Linux From Scratch

Version 6.0

Gerard Beekmans

Linux From Scratch: Version 6.0

by Gerard Beekmans

Table of Contents

Preface

1. Foreword

My adventures in Linux began six years ago when I downloaded and installed my first distribution. After working with it for awhile, I discovered issues I definitely would have liked to see improved upon. For example, I didn't like the arrangement of the bootscripts or the way programs were configured by default. I tried a number of alternate distributions to address these issues, yet each had its pros and cons. Finally, I realized that if I wanted full satisfaction from my Linux system, I would have to build my own from scratch.

What does this mean? I resolved not to use pre-compiled packages of any kind, nor CD-ROMs or boot disks that would install basic utilities. I would use my current Linux system to develop my own customized system. This “perfect” Linux system would then have the strengths of various systems without their associated weaknesses. In the beginning, the idea was rather daunting, but I remained committed to the idea that a system could be built that would conform to my needs and desires rather than to a standard that just did not fit what I was looking for.

After sorting through issues such as circular dependencies and compile-time errors, I created a custom-built Linux system that was fully operational and suitable to individual needs. This process also allowed me to create compact and streamlined Linux systems which are faster and take up less space than traditional operating systems. I called this system a Linux From Scratch system, or an LFS system for short.

As I shared my goals and experiences with other members of the Linux community, it became apparent that there was sustained interest in the ideas set forth in my Linux adventures. Such custom-built LFS systems not only to meet user specifications and requirements, but also serve as an ideal learning opportunity for programmers and system administrators to enhance their Linux skills. Out of this broadened interest, the Linux From Scratch Project was born.

This *Linux From Scratch* book provides readers with the background and instruction to design and build custom Linux systems. This book highlights the Linux from Scratch project and the benefits of using this system. Users can dictate all aspects of their system, including directory layout, script setup, and security. The resulting system will be compiled straight from the source code, and the user will be able to specify where, why, and how programs are installed. This book allows readers to fully customize Linux systems to their own needs and allows users more control over their system.

I hope you will have a great time working on your own LFS system, and enjoy the numerous benefits of having a system that is truly *your own*.

--
Gerard Beekmans
gerard@linuxfromscratch.org

2. Audience

There are many reasons why somebody would want to read this book. The principle reason is to install a Linux system straight from the source code. A question many people raise is, “why go through all the hassle of manually building a Linux system from scratch when you can just download and install an existing one?” That is a good question and is the impetus for this section of the book.

One important reason for LFS's existence is to help people learn how a Linux system works from the inside out. Building an LFS system helps demonstrate what makes Linux tick, and how things work together and depend on each other. One of the best things that this learning experience provides is the ability to customize Linux to your own tastes and needs.

A key benefit of LFS is that it allows users to have more control over the system without relying on someone else's Linux implementation. With LFS, *you* are in the driver's seat and dictate every aspect of the system, such as the directory layout and bootscript setup. You also dictate where, why, and how programs are installed.

Another benefit of LFS is the ability to create a very compact Linux system. When installing a regular distribution, one is often forced to install several programs which are probably never used. These programs waste precious disk space, or worse, CPU cycles. It is not difficult to build an LFS system of less than 100 megabytes (MB), which is substantially smaller compared to most existing setups. Does this still sound like a lot of space? A few of us have been working on creating a very small embedded LFS system. We successfully built a system that was specialized to run the Apache web server with approximately 8MB of disk space used. Further stripping could bring this down to 5 MB or less. Try that with a regular distribution! This is only one of the many benefits of designing your own Linux implementation.

We could compare Linux distributions to a hamburger purchased at a fast-food restaurant—you have no idea what might be in what you are eating. LFS, on the other hand, does not give you a hamburger. Rather, LFS provides the recipe to make the exact hamburger desired. This allows users to review the recipe, omit unwanted ingredients, and add your own ingredients to enhance the flavor of the burger. When you are satisfied with the recipe, move on to preparing it. It can be made to exact specifications—broil it, bake it, deep-fry it, or barbecue it.

Another analogy that we can use is that of comparing LFS with a finished house. LFS provides the skeletal plan of a house, but it is up to you to build it. LFS maintains the freedom to adjust plans throughout the process, customizing it to the user's needs and preferences.

