que todo binario compilado desde aquí hasta el final del Capítulo 5 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 el segundo paso 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 el segundo paso de Binutils podremos usar la opción *--with-lib-path* de configure para controlar la ruta de búsqueda de librerías de **ld**. A partir de este punto el corazón de las herramientas está autocontenido y autohospedado. El resto de los paquetes del Capítulo 5 se construirán todos contra la nueva Glibc en `/tools` como debe ser.

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

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, cálculo 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 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.

Creación del directorio $LFS/tools

Todos los programas que compilamos en este capítulo se instalarán bajo `$LFS/tools` para mantenerlos separados de los programas compilados en el próximo capítulo. 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.

Si más tarde quieres 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. En lugar del simple "tools" puedes usar algo como "herramientas−para−lfs".

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 /
```

Este enlace 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).

> **Nota:** Estudia con atención el comando anterior. Puede parecer confuso en un primer momento. El comando **`ln`** tiene bastantes variaciones de sintaxis, por lo que asegúrate de comprobar la página de manual de ln antes de informar de lo que puedes pensar que es un error.

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 los siguientes comandos para añadir el nuevo usuario:

```
useradd -s /bin/bash -m lfs
passwd lfs
```

Ahora, 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 nuevo y limpio.

Configuración del entorno

Mientras estás en el sistema como usuario *lfs*, ejecuta los siguientes comandos para establecer un buen entorno de trabajo:

```
cat > ~/.bash_profile << "EOF"
set +h
umask 022
```

```
LFS=/mnt/lfs
LC_ALL=POSIX
PATH=/tools/bin:$PATH
export LFS LC_ALL PATH
unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD
EOF

source ~/.bash_profile
```

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.

Las variables de entorno CC, CXX, CPP, LD_LIBRARY_PATH y LD_PRELOAD tienen el potencial de causar estragos con nuestras herramientas principales del Capítulo 5. Por tanto las desactivamos para evitar que esto pueda pasar.

Ahora, tras cargar con 'source' el perfil recién creado, tenemos todo preparado para comenzar a construir las herramientas temporales que nos ayudarán en los siguientes capítulos.

Instalación de Binutils−2.14 − Fase 1

```
Tiempo estimado de construcción:               1.0 SBU
Estimación del espacio necesario en disco:     194 MB
```

Contenido de Binutils

Binutils es una colección de herramientas para el desarrollo de software que contiene un enlazador, un ensamblador y otras utilidades para trabajar con ficheros de objetos y archivos.

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]

Dependencias de instalación de Binutils

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.

> **Nota:** Aunque Binutils es un paquete importante de las herramientas principales, no vamos a ejecutar su banco de pruebas en esta fase. Primero, porque el entorno de trabajo del banco de pruebas aún no está en su sitio y segundo, porque los programas de esta primera fase pronto serán sobreescritos por aquellos instalados en la segunda fase.

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 construir Binutils en un directorio aparte, 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. Para ello, haz lo siguiente: **`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). 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 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 primero al programa *libtool*, el cual luego le pasa **`"-static"`** al enlazador.

Instala el paquete:

```
make install
```

Ahora prepara al enlazador para "bloquear" Glibc más tarde:

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

Significado de las opciones de make:

- **`-C ld clean`**: Esto le indica al programa make que elimine todos los ficheros compilados que haya en el subdirectorio `ld` únicamente.
- **`-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. Lo que preparas aquí lo utilizarás 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.

Instalación de GCC-3.3.1 - Fase 1

```
Tiempo estimado de construcción:               4.4 SBU
Estimación del espacio necesario en disco:     300 MB
```

Contenido de GCC

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

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

Dependencias de instalación de GCC

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

Instalación de GCC

Por ahora no vamos a necesitar un compilador de C++, así que desempaqueta GCC−core solamente.

> **Nota:** Aunque GCC es un paquete importante de las herramientas principales, no vamos a ejecutar su banco de pruebas en esta fase. Primero, porque el entorno de trabajo del banco de pruebas aún no está en su sitio y segundo, porque los programas de esta primera fase pronto serán sobreescritos por aquellos instalados en la segunda fase.

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 GCC.

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

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

Prepara GCC para su compilación:

```
../gcc-3.3.1/configure --prefix=/tools \
    --with-local-prefix=/tools \
    --disable-nls --enable-shared \
    --enable-languages=c
```

El significado de las opciones de configure es:

- **`--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 queremos intentar 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 más adelante.
- **`--enable-languages=c`**: Esta opción nos asegura que sólo se construya el compilador de C. Es necesaria unicamente en caso de que hayas descargado y desempaquetado el paquete completo de GCC.

Continúa compilando el paquete:

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

El significado de las opciones de make es:

- **`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.

Instala el paquete:

```
make install
```

Como toque final crearemos el enlace `/tools/bin/cc`. Muchos programas y guiones ejecutan **`cc`** en vez de **`gcc`**. Esto es una forma de hacer que los programas sean genéricos y por tanto usables 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 -sf gcc /tools/bin/cc
```

Instalación de las cabeceras de Linux−2.4.22

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     186 MB
```

Contenido de Linux

El núcleo Linux es el corazón de todo sistema Linux. Es lo que hace a Linux funcionar. Cuando se enciende un ordenador y se inicia un sistema Linux, el núcleo es lo primero que se carga. El núcleo inicializa los componentes hardware del sistema: puertos serie, puertos paralelo, tarjetas de sonido, tarjetas de red, controladores IDE, controladores SCSI y mucho más. En pocas palabras, el núcleo hace que el hardware esté disponible para que el software pueda ejecutarse.

Ficheros instalados: el núcleo y las cabeceras del núcleo.

Dependencias de instalación de Linux

Linux depende de: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

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
```

Instala los ficheros de cabecera del núcleo independientes de la plataforma:

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

Hay ciertos ficheros de cabecera del núcleo que hacen uso del fichero `autoconf.h`. Puesto que todavía no hemos configurado el núcleo, necesitamos crear este fichero por nuestra cuenta para evitar fallos de compilación. Crea un fichero `autoconf.h` vacio:

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

Instalación de Glibc-2.3.2

```
Tiempo estimado de construcción:               11.8 SBU
Estimación del espacio necesario en disco:     800 MB
```

Contenido de Glibc

Glibc es la librería C que proporciona las llamadas al sistema y las funciones básicas, tales como open, malloc, printf, etc. La librería C es utilizada por todos los programas enlazados dinámicamente.

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]

Dependencias de instalación de Glibc

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

Instalación de Glibc

Antes de instalar Glibc, debes entrar al directorio `glibc-2.3.2` con el comando **`cd`** y desempaquetar Glibc-linuxthreads dentro de este directorio, no en el directorio donde normalmente desempaquetas las

fuentes.

> **Nota:** En este capítulo vamos a ejecutar el banco de pruebas para Glibc. Sin embargo, hay que resaltar que ejecutar aquí el banco de pruebas de Glibc no se considera tan importante como ejecutarlo en el Capítulo 6.

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 antes de construir Glibc.

Básicamente, compilar Glibc de forma diferente a como el libro sugiere pone la estabilidad de tu sistema en grave riesgo.

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

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

Igualmente, Glibc tiene un sutil problema cuando se compila con GCC 3.3.1. Aplica el siguiente parche para corregirlo:

```
patch -Np1 -i ../glibc-2.3.2-sscanf-1.patch
```

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-2.3.2/configure --prefix=/tools \
    --disable-profile --enable-add-ons \
    --with-headers=/tools/include \
    --with-binutils=/tools/bin \
    --without-gd
```

El significado de las opciones de configure:

- **`--disable-profile`**: Esto desactiva la construcción de librerías con información de perfiles. Este comando puede omitirse si planeas usar perfiles.
- **`--enable-add-ons`**: Esto activa los añadidos que se instalarán con Glibc, en nuestro caso Linuxthreads.
- **`--with-binutils=/tools/bin`** y **`--with-headers=/tools/include`**: Estrictamente hablando, estas opciones no son necsarias, 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`**: Esta opción asegura que no se construya el programa **`memusagestat`**, el cual insiste extrañamente en enlazarse contra las librerías del sistema anfitrión (libgd, libpng, libz, y demás).

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:
*** Estos programas auxiliares no se encontraron o son versiones incompatibles: msgfmt
*** algunas características serán desactivadas.
*** Compruebe en el fichero INSTALL la versión requerida.
```

La ausencia o incompatibilidad del programa `msgfmt` normalmente es inofensiva, pero se cree que en ocasiones puede causar problemas al ejecutar el banco de pruebas.

Compila el paquete:

```
make
```

Ejecuta el banco de pruebas:

```
make check
```

El banco de pruebas de Glibc depende en gran medida de ciertas funciones de tu sistema anfitrión, en particular del núcleo. Adicionalmente aquí, 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. 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 tambien 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 es la que acabaremos usando al final, por lo que es la que realmentente queremos ver pasar. Pero recuerda, incluso en el Capítulo 6 puede que todavía ocurran algunos fallos, por ejemplo la prueba *math*. 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, asegurate de registrar la salida para que más tarde puedas revisar el fichero de registro y examinar el número total de errores.

Ahora instala el paquete:

```
make install
```

Diferentes paises 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*. Instalaremos ahora las locales de Glibc:

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

Una alternativa al comando anterior es instalar sólamente 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-2.3.2`. 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 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
```

"Bloquear" Glibc

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 especificaciones del compilador.

Primero, instala el enlazador ajustado ejecutando el siguiente comando desde el directorio `binutils-build`:

```
make -C ld install
```

El enlazador se ajustó anteriormente, al final del primer paso de Binutils. Desde ahora todo se enlazará *sólamente* 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 está todo perdido. Sólo ignora el comando anterior. El resultado es la pequeña pega de que los siguientes programas se enlazarán contra las librerías del anfitrión. Esto no es lo ideal, pero no es un gran problema. La situación se corrige cuando instalemos más tarde el segundo paso 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 de especificaciones a mano si quieres: simplemente reemplaza "/lib/ld−linux.so.2" con "/tools/lib/ld−linux.so.2".

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

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
gcc dummy.c
readelf -l a.out | grep ': /tools'
```

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

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

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

Si no obtienes una salida como la mostrada, o no hay ninguna salida, algo está realmente mal. Necesitarás investigar y revisar tus pasos para encontrar el problema y corregirlo. No hay razón para continuar hasta hacer esto. Muy posiblemente algo fué mal con las modificaciones anteriores en los ficheros de especificaciones. Advierte especialmente que `/tools/lib` aparece como prefijo de nuestro enlazador dinámico. Por supuesto, si estás trabajando en una plataforma en la que el nombre del enlazador dinámico es distinto a `ld-linux.so.2`, entonces la salida será ligeramente diferente.

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

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

Esto completa la instalación de un conjunto de herramientas autosuficiente, que ahora pueden usarse para construir el resto de las herramientas temporales.

Instalación de Tcl−8.4.4

```
Tiempo estimado de construcción:                0.9 SBU
Estimación del espacio necesario en disco:      23 MB
```

Contenido de Tcl

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

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

Librería instalada: libtcl8.4.so

Dependencias de instalación de Tcl

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.

Prepara Tcl para su compilación:

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

Construye el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Sin embargo, se sabe que en este capítulo el banco de pruebas de Tcl 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. Si decides ejecutarlo, el siguiente comando lo hará:

```
TZ=UTC make test
```

Significado del parámetro de make:

- **TZ=UTC**: Esto establece la zona horaria al Tiempo Universal Coordinado (UTC), tambien 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. Mas adelante, en el Capítulo 7, tienes

más información sobre la variable de entorno TZ.

En ocasiones, el banco de pruebas de un paquete puede dar falsos errores. Puedes consultar el Wiki de LFS en http://wiki.linuxfromscratch.org/ para comprobar si estos fallos son normales. Esto es aplicable a todas las comprobaciones que se hagan a lo largo del libro.

Instala el paquete:

```
make install
```

Importante: *No borres* todavía el directorio de fuentes de `tcl8.4.4`, ya que el próximo paquete necesitará sus ficheros de cabecera internos.

Crea un enlace simbólico necesario:

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

Instalación de Expect-5.39.0

```
Tiempo estimado de construcción:                0.1 SBU
Estimación del espacio necesario en disco:      3.9 MB
```

Contenido de Expect

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

Programa instalado: expect

Librería instalada: libexpect5.39.a

Dependencias de instalación de Expect

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

Instalación de Expect

Primero, aplica un parche:

```
patch -Np1 -i ../expect-5.39.0-spawn.patch
```

Esto corrige un error en Expect que puede causar falsos fallos durante la ejecución del conjunto de pruebas de GCC.

Ahora, prepara Expect para su compilación:

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

El significado de las opciones de configure:

- **`--with-tcl=/tools/lib`**: Esto asegura que el guión configure encuentra 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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Sin embargo, se sabe que el banco de pruebas para Expect puede experimentar fallos aquí, en el Capítulo 5, 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. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make test
```

E instálalo:

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

El 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.

Instalación de DejaGnu−1.4.3

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     8.6 MB
```

Contenido de DejaGnu

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

Programa instalado: runtest

Dependencias de instalación de DejaGnu

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
```

Instalación de GCC-3.3.1 - Fase 2

```
Tiempo estimado de construcción:                11.0 SBU
Estimación del espacio necesario en disco:      274 MB
```

Renstalación de GCC

Ahora están instaladas las herramientas necesarias para comprobar GCC y Binutils (Expect, Tcl y DejaGnu). Podemos continuar reconstruyendo GCC y Binutils, enlazandolos con la nueva Glibc, y comprobarlos adecuádamente. Sin embargo, una cosa a tener en cuenta es que estos bancos de pruebas son altamente dependientes del correcto funcionamiento de los pseudoterminales (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"
```

Si recibes este mensaje:

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

El sistema no tiene más ptys. Pregunta a tu administrador del sistema para crear más.
```

Tu sistema anfitrión no está configurado para operar correctamente con PTY. 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.

Descomprime los tres paquetes de GCC (-core, -g++, y -testsuite) en un mismo directorio de trabajo. Todos ellos se desempaquetarán en un único subdirectorio `gcc-3.3.1/`.

Primero, corrige un problema y haz un ajuste esencial:

```
patch -Np1 -i ../gcc-3.3.1-no_fixincludes-2.patch
patch -Np1 -i ../gcc-3.3.1-specs-2.patch
```

El primer parche desactiva el guión "fixincludes" de GCC. Antes lo mencionamos brevemente, pero ahora queremos brindarte una explicación un poco más profunda del proceso de corrección de las cabeceras que realiza dicho guión. En circunstancias normales, el guión fixincludes de GCC busca en tu sistema los ficheros

de cabecera que necesita corregir. Puede encontrar que algún fichero de cabecera de Glibc de tu sistema anfitrión necesite ser corregido, en cuyo caso lo corrige y lo pone en un directorio privado de GCC. Más adelante, en el Capítulo 6, después de instalar la nueva Glibc, se buscará en el directorio privado antes que en el directorio del sistema, por lo que GCC encontrará las cabeceras corregidas del sistema anfitrión, que muy probablemente no se corresponderán con la versión de Glibc que usamos para el sistema LFS.

El último 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 de especificaciones 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:** Estos parches son *críticos* para asegurar una correcta construcción. No olvides aplicarlos.

Vuelve a crear un directorio de construcción aparte:

```
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-3.3.1/configure --prefix=/tools \
    --with-local-prefix=/tools \
    --enable-clocale=gnu --enable-shared \
    --enable-threads=posix --enable-__cxa_atexit \
    --enable-languages=c,c++
```

El significado de las nuevas opciones de configure:

- **`--enable-threads=posix`**: Esto activa el manejo de excepciones C++ para codigo 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. Tambien afecta al ABI de C++ obteniendo librerías compartidas y programas C++ interoperables con otras distribuciones Linux.
- **`--enable-clocale=gnu`**: Esta opción asegura que se seleccione el modelo de locale correcto para las librerías 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* corren el riesgo de construir librerías incompatibles con la ABI debido a que se selecciona el modelo de locale *generic* que es erróneo.
- **`--enable-languages=c,c++`**: Esta opción es necesaria para asegurar 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.

> **Nota:** Hay que resaltar que ejecutar aquí el banco de pruebas de GCC no se considera tan importante como ejecutarlo en el Capítulo 6.

Comprueba los resultados:

```
make -k check
```

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

```
../gcc-3.3.1/contrib/test_summary | more
```

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-3.3.1 en sistemas i686-pc-linux-gnu en http://gcc.gnu.org/ml/gcc-testresults/2003-08/msg01612.html.

Advierte que los resultados contienen:

```
* 1 XPASS (unexpected pass) for g++
* 1 FAIL (unexpected failure) for g++
* 2 FAIL for gcc
* 26 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 26 éxitos inesperados para libstdc++ son consecuencia de usar la opción **--enable-clocale=gnu**, que es la elección correcta en los sistemas basados en Glibc versiones 2.2.5 y posteriores. El soporte subyacente a los 'locale' en la librería de C de GNU es superior al modelo "genérico" elegido por defecto (que puede ser aplicable si estuvieras usando Newlibc, Sun-libc u otra libc). El conjunto de pruebas para libstdc++ parece esperar encontrar el modelo "genérico", por lo tanto no se asume que las puebas siempre tendrán éxito.