An additional advantage of a custom built Linux system is security. By compiling the entire system from source code, you are empowered to audit everything and apply all the security patches desired. It is no longer necessary to wait for somebody else to compile binary packages that fix a security hole. Unless you examine the patch and implement it yourself, you have no guarantee that the new binary package was built correctly and adequately fixes the problem.

The goal of Linux From Scratch is to build a complete and usable foundation-level system. Readers who do not wish to build their own Linux system from scratch may not benefit from the information in this book. If you only want to know what happens while the computer boots, we recommend the “From Power Up To Bash Prompt” HOWTO located at *http://axiom.anu.edu.au/~okeefe/p2b/* or on The Linux Documentation Project's (TLDP) website at *http://www.tldp.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html*. The HOWTO builds a system which is similar to that of this book, but it focuses strictly on creating a system capable of booting to a BASH prompt. Consider your objective. If you wish to build a Linux system while learning along the way, then this book is your best choice.

There are too many good reasons to build your own LFS system to list them all here. This section is only the tip of the iceberg. As you continue in your LFS experience, you will find the power that information and knowledge truly bring.

3. Prerequisites

This book assumes that the reader has a reasonable knowledge of using and installing Linux software. Before building an LFS system, we recommend reading the following HOWTOs:

- Software-Building-HOWTO
  http://www.tldp.org/HOWTO/Software-Building-HOWTO.html

  This is a comprehensive guide to building and installing “generic” Unix software distributions under Linux.

- The Linux Users' Guide
  http://espc22.murdoch.edu.au/~stewart/guide/guide.html

  This guide covers the usage of assorted Linux software.

- The Essential Pre-Reading Hint
  http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt

  This is an LFS Hint written specifically for users new to Linux. It includes a list of links to excellent sources of information on a wide range of topics. Anyone attempting to install LFS should have an understanding of many of the topics in this hint.

4. Typography

To make things easier to follow, there are a few typographical conventions used throughout this book. This section contains some examples of the typographical format found throughout Linux From Scratch.

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

This form of text is designed to be typed exactly as seen unless otherwise noted in the surrounding text. It is also used in the explanation sections to identify which of the commands is being referenced.

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

This form of text (fixed width text) shows screen output, probably as the result of commands issued. This format is also used to show filenames, such as `/etc/ld.so.conf`.

Emphasis

This form of text is used for several purposes in the book, mainly to emphasize important points or items.

http://www.linuxfromscratch.org/

This format is used for hyperlinks, both within the LFS community and to external pages, including HOWTOs, download locations, and websites.

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

This format is used when creating configuration files. The first command tells the system to create the file `$LFS/etc/group` from whatever is typed on the following lines until the sequence end of file (EOF) is encountered. Therefore, this entire section is generally typed as seen.

`[REPLACED TEXT]`

This format is used to encapsulate text that is not to be typed as seen.

5. Structure

This book is divided into the following parts.

5.1. Part I - Introduction

Part I explains a few important notes on how to proceed with the LFS installation. This section also provides meta-information about the book.

5.2. Part II - Preparing for the Build

Part II describes how to prepare for the building process—making a partition, downloading the packages, and compiling temporary tools.

5.3. Part III - Building the LFS System

Part III guides the reader through the building of the LFS system—compiling and installing all the packages one by one, setting up the boot scripts, and installing the kernel. The resulting Linux system is the foundation on which other software can be built to expand the system as desired. At the end of this book, there is an easy to use reference listing all of the programs, libraries, and important files that have been installed.

Part I. Introduction

Chapter 1. Introduction

1.1. How to Build an LFS System

The LFS system will be built by using a previously installed Linux distribution (such as Debian, Mandrake, Red Hat, or SuSE). This existing Linux system (the host) will be used as a starting point to provide necessary programs, including a compiler, linker, and shell, to build the new system. Select the “development” option during the distribution installation to be able to access these tools.

Chapter 2 of this book describes how to create a new Linux native partition and file system, the place where the new LFS system will be compiled and installed. Chapter 3 explains which packages and patches need to be downloaded to build an LFS system and how to store them on the new file system. Chapter 4 discusses the setup for an appropriate work environment. Please read Chapter 4 carefully as it explains several important issues the developer should be aware of before beginning to work through Chapter 5 and beyond.