Los fallos inesperados con frecuencia pueden ignorarse. Los desarrolladores de GCC normalmente los tienen en cuenta pero todavía no se han puesto a corregirlos. 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 la sección *"Bloquear" Glibc* y repite la prueba. Si los resultados son malos muy posiblemente se deba a que olvidaste aplicar el parche Specs de GCC mencionado arriba.

Instalación de Binutils-2.14 - Fase 2

```
Tiempo estimado de construcción:               1.5 SBU
Estimación del espacio necesario en disco:     108 MB
```

Reinstalación de Binutils

Vuelve a crear un directorio separado 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
```

El significado de la nueva opción de configure:

- **`--with-lib-path=/tools/lib`**: Esto le indica al guión configure que se especifica la ruta de búsqueda de librerías por defecto. No queremos que la ruta de búsqueda de librerías contenga directorios de librerías procedentes del sistema 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
```

> **Nota:** Hay que resaltar que ejecutar el banco de pruebas de Binutils aquí no es tan importante como ejecutarlo en el Capítulo 6.

Comprueba los resultados (No debería haber fallos inesperados; los fallos esperados son correctos):

```
make check
```

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
```

E instala el paquete:

```
make install
```

Ahora, prepara Binutils para reajustar las herramientas principales en el 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.

Instalación de Gawk−3.1.3

```
Tiempo estimado de construcción:                0.2 SBU
Estimación del espacio necesario en disco:      17 MB
```

Contenido de Gawk

Gawk es una implementación de awk utilizada para manipular ficheros de texto.

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

Dependencias de instalación de Gawk

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instálalo:

```
make install
```

Instalación de Coreutils−5.0

```
Tiempo estimado de construcción:                0.9 SBU
Estimación del espacio necesario en disco:      69 MB
```

Contenido de Coreutils

El paquete Coreutils contiene un completo conjunto de utilidades básicas para el intérprete de comandos.

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, kill, 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

Dependencias de instalación de Coreutils

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

Instalación de Coreutils

Prepara Coreutils para su compilación:

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

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make RUN_EXPENSIVE_TESTS=yes check
```

El significado del parámetro de make:

- **`RUN_EXPENSIVE_TESTS=yes`**: Esto 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
```

Instalación de Bzip2−1.0.2

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     2.5 MB
```

Contenido de Bzip2

Bzip2 es un compresor de ficheros por ordenación de bloques que, generalmente, consige una mejor compresión que el tradicional **`gzip`**.

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

Dependencias de instalación de Bzip2

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
```

Instalación de Gzip−1.3.5

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     2.6 MB
```

Contenido de Gzip

El paquete Gzip contiene programas para comprimir y descomprimir ficheros usando el codificador Lempel−Ziv (LZ77).

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

Dependencias de instalación de Gzip

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
```

Instalación de Diffutils−2.8.1

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     7.5 MB
```

Contenido de Diffutils

Los programas de este paquete te muestran las diferencias entre dos ficheros o directorios. Es muy común usarlos para crear parches de software.

Programas instalados: cmp, diff, diff3 y sdiff

Dependencias de instalación de Diffutils

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
```

Instalación de Findutils−4.1.20

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     7.6 MB
```

Contenido de Findutils

El paquete Findutils contiene pogramas para encontrar ficheros, tanto al vuelo (haciendo una búsqueda recursiva en vivo a través de los directorios y mostrando sólo los ficheros que cumplan las especificaciones) o mediante una búsqueda a través de una base de datos.

Programas instalados: bigram, code, find, frcode, locate, updatedb y xargs

Dependencias de instalación de Findutils

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Make-3.80

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     8.8 MB
```

Contenido de Make

Make determina, automáticamente, qué piezas de un programa largo es necesario recompilar y ejecuta los comandos para recompilarlas.

Programa instalado: make

Dependencias de instalación de Make

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el programa y su documentación:

```
make install
```

Instalación de Grep−2.5.1

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     5.8 MB
```

Contenido de Grep

Grep es un programa usado para imprimir las líneas de un fichero que cumplan un patrón especificado.

Programas instalados: egrep (enlace a grep), fgrep (enlace a grep) y grep

Dependencias de instalación de Grep

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, pero 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 esto utilizaría el código de Glibc, que se sabe que tiene algunos fallos.

Compila los programas:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete y su documentación:

```
make install
```

Instalación de Sed−4.0.7

```
Tiempo estimado de construcción:               0.2 SBU
```

```
Estimación del espacio necesario en disco:      5.2 MB
```

Contenido de Sed

sed es un editor de flujo. Un editor de flujo se utiliza para realizar transformaciones básicas de texto sobre un flujo de entrada (un fichero o la entrada procedente de una tubería).

Programa instalado: sed

Dependencias de instalación de Sed

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete y su documentación:

```
make install
```

Instalación de Gettext−0.12.1

```
Tiempo estimado de construcción:                7.2 SBU
Estimación del espacio necesario en disco:      55 MB
```

Contenido de Gettext

El paquete Gettext se utiliza para la internacionalización y localización. Los programas pueden compilarse con Soporte de Lenguaje Nativo (NLS), lo que les permite mostrar mensajes en el idioma nativo del usuario.

Programas instalados: autopoint, config.charset, config.rpath, 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]

Dependencias de instalación de Gettext

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Sin embargo, se sabe que aquí, en el Capítulo 5, el banco de pruebas de Gettext causa problemas bajo ciertas condiciones del anfitrión, por ejemplo si encuentra un compilador Java. Tambien tarda mucho tiempo en ejecutarse y no se considera crítico. Por todo esto no recomendamos ejecutarlo aquí. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Ncurses-5.3

```
Tiempo estimado de construcción:               0.7 SBU
Estimación del espacio necesario en disco:     26 MB
```

Contenido de Ncurses

El paquete Ncurses proporciona librerías para el manejo de caractéres y terminales, incluidos paneles y menús.

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

Dependencias de instalación de Ncurses

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

Instalación de Ncurses

Haz dos pequeñas correcciones:

```
patch -Np1 -i ../ncurses-5.3-etip-2.patch
patch -Np1 -i ../ncurses-5.3-vsscanf.patch
```

El primer parche corrige el fichero de cabecera `etip.h` y el segundo hace que el compilador no nos muestre ciertos mensajes de advertencia relacionados con el uso de ficheros de cabecera anticuados.

Ahora, 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
```

Instalación de Patch-2.5.4

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     1.9 MB
```

Contenido de Patch

El programa patch modifica un fichero basandose en un parche. Normalmente un parche es una lista, creada por el programa diff, que contiene instrucciones sobre cómo debe modificarse el fichero original.

Programa instalado: patch

Dependencias de instalación de Patch

Patch depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Instalación de Patch

Prepara Patch para su compilación:

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

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

Compila el programa:

```
make
```

E instálalo con su documentación:

```
make install
```

Instalación de Tar-1.13.25

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     10 MB
```

Contenido de Tar

Tar es un programa de archivado diseñado para almacenar y extraer ficheros en un archivo conocido como fichero tar.

Programas instalados: rmt y tar

Dependencias de instalación de Tar

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

Luego, instala el paquete y su documentación:

```
make install
```

Instalación de Texinfo−4.6

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     16 MB
```

Contenido de Texinfo

El paquete Texinfo contiene programas usados para leer, escribir y convertir documentos Info, que suministran documentación del sistema.

Programas instalados: info, infokey, install−info, makeinfo, texi2dvi y texindex

Dependencias de instalación de Texinfo

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

Luego, instala el paquete y su documentación:

```
make install
```

Instalación de Bash−2.05b

```
Tiempo estimado de construcción:               1.2 SBU
Estimación del espacio necesario en disco:     27 MB
```

Contenido de Bash

bash es la "Bourne−Again SHell", que es un completo intérprete de comandos usado ampliamente en sistemas Unix. El programa bash lee de la entrada estándar (el teclado). Un usuario escribe algo y el programa evalúa lo que ha escrito y hace algo con ello, como lanzar un programa.

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

Dependencias de instalación de Bash

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
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
```

Instalación de Util−linux−2.12

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     8 MB
```

Contenido de Util−linux

El paquete Util−linux contiene una miscelánea de utilidades. Algunas de las utilidades más destacables son las utilizadas para montar, desmontar, formatear, particionar y manejar dispositivos de disco, abrir puertos de consola o capturar los mensajes del núcleo.

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, kill, 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

Dependencias de instalación de Util−linux

Util−linux depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Zlib.

Instación de Util−linux

Util−linux no usa las cabeceras y librerías recién instaladas en el directorio /tools. Esto lo corregimos 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
```

Compila algunas de las rutinas de soporte:

```
make −C lib
```

Y como sólo necesitarás algunas de las utilidades contenidas en este paquete, construye únicamente estas:

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

Ahora, copia estos programas al directorio temporal de herramientas:

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

Instalación de Perl−5.8.0

```
Tiempo estimado de construcción:               0.8 SBU
Estimación del espacio necesario en disco:     74 MB
```

Contenido de Perl

El paquete Perl contiene perl, el Lenguaje Práctico de Extracción e Informe. Perl combina alguna de las mejores características de C, sed, awk y sh dentro de un poderoso lenguaje.

Programas instalados: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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)

Dependencias de instalación de Perl

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-5.8.0-libc-3.patch
```

Y asegúrate que se construyan algunas extensiones estáticas:

```
chmod u+w hints/linux.sh
echo 'static_ext="IO re Fcntl"' >> hints/linux.sh
```

Ahora, prepara Perl para su compilación:

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

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/perl5/5.8.0
cp -R lib/* /tools/lib/perl5/5.8.0
```

Eliminación de símbolos

Los pasos de esta sección son opcionales. Si tu partición LFS es pequeña, te encantará saber que puedes librarte de 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-unneeded /tools/{,s}bin/*
strip --strip-debug /tools/lib/*
```

El primero de los comandos anteriores se saltará unos veinte ficheros, avisando de que no reconoce su formato. Muchos de ellos son guiones en vez de binarios.

Ten cuidado de *no* utilizar **`--strip-unneeded`** con las librerías, podría destruirlas y tendrías que construir Glibc de nuevo.

Para recuperar algunos megas más puedes eliminar la documentación:

```
rm -rf /tools/{,share/}{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, tambien puedes construir e instalar el resto.

III. Parte III − Construcción del sistema LFS

Índice

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 programass 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.

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 rapido 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.

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).

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.

Pero 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 eliminar los símbolos de depuración de un binario (que debe ser un binario a.out o ELF) ejecuta **`strip --strip-debug fichero`**. Pueden usarse comodines para procesar múltiples ficheros (utilizando algo como: **`strip --strip-debug $LFS/tools/bin/*`**).

Para tu comodidad, en el Capítulo 9 se incluye un comando simple para eliminar todos los símbolos de depuración de los programas y librerías del sistema. Puedes encontrar información adicional en la receta de optimización que hay en http://www.escomposlinux.org/lfs-es/recetas/optimization.html (el original se encuentra en http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt).

Entrar al entorno chroot

Es hora de entrar en el entorno chroot para instalar los paquetes que necesitamos. Antes de que puedas hacer chroot, sin embargo, necesitas cambiar al usuario *root*, pues sólo él puede usar el comando **`chroot`**.

Al igual que antes, asegurate de que la variable de entorno LFS es correcta ejecutando **`echo $LFS`** y verificando que muestre la ruta al punto de montaje de tu partición LFS, que es `/mnt/lfs` si seguiste nuestro ejemplo.

Hazte *root* y ejecuta el siguiente comando para entrar al entorno chroot:

```
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
```

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 fijará la variable TERM dentro del chroot al mismo valor que fuera del chroot, pues programas como **`vim`** y **`less`** la necesitan para funcionar correctamente. Si necesitas tener presentes otras variables, como CFLAGS o CXXFLAGS, éste 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 binários ejecutados; por esta razón se desactiva un poco mas adelante su tabla interna de rutas.

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 entrar en el chroot y montar los sistemas de ficheros proc y devpts (como explicaremos más tarde) 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 posteriomente 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,tmp,usr/local,var,opt}
for dirname in /usr /usr/local
    do
    mkdir $dirname/{bin,etc,include,lib,sbin,share,src}
    ln -s share/{man,doc,info} $dirname
    mkdir $dirname/share/{dict,doc,info,locale,man}
    mkdir $dirname/share/{nls,misc,terminfo,zoneinfo}
    mkdir $dirname/share/man/man{1,2,3,4,5,6,7,8}
done
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.

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. 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.

El sistema de ficheros *proc* es el pseudosistema de ficheros de información de procesos que el núcleo utiliza para suministrar información sobre el estado del sistema.

El sistema de ficheros proc se monta en `/proc` ejecutando el siguiente comando.

```
mount proc /proc -t proc
```

Posiblemente el comando mount te muestre algunos mensajes de aviso como este:

```
warning: can't open /etc/fstab: No such file or directory
not enough memory

aviso: no se puede abrir /etc/fstab: No existe el fichero o directorio
memoria insuficiente
```

Ignóralos, se deben al hecho de que el sistema aún no se ha instalado por completo y faltan algunos ficheros. El montaje se realizará correctamente, que es todo lo que necesitamos en este momento.

Anteriormente se mencionó el sistema de ficheros *devpts*, que actualmente es el modo mas común de implementar los pseudoterminales (PTYs).

El sistema de ficheros devpts se monta en `/dev/pts` ejecutando:

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

Puede que este 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 problable 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 un comando como **`cat /proc/filesystems`**. Si está listado un tipo de sistema de ficheros llamado *devfs*, entonces seremos capaces de solventar el problema montando el sistema de ficheros devfs del anfitrión encima de la nueva estructura `/dev` que crearemos más tarde en la sección "Creación de los dispositivos (Makedev)". Si devfs no está listado, no te preocupes, pues aún hay un tercer camino para conseguir que los PTYs funcionen dentro del entorno chroot. Pronto cubriremos esto en la sección Makedev.

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.

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 y grupos

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 MAKEDEV utiliza en la siguiente sección. Aparte del grupo "root", el LSB (http://www.linuxbase.org) sólamente recomienda que 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 terminar, reentraremos en el entorno chroot. La resolución de usarios y grupos empezará a funcionar inmediatamente después de crear los ficheros `/etc/passwd` y `/etc/group` debido a que instalamos una Glibc completa en el Capítulo 5. Esto eliminará el "I have no name!" del símbolo del sistema.

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

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

Creación de los dispositivos (Makedev-1.7)

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     50 KB
```

Contenido de MAKEDEV

MAKEDEV es un guión que crea los ficheros de dispositivos estáticos necesarios, que usualmente residen en el directorio `/dev`. Puede encontrarse más información sobre los ficheros de dispositivos dentro de las fuentes del núcleo en `Documentation/devices.txt`.

Guión instalado: MAKEDEV

Dependencias de instalación de MAKEDEV

Make depende de: Bash, Coreutils.

Creación de los dispositivos

Ten en cuenta que al dempaquetar el fichero `MAKEDEV-1.7.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 `MAKEDEV`:

```
bzcat MAKEDEV-1.7.bz2 > /dev/MAKEDEV
chmod 754 /dev/MAKEDEV
```

Prepara el guión para su ejecución:

```
cd /dev
./MAKEDEV -v generic-nopty
```

Significado de los argumentos:

- **`-v`**: Esto le indica al guión que se ejecute en modo detallado.
- **`generic-nopty`**: Esto le indica a **`MAKEDEV`** que cree una selección genérica de los ficheros especiales de dispositivo comúnmente usados, excepto para los rangos de ficheros ptyXX y ttyXX. No necesitaremos estos ficheros debido a que vamos a usar los PTYs Unix98 mediante el sistema de ficheros *devpts*.

Si resulta que no encuentras algún dispositivo especial `zzz` que necesitas, prueba a ejecutar **`./MAKEDEV -v zzz`**. Alternativamente, puedes crear los dispositivos mediante el programa *mknod*. Consulta sus páginas de manual e info si necesitas más información.

Adicionalmente, si en la anterior sección "Montar los sistemas de ficheros proc y devpts" fuiste incapaz de montar el sistema de ficheros devpts, ahora es el momento de probar con las alternativas. Si tu núcleo soporta el sistema de ficheros devfs, ejecuta el siguiente comando para montar devfs:

```
mount -t devfs devfs /dev
```

Esto montará el sistema de ficheros devfs encima de la nueva estructura `/dev` estática. Esto no supone un problema pues los nodos de dispositivo creados todavía están presentes, sólo están ocultos bajo el nuevo sistema de ficheros devfs.

Si esto tampoco funciona, la única opción que queda es usar el guión MAKEDEV para crear los rangos de ficheros ptyXX y ttyXX que de otra forma no serían necesarios. Asegurate de que aún estás en el directorio `/dev` y ejecuta **`./MAKEDEV -v pty`**. La contrapartida de esto es que estamos creando 512 ficheros especiales de dispositivo extras que no serán necesarios cuando finalmente arranquemos nuestro sistema LFS terminado.

Instalación de las cabeceras de Linux-2.4.22

```
Tiempo estimado de construcción:                0.1 SBU
Estimación del espacio necesario en disco:      186 MB
```

Contenido de Linux

El núcleo Linux es el corazón de todo sistema Linux. Es lo que hace a Linux funcionar. Cuando se enciende un ordenador y se inicia un sistema Linux, el núcleo es lo primero que se carga. El núcleo inicializa los componentes hardware del sistema: puertos serie, puertos paralelo, tarjetas de sonido, tarjetas de red, controladores IDE, controladores SCSI y mucho más. En pocas palabras, el núcleo hace que el hardware esté disponible para que el software pueda ejecutarse.

Ficheros instalados: el núcleo y las cabeceras del núcleo.

Dependencias de instalación de Linux

Linux depende de: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Instalación de las cabeceras del núcleo

No compilaremos todavía un nuevo núcleo, lo haremos cuando terminemos la instalación de todos los paquetes. Pero como ciertos paquetes necesitan los ficheros de cabecera del núcleo, vamos a desempaquetar el archivo del núcleo ahora, configurarlo, y copiar los ficheros de cabecera a un lugar donde puedan encontrarlos esos paquetes.

Es importante resaltar que los ficheros del directorio de las fuentes del núcleo no pertenecen a *root*. Aunque desempaquetes un paquete como usuario *root* (como hacemos dentro del chroot), los ficheros seguirán teniendo los identificadores de usuario y grupo que tenían en la computadora en la que se empaquetó. Esto usualmente no es un problema para otros paquetes que instales debido a que eliminas el árbol de las fuentes despues de la instalación. Pero el árbol de las fuentes del núcleo Linux se guarda con frecuencia durante mucho tiempo, por lo que si en alguna ocasión ese identificador es asignado a alguien en tu máquina, esa persona obtendrá permisos de escritura sobre las fuentes del núcleo.

A la vista de esto, puede que quieras ejecutar **`chown -R 0:0`** sobre el directorio `linux-2.4.22` para asegurarte de que todos los ficheros son propiedad del usuario *root*.

Prepara la instalación de las cabeceras:

```
make mrproper
```

Esto asegurará que el árbol del núcleo está absolutamente limpio. El equipo de desarrollo del núcleo recomienda usar este comando antes de *cada* compilación del núcleo. 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:

```
cp -HR include/asm /usr/include
cp -R include/asm-generic /usr/include
```

Instala los ficheros de cabecera del núcleo independientes de la plataforma:

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

Hay ciertos fichero de cabecera del núcleo que hacen uso del fichero de cabecera `autoconf.h`. Puesto que todavía no hemos configurado el núcleo, necesitamos crear este fichero por nuestra cuenta para evitar fallos de compilación. Crea un fichero `autoconf.h` vacio:

```
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 zombi. 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}`.

Instalación de Man−pages−1.60

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     15 MB
```

Contenido de Man−pages

El paquete Man−pages contiene alrededor de 1200 páginas de manual. Esta documentación detalla las funciones de C y C++, describe varios ficheros de dispositivo importantes y proporciona documentación procedente de otros paquetes que posiblemente falte en los mismos.

Ficheros instalados: diversas páginas de manual.

Dependencias de instalación de Man−pages

Man depende de: Bash, Coreutils, Make.

Instalación de Man−pages

Instala Man−pages ejecutando:

```
make install
```

Instalación de Glibc−2.3.2

```
Tiempo estimado de construcción:               12.3 SBU
Estimación del espacio necesario en disco:     784 MB
```

Contenido de Glibc

Glibc es la librería C que proporciona las llamadas al sistema y las funciones básicas, tales como open, malloc, printf, etc. La librería C es utilizada por todos los programas enlazados dinámicamente.

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]

Dependencias de instalación de Glibc

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.

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

Antes de comenzar a construir Glibc, recuerda desempaquetar el archivo Glibc−linuxthreads dentro del directorio `glibc−2.3.2` y desactivar las variables de entorno que puedan sobreescribir las opciones de optimización por defecto.

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
```

Luego, aplica el mismo parche que utilizamos anteriormente:

```
patch -Np1 -i ../glibc-2.3.2-sscanf-1.patch
```

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

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

Ahora prepara Glibc para su compilación:

```
../glibc-2.3.2/configure --prefix=/usr \
    --disable-profile --enable-add-ons \
    --libexecdir=/usr/bin --with-headers=/usr/include
```

El significado de las nuevas opciones de configure:

- **`--libexecdir=/usr/bin`**: Esto hará que el programa `pt_chown` se instale en el directorio `/usr/bin`.
- **`--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 inmediátamente si te olvidaste de instalar los ficheros de cabecera del núcleo en `/usr/include`.

Compila el paquete:

```
make
```

Comprueba los resultados:

```
make check
```

Las notas sobre el banco de pruebas que hay en la sección *Instalación de Glibc-2.3.2* del Capítulo 5 son aún más apropiadas aquí. Asegurate de consultarlas si tienes alguna duda.

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 sólamente aquellas locales que necesites o desees. Esto puede hacerse usando el comando **`localedef`**. Se puede encontrar más información sobre esto en el fichero `INSTALL` del árbol de `glibc-2.3.2`. Sin embargo, hay un número de locales que son esenciales para que las comprobaciones de paquetes posteriores se realicen correctamente. En particular, la prueba de libstdc++ en GCC. Las siguientes instrucciones, en vez del comando anterior install-locales, instalarán el conjunto mínimo

de locales necesario para que las comprobaciones 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 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-2.3.2/linuxthreads/man
```

E instálalas:

```
make -C ../glibc-2.3.2/linuxthreads/man install
```

Configuración de Glibc

Necesitamos crear el fichero `/etc/nsswitch.conf`, porque aunque glibc nos facilta 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

publickey: files

hosts: files dns
networks: files

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

netgroup: db 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
```

Reajustar las herramientas

Ahora que hemos instalado las nuevas librerías de C, es hora de reajustar nuestro conjunto de herramientas. Lo haremos de forma que cada programa que compilemos se enlace con las nuevas librerías de C. Básicamente, revertiremos el "bloqueo" que realizamos al principio del capítulo anterior.

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

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

> **Nota:** Si de algún modo te saltaste el aviso sobre conservar los directorios de las fuentes y construcción del segundo paso de Binutils en el Capítulo 5, o los borraste accidentalmente o

no tienes acceso a ellos, no te preocupes, no todo está perdido. Ignora el comando anterior. El resultado será que el siguiente paquete, Binutils, se enlazará contra las librerías Glibc que hay en `/tools` en vez de las de `/usr`. Esto no es lo ideal, pero nuestras pruebas han mostrado que los programas binarios de Binutils resultantes deberían ser identicos.

Desde ahora todos los programas que compilemos se enlazarán *sólamente* 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 tambien evita esto.

Ya puedes borrar los directorios de fuentes y de construcción de Binutils.

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 verificar el fichero de especificaciones para corroborar que 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 la sección *Notas técnicas sobre las herramientas* del Capítulo 5.

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
gcc dummy.c
readelf -l a.out | grep ': /lib'
```

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

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

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

Si no obtienes una salida como la mostrada, o no hay ninguna salida, algo está realmente mal. Necesitarás investigar y revisar tus pasos para encontrar el problema y corregirlo. No hay razón para continuar hasta hacer esto. Muy posiblemente algo fué mal con las modificaciones anteriores en los ficheros de especificaciones. Advierte especialmente que `/lib` aparece como prefijo de nuestro enlazador dinámico. Por supuesto, si estás trabajando en una plataforma en la que el nombre del enlazador dinámico es distinto a `ld-linux.so.2`, entonces la salida será ligeramente diferente.

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

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

Instalación de Binutils−2.14

```
Tiempo estimado de construcción:                1.4 SBU
Estimación del espacio necesario en disco:      167 MB
```

Contenido de Binutils

Binutils es una colección de herramientas para el desarrollo de software que contiene un enlazador, un ensamblador y otras utilidades para trabajar con ficheros de objetos y archivos.

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]

Dependencias de instalación de Binutils

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 pseudoterminales (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. Pregunta a tu administrador del sistema para crear 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 la sección *Montar los sistemas de ficheros proc y devpts* y la sección *Creación de los dispositivos (Makedev−1.7)* y lleva a cabo los pasos recomendados para corregir el problema.

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

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).

Comprueba los resultados:

```
make check
```

Las notas sobre el banco de pruebas que hay en la sección *Instalación de Binutils-2.14 - Fase 2* del Capítulo 5 son aún más apropiadas aquí. Asegurate 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
```

Instalación de GCC-3.3.1

```
Tiempo estimado de construcción:               11.7 SBU
Estimación del espacio necesario en disco:     294 MB
```

Contenido de GCC

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

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

Dependencias de instalación de GCC

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

Instalación de GCC

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

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.

En esta ocasión construiremos los compiladores C y C++, por lo que necesitarás desempaquetar los paquetes GCC−core *y* GCC−g++, que se desempaquetarán dentro del mísmo 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−5.0/general/gcc.html (el original en inglés está en http://www.linuxfromscratch.org/blfs/view/stable/general/gcc.html).

```
patch −Np1 −i ../gcc−3.3.1−no_fixincludes−2.patch
patch −Np1 −i ../gcc−3.3.1−suppress−libiberty.patch
```

El segundo parche suprime la instalación de libiberty de GCC, pues utilizaremos en su lugar la suministrada por Binutils. Ten cuidado de *no* aplicar aquí el parche Specs para GCC del Capítulo 5.

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

```
mkdir ../gcc−build
cd ../gcc−build
```

Ahora prepara GCC para su compilación:

```
../gcc−3.3.1/configure −−prefix=/usr \
    −−enable−shared −−enable−threads=posix \
    −−enable−__cxa_atexit −−enable−clocale=gnu \
    −−enable−languages=c,c++
```

Compila el paquete:

```
make
```

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 la sección *Instalación de GCC−3.3.1 − Fase 2* del Capítulo 5 son aún más apropiadas aquí. Asegurate de consultarlas si tienes alguna duda.

E instala el paquete:

```
make install
```

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

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

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

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

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

Instalación de Coreutils-5.0

```
Tiempo estimado de construcción:               0.9 SBU
Estimación del espacio necesario en disco:     69 MB
```

Contenido de Coreutils

El paquete Coreutils contiene un completo conjunto de utilidades básicas para el intérprete de comandos.

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, kill, 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

Dependencias de instalación de Coreutils

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-5.0-uname.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-5.0-hostname-2.patch
```

Prepara Coreutils para su compilación:

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

Compila el paquete:

```
make
```

El programa **su** de Coreutils no se instaló en el Capítulo 5 debido a que necesita privilegios de *root*. Vamos a necesitarlo dentro de unos momentos para el banco de pruebas. Así que solucionemos el problema instalándolo ahora:

```
make install-root
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Sin embargo, este banco de pruebas en particular hace ciertas suposiciones relativas a la presencia de usuarios y grupos que no sean root y que no son de aplicación en esta temprana fase de la construcción de LFS. Por tanto crearemos un usuario del sistema ficticio y dos grupos ficticios para permitir que las pruebas se realicen correctamente. Si decides no ejecutar el banco de pruebas, salta hasta "Instala el paquete". Los siguientes comandos harán los preparativos para el banco de pruebas. Crea dos grupos y un nombre de usuario ficticios:

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

Algunas pruebas requieren que se ejecuten como *root*:

```
make check-root
```

El resto de las pruebas se ejecutan como usuario *dummy*:

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

Elimina los grupos y el nombre de 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/{dir,dircolors,du,date,echo,false,head} /bin
mv /usr/bin/{install,ln,ls,mkdir,mkfifo,mknod,mv,pwd} /bin
mv /usr/bin/{rm,rmdir,shred,sync,sleep,stty,su,test} /bin
mv /usr/bin/{touch,true,uname,vdir} /bin
mv /usr/bin/chroot /usr/sbin
```

Finalmente, crea algunos enlaces necesarios:

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

Instalación de Zlib-1.1.4

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     1.5 MB
```

Contenido de Zlib

El paquete Zlib contiene la librería libz, utilizada por varios programas para realizar las funciones de compresión y descompresión..

Librería instalada: libz[a,so]

Dependencias de instalación de Zlib

Zlib depende de: Binutils, Coreutils, GCC, Glibc, Make, Sed.

Instalación de Zlib

Zlib puede sufrir un desbordamiento de memoria en la función *gzprintf()*, que, aunque es difícil de explotar, debería ser corregido. Házlo aplicando este parche:

```
patch -Np1 -i ../zlib-1.1.4-vsnprintf.patch
```

Prepara Zlib para su compilación:

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

Advertencia: se sabe que Zlib construye incorrectamente sus librerías si en el entorno se ha especificado un CFLAGS. Si estás usando tus propias variables CFLAGS, asegurate de añadir la directiva ***-fPIC*** durante esta fase, y eliminala posteriormente.

Compila el paquete:

```
make
```

Instala las librerías compartidas:

```
make install
```

Ahora, construye también las librerías no compartidas:

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

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make test
```

E instala el paquete:

```
make install
```

La librería compartida de Zlib debe instalarse en el directorio `/lib`. De este modo, en el caso de que debas arrancar sin el directorio `/usr`, los programas vitales del sistema todavía tendrán acceso a la librería:

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

El enlace simbólico `/usr/lib/libz.so` apunta a un fichero que ya no existe, debido a que lo hemos movido. Crea un enlace simbólico a la nueva localización de la librería:

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

Zlib no instala su página de manual. Ejecuta el siguiente comando para instalar esta documentación:

```
cp zlib.3 /usr/share/man/man3
```

Instalación de Lfs−Utils−0.3

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     1.1 MB
```

Contenido de Lfs−Utils

El paquete Lfs−Utils contiene varios programas usados por otros paquetes, pero que no son lo suficientemente grandes como para proveerlos en paquetes individuales.

Programas instalados: mktemp, tempfile, http−get e iana−net

Ficheros instalados: protocols y services

Dependencias de instalación de Lfs−Utils

(Aún no se comprobaron las dependencias.)

Instalación de Lfs−Utils

Compila el paquete:

```
make
```

E instálalo:

```
make install
```

Ahora copia dos ficheros de soporte incuidos en el paquete Lfs−Utils a su destino:

```
cp etc/{services,protocols} /etc
```

El fichero `/etc/services` se usa para resolver números de servicios a nombres legibles, y el fichero `/etc/protocols` hace lo mismo para los números de protocolos.

Instalación de Findutils−4.1.20

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     7.5 MB
```

Contenido de Findutils

El paquete Findutils contiene pogramas para encontrar ficheros, tanto al vuelo (haciendo una búsqueda recursiva en vivo a través de los directorios y mostrando sólo los ficheros que cumplan las especificaciones) o mediante una búsqueda a través de una base de datos.

Programas instalados: bigram, code, find, frcode, locate, updatedb y xargs

Dependencias de instalación de Findutils

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/bin
```

Por defecto, la base de datos de updatedb se encuentra en `/usr/var`. Para hacer que la localización del fichero `locatedb` cumpla con el FHS, pásale la opción *−−localstatedir=/var/lib/misc* al guión **`configure`**.

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, el siguiente comando lo hará:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Gawk−3.1.3

```
Tiempo estimado de construcción:                0.2 SBU
Estimación del espacio necesario en disco:      17 MB
```

Contenido de Gawk

Gawk es una implementación de awk utilizada para manipular ficheros de texto.

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

Dependencias de instalación de Gawk

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

Instalación de Gawk

Primero aplica un parche para corregir lo siguiente:

- Para Gawk, la ubicación por defecto para algunos de sus ejecutables es `$prefix/libexec/awk`. Esta ubicación no cumple con el FHS, que nunca menciona un directorio llamado `libexec`. Este parche permite que le pasemos la opción *−−libexecdir* al guión configure para que podamos usar una ubicación más apropiada para los binarios **`grcat`** y **`pwcat`**: `/usr/bin`.
- El directorio de datos por defecto de Gawk es `$prefix/share/awk`. Pero un directorio específico de un paquete tendría que ser nombrado usando el paquete y su versión (por ejemplo: `gawk−7.7.2.`) y no simplemente el nombre del paquete, pues podría haber diferentes versiones de un paquete instaladas en el sistema. El parche cambia el nombre del directorio de datos al correcto `$prefix/share/gawk−3.1.3`.
- El parche también se asegura que este directorio de datos y su contenido se desistalen con un *make uninstall*.

```
patch −Np1 −i ../gawk−3.1.3−libexecdir.patch
```

Ahora prepara Gawk para su compilación:

```
./configure −−prefix=/usr −−libexecdir=/usr/bin
```

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Ncurses−5.3

```
Tiempo estimado de construcción:               0.6 SBU
Estimación del espacio necesario en disco:     27 MB
```

Contenido de Ncurses

El paquete Ncurses proporciona librerías para el manejo de caractéres y terminales, incluidos paneles y menús.

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

Dependencias de instalación de Ncurses

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

Instalación de Ncurses

Primero, corrige dos pequeños errores:

```
patch −Np1 −i ../ncurses−5.3−etip−2.patch
patch −Np1 −i ../ncurses−5.3−vsscanf.patch
```

El primer parche corrige el fichero de cabecera `etip.h`, mientras que el segundo evita que el compilador nos muestre advertencias sobre los ficheros de cabecera en desuso.

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/*.5.3
```

Y 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
```

Puesto que las librerías se han movido a `/lib`, algunos enlaces simbólicos apuntan ahora 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
```

Instalación de Vim−6.2

```
Tiempo estimado de construcción:                0.4 SBU
Estimación del espacio necesario en disco:      34 MB
```

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−5.0/postlfs/editors.html las instrucciones de instalación sugeridas (el original se encuentra en http://www.linuxfromscratch.org/blfs/view/stable/postlfs/editors.html).

Contenido de Vim

El paquete Vim contiene un editor de texto configurable construido para obtener una eficiente edición del texto.

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

Dependencias de instalación de Vim

Vim depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Instalación de Vim

Cambia la localización por defecto de los ficheros `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
```

E instálalo:

```
make install
```

Vim puede ejecutarse con el modo antiguo de *vi* mediante un enlace simbólico, que puede crearse con el siguiente comando:

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

Si piensas 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 modo vim (de otra forma, no habríamos incluido vim en este libro, sino el vi original). Crea el fichero `/root/.vimrc` ejecutando lo siguiente:

```
cat > /root/.vimrc << "EOF"
" Inicio de /root/.vimrc

set nocompatible
set bs=2

" Fin de /root/.vimrc
EOF
```

Instalación de M4−1.4

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     3.0 MB
```

Contenido de M4

M4 es un procesador de macros. Copia la entrada a la salida expandiendo las macros en el proceso. Las macros pueden ser internas o definidas por el usuario y pueden tomar cualquier número de argumentos. Aparte de hacer la expansión de macros, m4 tiene funciones internas para la inclusión de 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.

Programa instalado: m4

Dependencias de instalación de M4

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Bison-1.875

```
Tiempo estimado de construcción:               0.6 SBU
Estimación del espacio necesario en disco:     10.6 MB
```

Contenido de Bison

Bison es un generador de analizadores sintácticos, un sustituto de yacc. Bison genera un programa que analiza la estructura de un fichero de texto.

Programas instalados: bison y yacc

Librería instalada: liby.a

Dependencias de instalación de Bison

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

Instalación de Bison

Primero utilizaremos un parche para bison, extraido del CVS, que corrige un problema menor de compilación con algunos paquetes:

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

Prepara Bison para su compilación:

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

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando (tarda bastante tiempo):

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Less−381

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     3.4 MB
```

Contenido de Less

Less es un paginador de ficheros, o visor de texto. Muestra el contenido de un fichero, o de un flujo de datos, y tiene la habilidad de poder recorrerlo. Less tiene várias características no incluidas en el paginador **`more`**, como la habilidad de recorrer los ficheros hacia atrás.

Programas instalados: less, lessecho y lesskey

Dependencias de instalación de Less

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
```

E instálalo:

```
make install
```

Instalación de Groff−1.19

```
Tiempo estimado de construcción:               0.5 SBU
Estimación del espacio necesario en disco:     43 MB
```

Contenido de Groff

El paquete Groff incluye varios programas de procesamiento de texto para formatear el texto. Groff traduce el texto estándar y los comandos especiales a una salida formateada, como la que puedes ver en las páginas de manual.

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)

Dependencias de instalación de Groff

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
```

E 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
```

Instalación de Sed−4.0.7

```
Tiempo estimado de construcción:               0.2 SBU
```

```
Estimación del espacio necesario en disco:      5.2 MB
```

Contenido de Sed

sed es un editor de flujo. Un editor de flujo se utiliza para realizar transformaciones básicas de texto sobre un flujo de entrada (un fichero o la entrada procedente de una tubería).

Programa instalado: sed

Dependencias de instalación de Sed

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Flex−2.5.4a

```
Tiempo estimado de construcción:                0.1 SBU
Estimación del espacio necesario en disco:      3.4 MB
```

Contenido de Flex

El pquete Flex se utiliza para generar programas que reconocen patrones de texto.

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

Librería instalada: libfl.a

Dependencias de instalación de Flex

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make bigcheck
```

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 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
```

Instalación de Gettext-0.12.1

```
Tiempo estimado de construcción:               6.9 SBU
Estimación del espacio necesario en disco:     55 MB
```

Contenido de Gettext

El paquete Gettext se utiliza para la internacionalización y localización. Los programas pueden compilarse con Soporte de Lenguaje Nativo (NLS), lo que les permite mostrar mensajes en el idioma nativo del usuario.

Programas instalados: autopoint, config.charset, config.rpath, 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]

Dependencias de instalación de Gettext

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando (tarda bastante tiempo):

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Net−tools−1.60

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     9.4 MB
```

Contenido de Net−tools

El paquete Net−tools contiene una colección de programas que forman la base del trabajo en red en Linux.

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)

Dependencias de instalación de Net−tools

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 con:

```
make config
```

Si quieres aceptar todas las respuestas que se ofrecen por defecto, puedes saltarte las preguntas planteadas por *make config* ejecutando en su lugar **`yes "" | make config`**.

Compila el paquete:

```
make
```

E instálalo:

```
make update
```

Instalación de Inetutils−1.4.2

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     11 MB
```

Contenido de Inetutils

El paquete Inetutils contiene clientes y servidores de red.

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

Dependencias de instalación de Inetutils

Inetutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Instalación de Inetutils

Prepara Inetutils para su compilación:

```
./configure --prefix=/usr --disable-syslogd \
    --libexecdir=/usr/sbin --disable-logger \
    --sysconfdir=/etc --localstatedir=/var \
    --disable-whois --disable-servers
```

Significado de las opciones de configure:

- **`--disable-syslogd`**: Esta opción evita que inetutils instale el Demonio de Registro de Eventos del Sistema, que será instalado con el paquete Sysklogd.
- **`--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-whois`**: Esta opción desactiva la contrucción 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 contrucción de los diferentes servidores incluidos como parte del paquete Inetutils. Estos servidores no se condideran 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−5.0/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
```

Y mueve el programa **`ping`** a su lugar correcto:

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

Instalación de Perl−5.8.0

```
Tiempo estimado de construcción:                2.9 SBU
Estimación del espacio necesario en disco:      143 MB
```

Contenido de Perl

El paquete Perl contiene perl, el Lenguaje Práctico de Extracción e Informe. Perl combina alguna de las mejores características de C, sed, awk y sh dentro de un poderoso lenguaje.

Programas instalados: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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)

Dependencias de instalación de Perl

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

Instalación de Perl

Prepara Perl para compilarlo:

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

Si quieres más control sobre la forma en que Perl se autoconfigura para construirse, puedes ejecutar en su lugar el guión interactivo **`Configure`** y modificar el modo en que perl se construye. Si piensas que puedes vivir con los (razonables) valores por defecto que Perl autodetecta, entonces usa el comando listado anteriormente.

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo primero debes crear un fichero `/etc/hosts` básico, necesario 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
```

E instala el paquete:

```
make install
```

Instalación de Texinfo−4.6

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     17 MB
```

Contenido de Texinfo

El paquete Texinfo contiene programas usados para leer, escribir y convertir documentos Info, que suministran documentación del sistema.

Programas instalados: info, infokey, install−info, makeinfo, texi2dvi y texindex

Dependencias de instalación de Texinfo

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

Intalación de Texinfo

Prepara Texinfo para su compilación:

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

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

Instala el paquete:

```
make install
```

Y, 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.

Instalación de Autoconf-2.57

```
Tiempo estimado de construcción:               2.9 SBU
Estimación del espacio necesario en disco:     7.7 MB
```

Contenido de Autoconf

Autoconf produce guiones del intérprete de comandos que configuran automáticamente el código fuente.

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

Dependencias de instalación de Autoconf

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Automake−1.7.6

```
Tiempo estimado de construcción:               5.3 SBU
Estimación del espacio necesario en disco:     6.8 MB
```

Contenido de Automake

Automake genera ficheros Makefile.in, pensados para usar con Autoconf.

Programas instalados: acinstall, aclocal, aclocal−1.7, automake, automake−1.7, compile, config.guess, config.sub, depcomp, elisp−comp, install−sh, mdate−sh, missing, mkinstalldirs, py−compile, ylwrap

Dependencias de instalación de Automake

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

Instala el paquete:

```
make install
```

Y crea un enlace simbólico necesario:

```
ln −s automake−1.7 /usr/share/automake
```

Instalación de Bash−2.05b

```
Tiempo estimado de construcción:                1.2 SBU
Estimación del espacio necesario en disco:      27 MB
```

Contenido de Bash

bash es la "Bourne−Again SHell", que es un completo intérprete de comandos usado ampliamente en sistemas Unix. El programa bash lee de la entrada estándar (el teclado). Un usuario escribe algo y el programa evalúa lo que ha escrito y hace algo con ello, como lanzar un programa.

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

Dependencias de instalación de Bash

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
```

Prepara Bash para su compilación:

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

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make tests
```

Instala el paquete:

```
make install
```

Y recarga el programa **bash** recién compilado:

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

Instalación de File−4.04

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     6.3 MB
```

Contenido de File

File es una utilidad usada para determinar el tipo de los ficheros.

Programa instalado: file

Librería instalada: libmagic.[a,so]

Dependencias de instalación de File

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
```

E instálalo:

```
make install
```

Instalación de Libtool−1.5

```
Tiempo estimado de construcción:               1.5 SBU
Estimación del espacio necesario en disco:     20 MB
```

Contenido de Libtool

GNU libtool es un guión para soporte genérico de librerías. Libtool oculta la complejidad del uso de librerías compartidas tras una interfaz consistente y portable.

Programas instalados: libtool y libtoolize

Librerías instaladas: libltdl.[a,so].

Dependencias de instalación de Libtool

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Bzip2−1.0.2

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     3.0 MB
```

Contenido de Bzip2

Bzip2 es un compresor de ficheros por ordenación de bloques que, generalmente, consige una mejor compresión que el tradicional **`gzip`**.

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

Dependencias de instalación de Bzip2

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
```

Instálalo:

```
make install
```

Copia el binario dinámico **`bzip2`** al 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
```

Instalación de Diffutils-2.8.1

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     7.5 MB
```

Contenido de Diffutils

Los programas de este paquete te muestran las diferencias entre dos ficheros o directorios. Es muy común usarlos para crear parches de software.

Programas instalados: cmp, diff, diff3 y sdiff

Dependencias de instalación de Diffutils

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
```

E instala el paquete:

```
make install
```

Instalación de Ed-0.2

```
Tiempo estimado de construcción:                0.1 SBU
Estimación del espacio requierido en disco:     3.1 MB
```

Contenido de Ed

GNU ed es un editor de líneas de 8 bits limpio y que cumple con POSIX.

Programas instalados: ed y red (enlace a ed)

Dependencias de instalación de Ed

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 la sección Temporary Files en http://en.tldp.org/HOWTO/Secure-Programs-HOWTO/avoid-race.html). Este parche hace que Ed use mkstemp, que es la forma recomendada para crear ficheros temporales.

Aplica el parche:

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

Prepara Ed para su compilación:

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

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

Instala el paquete:

```
make install
```

Y mueve los programas al directorio `/bin`, pues deben poder usarse aún en el caso de que la partición `/usr` no esté disponible.

```
mv /usr/bin/{ed,red} /bin
```

Instalación de Kbd−1.08

```
Tiempo estimado de construcción:                0.1 SBU
Estimación del espacio necesario en disco:      12 MB
```

Contenido de Kbd

Kbd contiene ficheros de mapas de teclado y utilidades para el teclado.

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

Dependencias de instalación de Kbd

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−1.08−more−programs.patch
```

Ahora repara Kbd para compilarlo:

```
./configure
```

Compila el paquete:

```
make
```

E instálalo:

```
make install
```

Instalación de E2fsprogs−1.34

```
Tiempo estimado de construcción:               0.6 SBU
Estimación del espacio necesario en disco:     48.4 MB
```

Contenido de E2fsprogs

E2fsprogs proporciona las utilidades para los sistemas de ficheros ext2. También soporta los sistemas de ficheros ext3 con registro de transaciones.

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]

Dependencias de instalación de E2fsprogs

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-1.34/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 pasaramos 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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

Comienza la instalación del paquete:

```
make install
```

E instala las librerías compartidas:

```
make install-libs
```

Instalación de Grep-2.5.1

```
Tiempo estimado de construcción:        0.1 SBU
Estimación del espacio necesario en disco:      5.8 MB
```

Contenido de Grep

Grep es un programa usado para imprimir las líneas de un fichero que cumplan un patrón especificado.

Programas instalados: egrep (enlace a grep), fgrep (enlace a grep) y grep

Dependencias de instalación de Grep

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, el siguiente comando lo hará:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Grub−0.93

```
Tiempo estimado de construcción:                0.2 SBU
Estimación del espacio necesario en disco:      10 MB
```

Contenido de Grub

El paquete Grub contiene un cargador de arranque.

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

Dependencias de instalación de Grub

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.

Primero corrige un problema de compilación con GCC−3.3.1:

```
patch −Np1 −i ../grub−0.93−gcc33−1.patch
```

Ahora prepara Grub para su compilación:

```
./configure −−prefix=/usr
```

Compila el paquete:

```
make
```

E 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`.

Instalación de Gzip−1.3.5

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     2.6 MB
```

Contenido de Gzip

El paquete Gzip contiene programas para comprimir y descomprimir ficheros usando el codificador Lempel−Ziv (LZ77).

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

Dependencias de instalación de Gzip

Gzip depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Instalación de Gzip

Prepara Gzip para su compilación:

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

El programa **gzexe** guarda en su código la localización del binario **gzip**. Como luego vamos a cambiar la ubicación de este último binario, el siguiente comando asegura que la nueva ubicación se guarde dentro del binario.

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

Compila el paquete:

```
make
```

Instala el paquete:

```
make install
```

Y 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
```

Instalación de Man−1.5m2

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     1.9MB
```

Contenido de Man

Man es un paginador de manuales.

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

Dependencias de instalación de Man

Man depende de: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Instalación de Man

Haremos tres ajustes a las fuentes de Man.

El primer parche comenta la línea "MANPATH /usr/man" del fichero `man.conf` para evitar resultados redundantes cuando utilicemos programas como **`whatis`**:

```
patch −Np1 −i ../man−1.5m2−manpath.patch
```

El segundo parche añade la opción *−R* a la variable *PAGER* para que las secuencias de escape se manejen correctamente:

```
patch −Np1 −i ../man−1.5m2−pager.patch
```

El tercero y último parche evita problemas cuando las páginas de manual que no están formateadas para más de 80 columnas se usan en conjunto con las versiones recientes de **`groff`**:

```
patch −Np1 −i ../man−1.5m2−80cols.patch
```

Ahora prepara Man para compilarlo:

```
./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
```

E 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-5.0/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.

Instalación de Make-3.80

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     8.8 MB
```

Contenido de Make

Make determina, automáticamente, qué piezas de un programa largo es necesario recompilar y ejecuta los comandos para recompilarlas.

Programa instalado: make

Dependencias de instalación de Make

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
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Modutils−2.4.25

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     2.9 MB
```

Contenido de Modutils

El paque Modutils contiene programas que puedes utilizar para trabajar con los módulos del núcleo.

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)

Dependencias de instalación de Modutils

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
```

E instálalo:

```
make install
```

Instalación de Patch−2.5.4

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     1.9 MB
```

Contenido de Patch

El programa patch modifica un fichero basandose en un parche. Normalmente un parche es una lista, creada por el programa diff, que contiene instrucciones sobre cómo debe modificarse el fichero original.

Programa instalado: patch

Dependencias de instalación de Patch

Patch depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Instalación de Patch

Prepara Patch para su compilación:

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

De nuevo, la opción **`-D_GNU_SOURCE`** del preprocesor sólo es necesaria en la plataforma PowerPC. En otras plataformas puedes ignorarla.

Compila el paquete:

```
make
```

E instálalo:

```
make install
```

Instalación de Procinfo-18

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     0.2 MB
```

Contenido de Procinfo

El programa procinfo obtiene datos del sistema, como el uso de la memoria y los números de las interrupciones (IRQ), a partir del directorio `/proc`, y formatea estos datos de una forma atractiva.

Programas instalados: lsdev, procinfo y socklist

Dependencias de instalación de Procinfo

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 obsleta `libtermcap`.

E instala el paquete:

```
make install
```

Instalación de Procps−3.1.11

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     6.2 MB
```

Contenido de Procps

El paquete Procps proporciona programas para supervisar y parar procesos del sistema. Procps obtiene la información sobre los procesos a través del directorio `/proc`.

Programas instalados: free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w y watch

Librería instalada: libproc.so

Dependencias de instalación de Procps

Procps depende de: Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses.

Instalación de Procps

Primero corrige un problema que hace que **`w`** falle bajo ciertos ajustes de las locales:

```
patch −Np1 −i ../procps−3.1.11−locale−fix.patch
```

Ahora compila Procps:

```
make
```

Instálalo:

```
make install
```

Y elimina un enlace simbólico molesto:

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

Instalación de Psmisc−21.3

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     2.2 MB
```

Contenido de Psmisc

El paquete Psmisc contiene tres programas que ayudan en el manejo del directorio `/proc`.

Programas instalados: fuser, killall y pstree

Dependencias de instalación de Psmisc

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` y no en `/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
```

E instálalo:

```
make install
```

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
```

Instalación de Shadow-4.0.3

```
Tiempo estimado de construcción:               0.4 SBU
Estimación del espacio necesario en disco:     11 MB
```

Contenido de Shadow

El paquete Shadow se creó para consolidar la seguridad del sistema de contraseñas.

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

Dependencias de instalación de Shadow

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

Instalación de Shadow

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.

Shadow fija en su código la ruta al binario **passwd**, pero lo hace de forma incorrecta. Si no existe un binario **passwd** antes de instalar Shadow, este asume erróneamente que el binario estará en `/bin/passwd`, pero luego lo instala en `/usr/bin/passwd`, lo que llevará a obtener errores extraños acerca de que no puede encontrar `/bin/passwd`. Para corregir este error de Shadow crearemos un fichero `passwd` temporal para que en su código se fije la ruta correcta:

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

El paquete Shadow actual tiene un problema que hace que el comando **newgrp** falle. El siguiente parche (también aparece en el código CVS de Shadow) corrige este problema.

```
patch -Np1 -i ../shadow-4.0.3-newgrp-fix.patch
```

Ahora, prepara Shadow para su compilación:

```
./configure --prefix=/usr --libdir=/usr/lib --enable-shared
```

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
```

Queremos cambiar el método de contraseñas para activar las contraseñas MD5 que, teoricamente, son mas seguras que el método "crypt" que hay por defecto, y tambien permite contraseñas de mas de 8 caracteres de longitud. Al mismo tiempo necesitamos cambiar la antigua localización de los buzones de correo de usuario `/var/spool/mail` a su localización actual `/var/mail`. Haremos esto cambiando en el fichero adecuado de configuración mientras lo copiamos a su destino:

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

> **Nota:** Ten mucho cuidado cuando teclees lo anterior. Posiblemente sea más seguro cortar y copiar que intentar escribirlo.

De acuerdo a la página de manual de **`vipw`**, debería existir también el programa **`vigr`**. Como el procedimiento de instalación no crea este programa, haz manualmente un enlace simbólico:

```
ln -s vipw /usr/sbin/vigr
```

Puesto que el enlace `vipw` es redundante (además de apuntar a un fichero que no existe), elimínalo:

```
rm /bin/vipw
```

Mueve el programa **`sg`** a una ubicación adecuada:

```
mv /bin/sg /usr/bin
```

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
```

Coreutils ya ha instalado un programa **`groups`** en `/usr/bin`. Si lo deseas, puedes borrar el que acaba de instalar Shadow:

```
rm /bin/groups
```

Configuración del entorno de contraseñas ocultas (Shadow Password Suite)

Este paquete contiene las utilidades para modificar las contraseñas de los usuarios, añadir o borrar usuarios y grupos, etc. No vamos a explicar lo que significa 'password shadowing'. Puedes econtrar una completa explicación en el fichero `doc/HOWTO` que está en el árbol de fuentes de Shadow al desempaquetarlo. Hay una cosa que debes recordar si decides usar soporte para Shadow: los programas que necesiten verificar contraseñas (por ejemplo xdm, demonios de ftp, demonios de pop3, etc) necesitarán ser 'compatibles con

shadow', es decir, necesitan ser capaces de trabajar con contraseñas ocultas.

Para habilitar las contraseñas ocultas, ejecuta el siguiente comando:

```
/usr/sbin/pwconv
```

Y para habilitar las contraseñas de grupo ocultas, ejecuta este otro comando:

```
/usr/sbin/grpconv
```

Bajo circunstancias normales aún no habrás creado ninguna contraseña. Sin embargo, si 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**.

Instalación de Sysklogd−1.4.1

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     0.5 MB
```

Contenido de Sysklogd

El paquete Sysklogd contiene programas para grabar tanto los mensajes del sistema como los generados por el núcleo

Programas instalados: klogd y syslogd

Dependencias de instalación de Sysklogd

Sysklogd depende de: Binutils, Coreutils, GCC, Glibc, Make.

Instalación de Sysklogd

Compila Sysklogd:

```
make
```

E 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
```

Instalación de Sysvinit-2.85

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     0.9 MB
```

Contenido de Sysvinit

El paquete Sysvinit contiene programas para controlar el arranque, ejecución y descarga de todos los demás programas.

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

Dependencias de instalación de Sysvinit

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 TERM y KILL a todos los procesos que él mismo inició, mostrando por pantalla el mensaje "Sending processes the TERM signal" (Enviando la señal TERM a los procesos). Esto parece indicar que init envía esta señal a todos los procesos en ejecución. Para evitar esta confusión, puede modificarse el fichero init.c de manera que el mensaje diga "Sending processes started by init the TERM signal" (Enviando la señal TERM a los procesos iniciados por init).

Edita el mensaje de parada:

```
cp src/init.c{,.backup}
sed 's/Sending processes/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
```

Instalación de Tar−1.13.25

```
Tiempo estimado de construcción:                0.2 SBU
Estimación del espacio necesario en disco:      10 MB
```

Contenido de Tar

Tar es un programa de archivado diseñado para almacenar y extraer ficheros en un archivo conocido como fichero tar.

Programas instalados: rmt y tar

Dependencias de instalación de Tar

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/bin
```

Compila el paquete:

```
make
```

Este paquete contiene un banco de pruebas que puede realizar una serie de comprobaciones para asegurar que se ha construido correctamente. Si decides ejecutarlo, hazlo con el siguiente comando:

```
make check
```

E instala el paquete:

```
make install
```

Instalación de Util−linux−2.12

```
Tiempo estimado de construcción:               0.2 SBU
Estimación del espacio necesario en disco:     16 MB
```

Contenido de Util−linux

El paquete Util−linux contiene una miscelánea de utilidades. Algunas de las utilidades más destacables son las utilizadas para montar, desmontar, formatear, particionar y manejar dispositivos de disco, abrir puertos de consola o capturar los mensajes del núcleo.

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, kill, 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

Dependencias de instalación de Util−linux

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_SLN=yes
```

Significado del parámetro de make:

- **HAVE_SLN=yes**: Esto evita que el programa **sln** (un **ln** enlazado estáticamente, ya instalado por Glibc) se vuelva a construir.

E instala el paquete:

```
make HAVE_SLN=yes install
```

Instalación de GCC−2.95.3

```
Tiempo estimado de construcción:                1.5 SBU
Estimación del espacio necesario en disco:      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 proposito de compilar el núcleo Linux en el Capítulo 8. 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−5.0/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
```

Comando chroot revisado

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 es que ya no hoy necesidad de utilizar los programas del directorio `/tools`. Sin embargo, no queremos eliminar en estos momentos el directorio `/tools`. Todavía tienen cierto uso hacia el final del libro.

Instalación de LFS-Bootscripts-1.12

```
Tiempo estimado de construcción:               0.1 SBU
Estimación del espacio necesario en disco:     0.3 MB
```

Contenido de LFS-bootscripts

El paquete LFS-Bootscripts contiene guiones del interprete de comandos para el inicio del sistema al estilo SysV. Estos guiones realizan varias tareas como comprobar la integridad del sistema de ficheros durante el arranque, cargar el mapa del teclado, establecer la red y parar los procesos durante el cierre del sistema.

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

Dependencias de instalación de LFS-Bootscripts

LFS-Bootscripts depende de: Bash, Coreutils.

Instalación de LFS−Bootscripts

Nosotros usaremos guiones de inicio al estilo SysV. Lo hemos elegido porque es ampliamente usado y nos sentimos cómodos con él. Si quieres 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 un cambio más radical, busca "depinit" en las listas de correo de LFS.

Si decides usar el estilo BSD o cualquier otro estilo de guiones, puedes saltarte el siguiente capítulo e ir directamente al Capítulo 8.

Instala los guiones de arranque:

```
cp −a rc.d sysconfig /etc
```

Asígnale a *root* la propiedad de los guiones:

```
chown −R root:root /etc/rc.d /etc/sysconfig
```

Configuración de los componentes del sistema

Ahora que están todos los paquetes instalados, lo que necesitamos hacer son algunas tareas de configuración.

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 `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.

Ejecuta el siguiente comando para parchear el mapa de teclado correcto dentro de las fuentes del núcleo. Debes repetir este comando siempre que desempaquetes un nuevo núcleo:

```
loadkeys −m /usr/share/kbd/keymaps/defkeymap.map.gz > \
    /usr/src/linux−2.4.22/drivers/char/defkeymap.c
```

Establecer la contraseña de root

Elige una contraseña para el administrador (root) y establécela ejecutando el siguiente comando:

```
passwd root
```

Capítulo 7. Preparación de los guiones de arranque

Introducción

En este capítulo se configurarán los guiones de arranque que has instalado en el capítulo 6. Muchos de estos guiones funcionarán sin necesidad de modificaciones, pero algunos requieren ficheros de configuración adicionales para que puedan manejar la información dependiente del hardware específico de tu sistema.

¿Cómo hacen estos guiones que funcione el proceso 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 (también conocido como el reloj CMOS o BIOS) 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 él se explican conceptos como las zonas horarias, UTC, y la variable de entorno TZ.

¿Necesito el guión loadkeys?

Si decidiste compilar tu fichero de mapa de teclado dentro del núcleo al final del Capítulo 6, no es estrictamente necesario ejecutar el guión loadkeys, ya que será el núcleo el que activará dicho mapa. Aunque puedes ejecutarlo de todas maneras sin que te cause ningún problema. De todas formas, es beneficioso que mantengas el guión en el caso de tener varios núcleos y no te acuerdes o no quieras introducir el fichero de

mapa de teclado en todos ellos.

Si has decidido que no necesitas o que no quieres 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 se 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> minombre.midominio.org alias
```

Debes asegurarte de utilizar una dirección IP que pertenezca al rango de direcciones IP privadas. 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 puede ser www.linuxfromscratch.org.

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>.midominio.com <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.localdomain localhost
192.168.1.1 <valor de HOSTNAME>.midominio.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>.midominio.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 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, los nuevos ficheros que creemos en este directorio contienen lo siguiente. Como ejemplo vamos a crear el fichero ifconfig.eth0 ejecutando:

```
cat > /etc/sysconfig/network-devices/ifconfig.eth0 << "EOF"
ONBOOT=yes
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 que creemos por los valores apropiados para nuestra máquina. Si la variable ONBOOT tiene el valor yes, el guión network activará la interfaz durante el arranque del sistema. Si contiene cualquier otro valor, el guión network ignorará el contenido del archivo y, por lo tanto, no la activará.

Capítulo 8. Hacer el sistema LFS arrancable

Introducción

Este capítulo hará arrancable el sistema LFS. Trataremos la creación de un nuevo fichero fstab, la construcción de un nuevo 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 las particiones, qué sistemas de ficheros 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 estás 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. Te 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.

Instalación de Linux−2.4.22

```
Tiempo estimado de construcción:               Todas las opciones por defecto: 4.20 SBU
Estimación del espacio necesario en disco:     Todas las opciones por defecto: 181 MB
```

Contenido de Linux

El núcleo Linux es el corazón de todo sistema Linux. Es lo que hace a Linux funcionar. Cuando se enciende un ordenador y se inicia un sistema Linux, el núcleo es lo primero que se carga. El núcleo inicializa los componentes hardware del sistema: puertos serie, puertos paralelo, tarjetas de sonido, tarjetas de red, controladores IDE, controladores SCSI y mucho más. En pocas palabras, el núcleo hace que el hardware esté disponible para que el software pueda ejecutarse.

Ficheros instalados: el núcleo y las cabeceras del núcleo.

Dependencias de instalación de Linux

Linux depende de: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Instalación del núcleo

Construir el núcleo comprende tres pasos: configurarlo, compilarlo e instalarlo. Si no te gusta la forma en que se configura el núcleo en este libro, lee el fichero `README` que acompaña al árbol de código fuente del núcleo y busca qué otras opciones existen.

Prepara la compilación ejecutando el siguiente comando:

```
make mrproper
```

Esto asegura que las fuentes del núcleo están completamente limpias. 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 después de desempaquetarlo.

Configura el núcleo mediante una interfaz de menús:

```
make menuconfig
```

Puede que **`make oldconfig`** sea mejor elección 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−2.4.22`. 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 necesitas el 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 `/usr/src/linux-2.4.22/Documentation`. La página de manual de modules.conf y el kernel-CÓMO en http://es.tldp.org/COMO-INSFLUG/COMOs/Kernel-Como/ (el original 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
```

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 algunos de los ficheros creados aún residen en el árbol de las fuentes. Para completar la instalación, dos ficheros deben copiarse al directorio `/boot`.

La ruta al fichero 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
```

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, por ejemplo 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 pueda 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 seguiríá siendo (hd1).

Usando la información anterior, determina la denominación apropiada para tu partición raíz. Para los siguientes ejemplos asumiremos que tu partición raíz 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 el comando sería **`setup (hd0,3)`**.

Ahora indícale 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 5.0
root (hd0,3)
kernel /boot/lfskernel root=/dev/hda4 ro
EOF
```

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 ro
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.

Ahora podría ser un buen momento para quitar todos los símbolos de depuración de los archivos binarios de tu sistema LFS. Si no eres un programador y no planeas depurar tus programas, entonces te alegrará saber que puedes recuperar algunas decenas de megabytes borrando estos símbolos. Este proceso no produce ningún otro inconveniente que no sea no poder depurar los programas nunca más, lo que no es problema si no sabes cómo depurarlos.

Advertencia: El 98% de la gente que usa el comando mencionado más adelante no experimenta ningún problema. Pero haz una copia de seguridad de tu sistema LFS antes de ejecutar este comando. Hay una pequeña posibilidad de que te salga el tiro por la culata, y convierta tu sistema en inutilizable (principalmente destruyendo los módulos del núcleo y las librerías dinámicas y compartidas). Sin embargo, suele ocurrir más a menudo por un error tipográfico que por un problema con el comando utilizado.

Después de haber dicho esto, la opción −−strip−debug que usaremos para quitar los símbolos de depuración es, bajo circunstancias normales, bastante inocua. No borrará nada vital de los ficheros. También es bastante seguro usar −−strip−all con programas normales (no se debe usar en librerías, se destruirían), pero no es tan seguro como el anterior y el espacio que ganas no es tan grande. Pero si andas justo de espacio de disco, cada granito de arena ayuda, así que decide por ti mismo. Por favor, lee la página del manual de strip para ver las opciones que puedes usar. La idea general es no ejecutar strip sobre librerías (usando otra opción que no sea −−strip−debug) para asegurarnos de hacer la apuesta segura.

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, incluido el intérprete de comandos bash activo. Por tanto, necesitarás salir del entorno chroot y reentrar en él utilizando un comando chroot modificado:

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

Ahora ejecuta el siguiente comando:

```
/tools/bin/find /{,usr/,usr/local/}{bin,sbin,lib} -type f \
   -exec /tools/bin/strip --strip-debug '{}' ';'
```

Se avisará de que no se reconoce el formato de un buen número de ficheros. Muchos de ellos son giones en vez de binarios. Puedes ignorar estos avisos.

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 5.0 > /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 salir del entorno chroot y reiniciar el ordenador. Antes de salir del entorno chroot desmonta los sistemas de ficheros virtuales montados ejecutando:

```
umount /proc
umount /dev/pts
```

Sal del entorno chroot:

```
logout
```

Adicionalmente, ahora que todo el software ha sido instalado, ya no es necesario el directorio `/tools`. Puedes borrarlo. Como esto también eliminará las copias temporales de Tcl, Expect y DejaGnu, que se usaron para ejecutar los bancos de pruebas, necesitarás recompilarlos y reinstalarlos en tu sistema LFS si quieres utilizar más tarde estos programas.

Puede que también quieras mover el contenido de `/sources` a `/usr/src/packages` o algo similar (o simplemente borrarlo si lo has quemado en un CD) y borrar el directorio.

Antes de reiniciar, desmonta la propia partición LFS:

```
umount $LFS
```

Si has decidido crear varias particiones, necesitas desmontar las otras particiones antes de desmontar $LFS, por ejemplo:

```
umount $LFS/usr
umount $LFS/home
umount $LFS
```

Y ahora puedes reiniciar el sistema ejecutando algo como:

```
/sbin/shutdown −r now
```

Asumiendo que el gestor de arranque Grub fué configurado como se indicó anteriormente, el menú por defecto debería estar estabecido para que *LFS 5.0* arrancase 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?

Te agradecemos que hayas leído el Libro LFS y 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 cubre los procesos de instalación de una amplia gama de software que está fuera del alcance 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−5.0/.
- Recetas de LFS

  Las recetas de LFS son una serie de pequeños 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 buén 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. Mira el Capítulo 1 − Listas de correo para más información.

  La comunidad hispanoparlante puede recurrir a la lista de correo linux−desde−cero. Esta lista de correo no pertenece oficialmente al proyecto LFS, pero igualmente encontrarás gente dispuesta a ayudarte.

  Si estás interesado en colaborar en la traducción al castellano de los diversos documentos del LFS, tienes a tu disposición la lista de correo lfs−es.
- 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. El sitio principal se encuentra en http://www.tldp.org/, y la sección en castellano en http://es.tldp.org.

IV. Parte IV − Apéndices

Índice

Apéndice A. Descripción de los paquetes y sus dependencias

Introducción

En este apéndice se describen los siguiente aspectos de cada paquete instalado en este libro.

- la localización oficial para la descarga del paquete.
- el contenido de cada paquete.
- lo que cada programa de dicho paquete hace.
- lo que cada paquete necesita para poder compilarlo.

Mucha de la información sobre estos paquetes (especialmente, su descripción) se ha extraído de las páginas del manual de esos paquetes. No incluimos las páginas del manual completas, sólo los elementos clave que hagan posible entender lo que cada programa hace. Para conseguir información detallada de un programa, dirijete a su página de manual o a su página info.

Ciertos paquetes están documentados con mayor profundidad que otros, sencillamente porque sabemos más sobre unos que sobre otros. Si algo debería ser añadido a las siguientes descripciones, por favor no dudes en comunicarlo en las listas de correo. Intentamos que la lista contenga una descripción detallada de cada paquete, pero no podemos hacerlo sin ayuda.

Ten en cuenta que actualmente sólo está descrito lo que hace un paquete , y no lo que necesita que esté instalado. Esto se añadirá más adelante.

También están listadas todas las dependencias para la instalación de todos los paquetes instalados en el libro. La lista incluye qué programas de qué paquetes son necesarios para compilar correctamente el paquete a instalar.

Estas no son las dependencias necesarias para su ejecución, por lo tanto no te ayudarán para saber qué programas son necesarios para usar los programas del paquete. Son solamente las dependencias necesarias para compilarlo.

La lista de dependencias puede estar en ocasiones anticuada con respecto a la versión del paquete usada actualmente. Comprobar las dependencias es un trabajo pesado por lo que puede haber un desfase en la actualización de los paquetes. Pero, normalmente, en la actualización de versiones menores del paquete, las dependencias de instalación no cambian, por lo que son actuales en muchos casos. Cuando actualizamos a una versión mayor, nos aseguramos de hacer también un chequeo de las dependencias.

Autoconf

Las instrucciones de instalación están en la sección *Instalación de Autoconf−2.57* del Capítulo 6.

Localización oficial para descarga

Autoconf (2.57):
ftp://ftp.gnu.org/gnu/autoconf/

Contenido de Autoconf

Autoconf produce guiones del intérprete de comandos que configuran automáticamente el código fuente.

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.

Dependencias de instalación de Autoconf

Autoconf depende de: Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Automake

Las instrucciones de instalación están en la sección *Instalación de Automake−1.7.6* del Capítulo 6.

Localización oficial para descarga

Automake (1.7.6):
ftp://ftp.gnu.org/gnu/automake/

Contenido de Automake

Automake genera ficheros Makefile.in, pensados para usar con Autoconf.

Programas instalados: acinstall, aclocal, aclocal−1.7, automake, automake−1.7, compile, config.guess, config.sub, depcomp, elisp−comp, install−sh, mdate−sh, missing, mkinstalldirs, py−compile, 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 automaticamente 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` automaticamente 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 codigo 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.

ylwrap es un envoltorio para lex y yacc.

Dependencias de instalación de Automake

Automake depende de: Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Bash

Las instrucciones de instalación están en la sección *Instalación de Bash−2.05b* del Capítulo 6.

Localización oficial para descarga

Bash (2.05b):
ftp://ftp.gnu.org/gnu/bash/

Parche para Bash:
http://www.linuxfromscratch.org/patches/lfs/5.0/bash−2.05b−2.patch

Contenido de Bash

bash es la "Bourne−Again SHell", que es un completo intérprete de comandos usado ampliamente en sistemas Unix. El programa bash lee de la entrada estándar (el teclado). Un usuario escribe algo y el programa evalúa lo que ha escrito y hace algo con ello, como lanzar un programa.

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

Descripciones cortas

bash es un intérprete de comandos ampliamente usado. Realiza todo tipo 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.

Dependencias de instalación de Bash

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

Binutils

Las instrucciones de instalación están en la sección *Instalación de Binutils−2.14* del Capítulo 6.

Localización oficial para descarga

Binutils (2.14):
ftp://ftp.gnu.org/gnu/binutils/

Contenido de Binutils

Binutils es una colección de herramientas para el desarrollo de software que contiene un enlazador, un ensamblador y otras utilidades para trabajar con ficheros de objetos y archivos.

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 esplora el fichero al completo.

strip elimina símbolos de ficheros objeto.

libiberty contiene rutinas usadas por varios programas GNU, icluidos 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.

Dependencias de instalación de Binutils

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

Bison

Las instrucciones de instalación están en la sección *Instalación de Bison−1.875* del Capítulo 6.

Localización oficial para descarga

Bison (1.875):
ftp://ftp.gnu.org/gnu/bison/

Parche Attribute para Bison:
http://www.linuxfromscratch.org/patches/lfs/5.0/bison−1.875−attribute.patch

Contenido de Bison

Bison es un generador de analizadores sintácticos, un sustituto de yacc. Bison genera un programa que analiza la estructura de un fichero de texto.

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 util, pero POSIX la solicita.

Dependencias de instalación de Bison

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

Bzip2

Las instrucciones de instalación están en la sección *Instalación de Bzip2−1.0.2* del Capítulo 6.

Localización oficial para descarga

Bzip2 (1.0.2):
http://sources.redhat.com/bzip2/

Contenido de Bzip2

Bzip2 es un compresor de ficheros por ordenación de bloques que, generalmente, consige una mejor compresión que el tradicional **gzip**.

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.

Dependencias de instalación de Bzip2

Bzip2 depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make.

Coreutils

Las instrucciones de instalación están en la sección *Instalación de Coreutils−5.0* del Capítulo 6.

Localización oficial de descarga

Coreutils (5.0):
ftp://ftp.gnu.org/gnu/coreutils/

Parche Hostname para Coreutils:
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−hostname−2.patch

Parche Uname para Coreutils:
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−uname.patch

Contenido de Coreutils

El paquete Coreutils contiene un completo conjunto de utilidades básicas para el intérprete de comandos.

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, kill, 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 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, separandolos 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 identicos.

kill termina un proceso especificado.

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 alfabeticamente.

md5sum muestra o verifica sumas de comprobación MD5.

mkdir crea directorios con los nombres indicados.

mkfifo crea tuberías (FIFO) 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, separandolas 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 dificil 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.

Dependencias de instalación de Coreutils

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

DejaGnu

Las instrucciones de instalación están en la sección *Instalación de DejaGnu−1.4.3* del Capítulo 5.

Localización oficial para descarga

DejaGnu (1.4.3):
ftp://ftp.gnu.org/gnu/dejagnu/

Contenido de DejaGnu

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

Programa instalado: runtest

Descripción corta

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

Dependencias de instalación de DejaGnu

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

Diffutils

Las instrucciones de instalación están en la sección *Instalación de Diffutils−2.8.1* del Capítulo 6.

Localización oficial para descarga

Diffutils (2.8.1):
ftp://ftp.gnu.org/gnu/diffutils/

Contenido de Diffutils

Los programas de este paquete te muestran las diferencias entre dos ficheros o directorios. Es muy común usarlos para crear parches de software.

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.

Dependencias de instalación de Diffutils

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

E2fsprogs

Las instrucciones de instalación están en la sección *Instalación de E2fsprogs−1.34* del Capítulo 6.

Localización oficial para descarga

E2fsprogs (1.34):
ftp://download.sourceforge.net/pub/sourceforge/e2fsprogs/
http://download.sourceforge.net/e2fsprogs/

Contenido de E2fsprogs

E2fsprogs proporciona las utilidades para los sistemas de ficheros ext2. También soporta los sistemas de ficheros ext3 con registro de transaciones.

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.

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.

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 identificador universal único (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.

Dependencias de instalación de E2fsprogs

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

Ed

Las instrucciones de instalación están en la sección *Instalación de Ed−0.2* del Capítulo 6.

Localización oficial para descarga

Ed (0.2):
ftp://ftp.gnu.org/gnu/ed/

Parche Mkstemp para Ed:
http://www.linuxfromscratch.org/patches/lfs/5.0/ed−0.2−mkstemp.patch

Contenido de Ed

GNU ed es un editor de líneas de 8 bits limpio y que cumple con POSIX.

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.

Dependencias de instalación de Ed

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

Expect

Las instrucciones de instalación están en: la sección *Instalación de Expect−5.39.0* del Capítulo 5.

Localización oficial para descarga

Expect (5.39.0):
http://expect.nist.gov/src/

Parche Spawn para Expect:
http://www.linuxfromscratch.org/patches/lfs/5.0/expect−5.39.0−spawn.patch

Contenido de Expect

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

Programa instalado: expect

Librería instalada: libexpect5.39.a

Descripción corta

expect "habla" con otros programas interactivos según un guión.

Dependencias de instalación de Expect

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

File

Las instrucciones de instalación están en la sección *Instalación de File−4.04* del Capítulo 6.

Localización oficial para descarga

File (4.04):
ftp://ftp.gw.com/mirrors/pub/unix/file/

Localización alternativa de descarga:
ftp://gaosu.rave.org/pub/linux/lfs/

Contenido de File

File es una utilidad usada para determinar el tipo de los ficheros.

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.

Dependencias de instalación de File

File depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Zlib.

Findutils

Las instrucciones de instalación están en la sección *Instalación de Findutils−4.1.20* del Capítulo 6.

Localización oficial para descarga

Findutils (4.1.20):
ftp://alpha.gnu.org/gnu/findutils/

Contenido de Findutils

El paquete Findutils contiene pogramas para encontrar ficheros, tanto al vuelo (haciendo una búsqueda recursiva en vivo a través de los directorios y mostrando sólo los ficheros que cumplan las especificaciones) o mediante una búsqueda a través de una base de datos.

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.

Dependencias de instalación de Findutils

Findutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Flex

Las instrucciones de instalación están en la sección *Instalación de Flex−2.5.4a* del Capítulo 6.

Localización oficial para descarga

Flex (2.5.4a):
ftp://ftp.gnu.org/non−gnu/flex/

Contenido de Flex

El pquete Flex se utiliza para generar programas que reconocen patrones de texto.

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.

Dependencias de instalación de Flex

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

Gawk

Las instrucciones de instalación están en la sección *Instalación de Gawk−3.1.3* del Capítulo 6.

Localización oficial para descarga

Gawk (3.1.3):
ftp://ftp.gnu.org/pub/gnu/gawk/

Parche Libexecdir para Gawk:
http://www.linuxfromscratch.org/patches/lfs/5.0/gawk−3.1.3−libexecdir.patch

Contenido de Gawk

Gawk es una implementación de awk utilizada para manipular ficheros de texto.

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`.

Dependencias de instalación de Gawk

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

GCC

Las instrucciones de instalación están en la sección *Instalación de GCC−3.3.1* del Capítulo 6.

Localización oficial para descarga

GCC (3.3.1):
ftp://ftp.gnu.org/pub/gnu/gcc/

Parche No−Fixincludes para GCC:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−no_fixincludes−2.patch

Parche Specs para GCC:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−specs−2.patch

Parche que suprime Libiberty para GCC:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−suppress−libiberty.patch

GCC−2 (2.95.3):
ftp://ftp.gnu.org/pub/gnu/gcc/

Parche para GCC−2:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−2.patch

Parche No−Fixincludes para GCC−2:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−no−fixinc.patch

Parche Return−Type para GCC−2:
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−returntype−fix.patch

Contenido de GCC

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

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++.

Dependencias de instalación de GCC

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

Gettext

Las instrucciones de instalación están en la sección *Instalación de Gettext−0.12.1* del Capítulo 6.

Localización oficial para descarga

Gettext (0.12.1):
ftp://ftp.gnu.org/gnu/gettext/

Contenido de Gettext

El paquete Gettext se utiliza para la internacionalización y localización. Los programas pueden compilarse con Soporte de Lenguaje Nativo (NLS), lo que les permite mostrar mensajes en el idioma nativo del usuario.

Programas instalados: autopoint, config.charset, config.rpath, 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 copìa los ficheros estándar de infraestructutura 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 busqueda en tiempo de ejecución de las librerías compartidas en un ejecutable.

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.

Dependencias de instalación de Gettext

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

Glibc

Las instrucciones de instalación están en la sección *Instalación de Glibc−2.3.2* del Capítulo 6.

Localización oficial para descarga

Glibc (2.3.2):
ftp://ftp.gnu.org/gnu/glibc/

Glibc−linuxthreads (2.3.2):
ftp://ftp.gnu.org/gnu/glibc/

Parche Sscanf para Glibc:
http://www.linuxfromscratch.org/patches/lfs/5.0/glibc−2.3.2−sscanf−1.patch

Contenido de Glibc

Glibc es la librería C que proporciona las llamadas al sistema y las funciones básicas, tales como open, malloc, printf, etc. La librería C es utilizada por todos los programas enlazados dinámicamente.

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 acceseo para un pseudoterminal 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, servicos, 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.

Dependencias de instalación de Glibc

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

Grep

Las instrucciones de instalación están en la sección *Instalación de Grep−2.5.1* del Capítulo 6.

Localización oficial para descarga

Grep (2.5.1):
ftp://ftp.gnu.org/gnu/grep/

Contenido de Grep

Grep es un programa usado para imprimir las líneas de un fichero que cumplan un patrón especificado.

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.

Dependencias de instalación de Grep

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

Groff

Las instrucciones de instalación están en la sección *Instalación de Groff−1.19* del Capítulo 6.

Localización oficial para descarga

Groff (1.19):
ftp://ftp.gnu.org/gnu/groff/

Contenido de Groff

El paquete Groff incluye varios programas de procesamiento de texto para formatear el texto. Groff traduce el texto estándar y los comandos especiales a una salida formateada, como la que puedes ver en las páginas de manual.

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 mostrar 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.

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.

Dependencias de instalación de Groff

Groff depende de: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Grub

Las instrucciones de instalación están en la sección *Instalación de Grub−0.93* del Capítulo 6.

Localización oficial para descarga

Grub (0.93):
ftp://alpha.gnu.org/pub/gnu/grub/

Parche Gcc33 para Grub:
http://www.linuxfromscratch.org/patches/lfs/5.0/grub−0.93−gcc33−1.patch

Contenido de Grub

El paquete Grub contiene un cargador de arranque.

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.

Dependencias de instalación de Grub

Grub depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Gzip

Las instrucciones de instalación están en la sección *Instalación de Gzip−1.3.5* del Capítulo 6.

Localización oficial para descarga

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

Contenido de Gzip

El paquete Gzip contiene programas para comprimir y descomprimir ficheros usando el codificador Lempel−Ziv (LZ77).

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 estandar 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).

Dependencias de instalación de Gzip

Gzip depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Inetutils

Las instrucciones de instalación están en la sección *Instalación de Inetutils−1.4.2* del Capítulo 6.

Localización oficial para descarga

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

Contenido de Inetutils

El paquete Inetutils contiene clientes y servidores de red.

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.

Dependencias de instalación de Inetutils

Inetutils depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Kbd

Las instrucciones de instalación están en la sección *Instalación de Kbd−1.08* del Capítulo 6.

Localización oficial para descarga

Kbd (1.08):
ftp://ftp.win.tue.nl/pub/linux−local/utils/kbd/

Parche para Kbd:
http://www.linuxfromscratch.org/patches/lfs/5.0/kbd−1.08−more−programs.patch

Contenido de Kbd

Kbd contiene ficheros de mapas de teclado y utilidades para el teclado.

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 (NumLock) 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.

Dependencias de instalación de Kbd

Kbd depende de: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, Gzip, M4, Make, Sed.

Less

Las instrucciones de instalación están en la sección *Instalación de Less−381* del Capítulo 6.

Localización oficial para descarga

Less (381):
ftp://ftp.gnu.org/gnu/less/

Contenido de Less

Less es un paginador de ficheros, o visor de texto. Muestra el contenido de un fichero, o de un flujo de datos, y tiene la habilidad de poder recorrerlo. Less tiene várias características no incluidas en el paginador **`more`**, como la habilidad de recorrer los ficheros hacia atrás.

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.

Dependencias de instalación de Less

Less depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

LFS−Bootscripts

Las instrucciones de instalación están en la sección *Instalación de LFS−Bootscripts−1.12* del Capítulo 6.

Localización oficial para descarga

LFS−Bootscripts (1.12):
http://downloads.linuxfromscratch.org/

Contenido de LFS−bootscripts

El paquete LFS−Bootscripts contiene guiones del interprete de comandos para el inicio del sistema al estilo SysV. Estos guiones realizan varias tareas como comprobar la integridad del sistema de ficheros durante el arranque, cargar el mapa del teclado, establecer la red y parar los procesos durante el cierre del sistema.

Guiones instalados: checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountproc, network, rc, reboot, sendsignals, setclock, 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 **mountproc** se usa para montar el sistema de ficheros 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 específica.

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 **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.

Dependencias de instalación de LFS−Bootscripts

LFS−Bootscripts depende de: Bash, Coreutils.

Lfs−Utils

Las instrucciones de instalación están en la sección *Instalación de Lfs−Utils−0.3* del Capítulo 6.

Localización oficial para descarga

Lfs−utils (0.3):
http://www.linuxfromscratch.org/~winkie/downloads/lfs−utils/

Contenido de Lfs−Utils

El paquete Lfs−Utils contiene varios programas usados por otros paquetes, pero que no son lo suficientemente grandes como para proveerlos en paquetes individuales.

Programas instalados: mktemp, tempfile, http−get e iana−net

Ficheros instalados: protocols y services

Descripciones cortas

mktemp crea ficheros temporales de forma segura para usarlos en guiones.

tempfile crea ficheros temporales de una forma no tan segura como lo hace **`mktemp`**. Lo instalamos para mantener compatibilidad hacia atrás.

http−get es un guión que aprovecha una capacidad poco conocida de **`bash`** llamada "redirección en red". Sirve para hacer descargas desde sitios web sin necesidad de usar otros programas.

iana−net usa **`http−get`** para simplificar el proceso de obtener los ficheros de configuración de servicios y protocolos de IANA.

Dependencias de instalación de Lfs−Utils

(Aún no se comprobaron las dependencias.)

Libtool

Las instrucciones de instalación están en la sección *Instalación de Libtool−1.5* del Capítulo 6.

Localización oficial para descarga

Libtool (1.5):
ftp://ftp.gnu.org/gnu/libtool/

Contenido de Libtool

GNU libtool es un guión para soporte genérico de librerías. Libtool oculta la complejidad del uso de librerías compartidas tras una interfaz consistente y portable.

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.

Dependencias de instalación de Libtool

Libtool depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Linux (el núcleo)

Las instrucciones de instalación están en la sección *Instalación de Linux−2.4.22* del Capítulo 8.

Localización oficial para descarga

Linux (2.4.22):
ftp://ftp.kernel.org/pub/linux/kernel/

Contenido de Linux

El núcleo Linux es el corazón de todo sistema Linux. Es lo que hace a Linux funcionar. Cuando se enciende un ordenador y se inicia un sistema Linux, el núcleo es lo primero que se carga. El núcleo inicializa los componentes hardware del sistema: puertos serie, puertos paralelo, tarjetas de sonido, tarjetas de red, controladores IDE, controladores SCSI y mucho más. En pocas palabras, el núcleo hace que el hardware esté disponible para que el software pueda ejecutarse.

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, el núcleo es la primera cosa de tu sistema operativo que se carga. 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.

Dependencias de instalación de Linux

Linux depende de: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

M4

Las instrucciones de instalación están en la sección *Instalación de M4−1.4* del Capítulo 6.

Localización oficial para descarga

M4 (1.4):
ftp://ftp.gnu.org/gnu/m4/

Contenido de M4

M4 es un procesador de macros. Copia la entrada a la salida expandiendo las macros en el proceso. Las macros pueden ser internas o definidas por el usuario y pueden tomar cualquier número de argumentos. Aparte de hacer la expansión de macros, m4 tiene funciones internas para la inclusión de 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.

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.

Dependencias de instalación de M4

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

Make

Las instrucciones de instalación están en la sección *Instalación de Make−3.80* del Capítulo 6.

Localización oficial para descarga

Make (3.80):
ftp://ftp.gnu.org/gnu/make/

Contenido de Make

Make determina, automáticamente, qué piezas de un programa largo es necesario recompilar y ejecuta los comandos para recompilarlas.

Programa instalado: make

Descripción corta

make determina, automáticamente, qué partes de un paquete grande necesitan ser recompiladas y lanza los comandos para hacerlo.

Dependencias de instalación de Make

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

MAKEDEV

Las instrucciones de instalación están en la sección *Creación de los dispositivos (Makedev−1.7)* del Capítulo 6.

Localización oficial para descarga

MAKEDEV (1.7):
http://downloads.linuxfromscratch.org/

Contenido de MAKEDEV

MAKEDEV es un guión que crea los ficheros de dispositivos estáticos necesarios, que usualmente residen en el directorio `/dev`. Puede encontrarse más información sobre los ficheros de dispositivos dentro de las fuentes del núcleo en `Documentation/devices.txt`.

Guión instalado: MAKEDEV

Descripción corta

MAKEDEV es un guión que crea los ficheros de dispositivos estáticos necesarios, que usualmente residen en el directorio `/dev`.

Dependencias de instalación de MAKEDEV

Make depende de: Bash, Coreutils.

Man

Las instrucciones de instalación están en la sección *Instalación de Man−1.5m2* del Capítulo 6.

Localización oficial para descarga

Man (1.5m2):
ftp://ftp.win.tue.nl/pub/linux−local/utils/man/

Parche 80−Columns para Man:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−80cols.patch

Parche Manpath para Man:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−manpath.patch

Parche Pager para Man:
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−pager.patch

Contenido de Man

Man es un paginador de manuales.

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.

Dependencias de instalación de Man

Man depende de: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Man−pages

Las instrucciones de instalación están en la sección *Instalación de Man−pages−1.60* del Capítulo 6.

Localización oficial para descarga

Man−pages (1.60):
ftp://ftp.kernel.org/pub/linux/docs/manpages/

Contenido de Man−pages

El paquete Man−pages contiene alrededor de 1200 páginas de manual. Esta documentación detalla las funciones de C y C++, describe varios ficheros de dispositivo importantes y proporciona documentación procedente de otros paquetes que posiblemente falte en los mismos.

Ficheros instalados: diversas páginas de manual.

Descripción corta

(Última versión comprobada: 1.60)

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.

Dependencias de instalación de Man−pages

Man depende de: Bash, Coreutils, Make.

Modutils

Las instrucciones de instalación están en la sección *Instalación de Modutils−2.4.25* del Capítulo 6.

Localización oficial para descarga

Modutils (2.4.25):
ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils/

Contenido de Modutils

El paque Modutils contiene programas que puedes utilizar para trabajar con los módulos del núcleo.

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.

Dependencias de instalación de Modutils

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

Ncurses

Las instrucciones de instalación están en la sección *Instalación de Ncurses−5.3* del Capítulo 6.

Localización oficial para descarga

Ncurses (5.3):
ftp://ftp.gnu.org/gnu/ncurses/

Parche Etip para Ncurses:
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−etip−2.patch

Parche Vsscanf para Ncurses:
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−vsscanf.patch

Contenido de Ncurses

El paquete Ncurses proporciona librerías para el manejo de caractéres y terminales, incluidos paneles y menús.

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 requqerido 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* contiene funciones para implementar formularios.

libmenu* contiene funciones para implementar menús.

libpanel* contiene funciones para implementar paneles.

Dependencias de instalación de Ncurses

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

Net−tools

Las instrucciones de instalación están en la sección *Instalación de Net−tools−1.60* del Capítulo 6.

Localización oficial para descarga

Net−tools (1.60):
http://www.tazenda.demon.co.uk/phil/net−tools/

Parche Mii−Tool−Gcc33 para Net−Tools:
http://www.linuxfromscratch.org/patches/lfs/5.0/net−tools−1.60−miitool−gcc33−1.patch

Contenido de Net−tools

El paquete Net−tools contiene una colección de programas que forman la base del trabajo en red en Linux.

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 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.

Dependencias de instalación de Net−tools

Net−tools depende de: Bash, Binutils, Coreutils, GCC, Glibc, Make.

Patch

Las instrucciones de instalación están en la sección *Instalación de Patch−2.5.4* del Capítulo 6.

Localización oficial para descarga

Patch (2.5.4):
ftp://ftp.gnu.org/gnu/patch/

Contenido de Patch

El programa patch modifica un fichero basandose en un parche. Normalmente un parche es una lista, creada por el programa diff, que contiene instrucciones sobre cómo debe modificarse el fichero original.

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.

Dependencias de instalación de Patch

Patch depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Perl

Las instrucciones de instalación están en la sección *Instalación de Perl−5.8.0* del Capítulo 6.

Localización oficial para descarga

Perl (5.8.0):
http://www.perl.com/

Parche Libc para Perl:
http://www.linuxfromscratch.org/patches/lfs/5.0/perl−5.8.0−libc−3.patch

Contenido de Perl

El paquete Perl contiene perl, el Lenguaje Práctico de Extracción e Informe. Perl combina alguna de las mejores características de C, sed, awk y sh dentro de un poderoso lenguaje.

Programas instalados: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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.

en2cxsconstruse una extensión Perl para el módulo Encode, a partir de cualquier Mapa de Caractéres 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 caractéres **`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.

Dependencias de instalación de Perl

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

Procinfo

Las instrucciones de instalación están en la sección *Instalación de Procinfo−18* del Capítulo 6.

Localización oficial para descarga

Procinfo (18):
ftp://ftp.cistron.nl/pub/people/svm/

Contenido de Procinfo

El programa procinfo obtiene datos del sistema, como el uso de la memoria y los números de las interrupciones (IRQ), a partir del directorio `/proc`, y formatea estos datos de una forma atractiva.

Programas instalados: lsdev, procinfo y socklist

Descripciones cortas

lsdev lista los dispositivos presentes en tu sistema y que IRQs 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.

Dependencias de instalación de Procinfo

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

Procps

Las instrucciones de instalación están en la sección *Instalación de Procps−3.1.11* del Capítulo 6.

Localización oficial para descarga

Procps (3.1.11):
http://procps.sourceforge.net/

Parche Locale para Procps:
http://www.linuxfromscratch.org/patches/lfs/5.0/procps−3.1.11−locale−fix.patch

Contenido de Procps

El paquete Procps proporciona programas para supervisar y parar procesos del sistema. Procps obtiene la información sobre los procesos a través del directorio `/proc`.

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.

Dependencias de instalación de Procps

Procps depende de: Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses.

Psmisc

Las instrucciones de instalación están en la sección *Instalación de Psmisc−21.3* del Capítulo 6.

Localización oficial para descarga

Psmisc (21.3):
http://download.sourceforge.net/psmisc/
ftp://download.sourceforge.net/pub/sourceforge/psmisc/

Contenido de Psmisc

El paquete Psmisc contiene tres programas que ayudan en el manejo del directorio `/proc`.

Programas instalados: fuser, killall y 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.

pidof muestra los números de identificación de los programas especificados. (Sin embargo, no usamos este programa pidof sino el de Sysvinit.

pstree muestra los procesos en ejecución en forma de árbol.

Dependencias de instalación de Psmisc

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

Sed

Las instrucciones de instalación están en la sección *Instalación de Sed−4.0.7* del Capítulo 6.

Localización oficial para descarga

Sed (4.0.7):
ftp://ftp.gnu.org/gnu/sed/

Contenido de Sed

sed es un editor de flujo. Un editor de flujo se utiliza para realizar transformaciones básicas de texto sobre un flujo de entrada (un fichero o la entrada procedente de una tubería).

Programa instalado: sed

Descripción corta

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

Dependencias de instalación de Sed

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

Shadow

Las instrucciones de instalación están en la sección *Instalación de Shadow−4.0.3* del Capítulo 6.

Localización oficial para descarga

Shadow (4.0.3):
ftp://ftp.pld.org.pl/software/shadow/

Parche Newgrp para Shadow:
http://www.linuxfromscratch.org/patches/lfs/5.0/shadow−4.0.3−newgrp−fix.patch

Contenido de Shadow

El paquete Shadow se creó para consolidar la seguridad del sistema de contraseñas.

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 agegar 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.

Dependencias de instalación de Shadow

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

Sysklogd

Las instrucciones de instalación están en la sección *Instalación de Sysklogd−1.4.1* del Capítulo 6.

Localización oficial para descarga

Sysklogd (1.4.1):
http://www.infodrom.org/projects/sysklogd/

Contenido de Sysklogd

El paquete Sysklogd contiene programas para grabar tanto los mensajes del sistema como los generados por el núcleo

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.

Dependencias de instalación de Sysklogd

Sysklogd depende de: Binutils, Coreutils, GCC, Glibc, Make.

Sysvinit

Las instrucciones de instalación están en la sección *Instalación de Sysvinit−2.85* del Capítulo 6.

Localización oficial para descarga

Sysvinit (2.85):
ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/

Contenido de Sysvinit

El paquete Sysvinit contiene programas para controlar el arranque, ejecución y descarga de todos los demás programas.

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.

Dependencias de instalación de Sysvinit

Sysvinit depende de: Binutils, Coreutils, GCC, Glibc, Make.

Tar

Las instrucciones de instalación están en la sección *Instalación de Tar−1.13.25* del Capítulo 6.

Localización oficial para descarga

Tar (1.13.25):
ftp://alpha.gnu.org/gnu/tar/

Contenido de Tar

Tar es un programa de archivado diseñado para almacenar y extraer ficheros en un archivo conocido como fichero 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).

Dependencias de instalación de Tar

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

Tcl

Las instrucciones de instalación están en: la sección *Instalación de Tcl−8.4.4* del Capítulo 5.

Localización oficial para descarga

Tcl (8.4.4):
http://download.sourceforge.net/tcl/
ftp://download.sourceforge.net/pub/sourceforge/tcl/

Contenido de Tcl

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

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.

Dependencias de instalación de Tcl

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

Texinfo

Las instrucciones de instalación están en la sección *Instalación de Texinfo−4.6* del Capítulo 6.

Localización oficial para descarga

Texinfo (4.6):
ftp://ftp.gnu.org/gnu/texinfo/

Contenido de Texinfo

El paquete Texinfo contiene programas usados para leer, escribir y convertir documentos Info, que suministran documentación del sistema.

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.

Dependencias de instalación de Texinfo

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

Util−linux

Las instrucciones de instalación están en la sección *Instalación de Util−linux−2.12* del Capítulo 6.

Localización oficial para descarga

Util−linux (2.12):
http://ftp.cwi.nl/aeb/util−linux/

Contenido de Util−linux

El paquete Util−linux contiene una miscelánea de utilidades. Algunas de las utilidades más destacables son las utilizadas para montar, desmontar, formatear, particionar y manejar dispositivos de disco, abrir puertos de consola o capturar los mensajes del núcleo.

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, kill, 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 colmnas indicadas.

column formatea un fichero a multiples 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 o reloj BIOS).

ipcrm elimina el recurso IPC especificado.

ipcs facilita información sobre el estado IPC.

isosize muestra el tamaño de un sistema de ficheros iso9660.

kill termina los procesos especificados.

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.

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 del árbol de ficheros del sistema .

namei muestra los enlaces simbólicos en la ruta de nombres indicada.

pg Muestra un fichero de texto a pantalla completa.

pivot_root hace que el sistema de ficheros indicado sea el raiz del proceso actual.

ramsize puede usarse 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 puede usarse para establecer las opciones de partición raiz 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 puede usarse 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. Esto es, si ese usuario no ha desactivado dichos mensajes.

Dependencias de instalación de Util−linux

Util−linux depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Zlib.

Vim

Las instrucciones de instalación están en la sección *Instalación de Vim−6.2* del Capítulo 6.

Localización oficial para descarga

Vim (6.2):
ftp://ftp.vim.org/pub/editors/vim/unix/

Contenido de Vim

El paquete Vim contiene un editor de texto configurable construido para obtener una eficiente edición del texto.

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 leido 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.

vi arranca vim en modo compatible con vi.

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 básios de vim.

xxd genera un volcado hexadecimal. También puede hacer lo contrario, por lo que puede usarse para parchear binarios.

Dependencias de instalación de Vim

Vim depende de: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Zlib

Las instrucciones de instalación están en la sección *Instalación de Zlib−1.1.4* del Capítulo 6.

Localización oficial para descarga

Zlib (1.1.4):
http://www.gzip.org/zlib/

Parche Zlib Vsnprintf:
http://www.linuxfromscratch.org/patches/lfs/5.0/zlib−1.1.4−vsnprintf.patch

Contenido de Zlib

El paquete Zlib contiene la librería libz, utilizada por varios programas para realizar las funciones de

compresión y descompresión..

Librería instalada: libz[a,so]

Descripción corta

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

Dependencias de instalación de Zlib

Zlib depende de: Binutils, Coreutils, GCC, Glibc, Make, Sed.

Apéndice B. Índice de programas y librerías

Este es un listado con todos los programas y librerías instalados en este libro, cada uno con un enlace a la página del Apéncice A del paquete al que pertenece.

- diff3 : Diffutils
- dir : Coreutils
- dircolors : Coreutils
- dirname : Coreutils
- dmesg : Util−linux
- dnsdomainname : Net−tools
- domainname : Net−tools
- dpasswd : Shadow
- dprofpp : Perl
- du : Coreutils
- dumpe2fs : E2fsprogs
- dumpkeys : Kbd
- e2fsck : E2fsprogs
- e2image : E2fsprogs
- e2label : E2fsprogs
- echo : Coreutils
- ed : Ed
- efm_filter.pl : Vim
- efm_perl.pl : Vim
- egrep : Grep
- elisp−comp : Automake
- elvtune : Util−linux
- env : Coreutils
- enc2xs : Perl
- eqn : Groff
- e2n2graph : Groff
- ex : Vim
- expand : Coreutils
- expiry : Shadow
- expr : Coreutils
- factor : Coreutils
- faillog : Shadow
- false : Coreutils
- fdformat : Util−linux
- fdisk : Util−linux
- fgconsole : Kbd
- fgrep : Grep
- file : File
- find : Findutils
- find2perl : Perl
- findfs : E2fsprogs
- flex : Flex
- flex++ : Flex
- fmt : Coreutils
- fold : Coreutils
- frcode : Findutils
- free : Procps
- fsck : E2fsprogs
- fsck.cramfs : Util−linux
- fsck.ext2 : E2fsprogs
- fsck.ext3 : E2fsprogs

- objcopy : Binutils
- objdump : Binutils
- od : Coreutils
- oldps : Procps
- openvt : Kbd
- parse.bash : Util−linux
- parse.tcsh : Util−linux
- passwd : Shadow
- paste : Coreutils
- patch : Patch
- pathchk : Coreutils
- pcprofiledump : Glibc
- perl : Perl
- perlbug : Perl
- perlcc : Perl
- perldoc : Perl
- perlivp : Perl
- pfbtops : Groff
- pg : Util−linux
- pgawk : Gawk
- pgrep : Procps
- pic : Groff
- pic2graph : Groff
- piconv : Perl
- pidof : Sysvinit
- ping : Inetutils
- pinky : Coreutils
- pivot_root : Util−linux
- pkill : Procps
- pl2pm : Perl
- plipconfig : Net−tools
- pltags.pl : Vim
- pmap : Procps
- pod2html : Perl
- pod2latex : Perl
- pod2man : Perl
- pod2text : Perl
- pod2usage : Perl
- podchecker : Perl
- podselect : Perl
- post−grohtml : Groff
- poweroff : Sysvinit
- pr : Coreutils
- pre−grohtml : Groff
- printenv : Coreutils
- printf : Coreutils
- procinfo : Procinfo
- project−id : Gettext
- ps : Procps
- psed : Perl
- psfaddtable : Kbd

- sed : Sed
- sendsignals : LFS−Bootscripts
- seq : Coreutils
- setclock : LFS−Bootscripts
- setfdprm : Util−linux
- setfont : Kbd
- setkeycodes : Kbd
- setleds : Kbd
- setlogcons : Kbd
- setmetamode : Kbd
- setsid : Util−linux
- setterm : Util−linux
- setvesablank : Kbd
- sfdisk : Util−linux
- sg : Shadow
- sh : Bash
- sha1sum : Coreutils
- showconsolefont : Kbd
- showkey : Kbd
- shred : Coreutils
- shtags.pl : Vim
- shutdown : Sysvinit
- size : Binutils
- skill : Procps
- slattach : Net−tools
- sleep : Coreutils
- sln : Glibc
- snice : Procps
- socklist : Procinfo
- soelim : Groff
- sort : Coreutils
- splain : Perl
- split : Coreutils
- sprof : Glibc
- stat : Coreutils
- strings : Binutils
- strip : Binutils
- stty : Coreutils
- su : Coreutils
- sulogin : Sysvinit
- sum : Coreutils
- swap : LFS−Bootscripts
- swapoff : Util−linux
- swapon : Util−linux
- sync : Coreutils
- sysctl : Procps
- sysklogd : LFS−Bootscripts
- syslogd : Sysklogd
- tac : Coreutils
- tack : Ncurses
- tail : Coreutils