Chapter 5 explains the installation of a number of packages that will form the basic development suite (or toolchain) which is used to build the actual system in Chapter 6. Some of these packages are needed to resolve circular dependencies—for example, to compile a compiler, you need a compiler.

Chapter 5 also shows the user how to build a first pass of the toolchain, including Binutils and GCC (first pass basically means these two core packages will be re-installed a second time). The programs from these packages will be linked statically in order to be used independently of the host system. The next step is to build Glibc, the C library. Glibc will be compiled by the toolchain programs built in the first pass. Then, a second pass of the toolchain will be built. This time, the toolchain will be dynamically linked against the newly built Glibc. The remaining Chapter 5 packages are built using this second pass toolchain. When this is done, the LFS installation process will no longer depend on the host distribution, with the exception of the running kernel.

While this may initially seem like a lot of work to get away from a host distribution, a full technical explanation is provided at the beginning of Chapter 5, including notes on the differences between statically and dynamically-linked programs.

In Chapter 6, the full LFS system is built. The chroot (change root) program is used to enter a virtual environment and start a new shell whose root directory will be set to the LFS partition. This is very similar to rebooting and instructing the kernel to mount the LFS partition as the root partition. The system does not actually reboot, but instead chroots because creating a bootable system requires additional work which is not necessary just yet. The major advantage is that “chrooting” allows the builder to continue using the host while LFS is being built. While waiting for package compilation to complete, a user can switch to a different virtual console (VC) or X desktop and continue using the computer as normal.

To finish the installation, the bootscripts are set up in Chapter 7, and the kernel and boot loader are set up in Chapter 8. Chapter 9 contains information on furthering the LFS experience beyond this book. After the steps in this book have been implemented, the computer will be ready to reboot into the new LFS system.

This is the process in a nutshell. Detailed information on each step is discussed in the following chapters and package descriptions. Items that may seem complicated will be clarified, and everything will fall into place as the developer embarks on the LFS adventure.

1.2. Resources

1.2.1. FAQ

If during the building of the LFS system you encounter any errors, have any questions, or think there is a typo in the book, please start by consulting the Frequently Asked Questions (FAQ) at *http://www.linuxfromscratch.org/faq/*.

1.2.2. Mailing Lists

The `linuxfromscratch.org` server hosts a number of mailing lists used for the development of the LFS project. These lists include the main development and support lists, among others.

For information on the different lists, how to subscribe, archive locations, and additional information, visit *http://www.linuxfromscratch.org/mail.html*.

1.2.3. IRC

Several members of the LFS community offer assistance on our community Internet Relay Chat (IRC) network. Before using this support, please make sure that your question is not already answered in the LFS FAQ or the mailing list archives. You can find the IRC network at `irc.linuxfromscratch.org` or `irc.linux-phreak.net`. The support channel is named #LFS-support.

1.2.4. News Server

The mailing lists hosted at `linuxfromscratch.org` are also accessible via the Network News Transfer Protocol (NNTP) server. All messages posted to a mailing list are copied to the corresponding newsgroup, and vice versa.

The news server is located at `news.linuxfromscratch.org`.

1.2.5. Wiki

For more information on packages, updated versions, tweaks, and personal experiences, see the LFS Wiki at *http://wiki.linuxfromscratch.org/*. Users can also add information there to help others with their future LFS activities.

1.2.6. References

For additional information on the packages, useful tips are available at *http://www.linuxfromscratch.org/~matthew/LFS-references.html*.

1.2.7. Mirror Sites

The LFS project has a number of world-wide mirrors to make accessing the website and downloading the required packages more convenient. Please visit the LFS website at *http://www.linuxfromscratch.org/* for a list of current mirrors.

1.2.8. Contact Information

Please direct all your questions and comments to one of the LFS mailing lists (see above).

1.3. Help

If an issue or a question is encountered while working through this book, check the FAQ page at *http://www.linuxfromscratch.org/faq/#generalfaq*. Questions are often already answered there. If your question is not answered on this page, try to find the source of the problem. The following hint will give you some guidance for troubleshooting: *http://www.linuxfromscratch.org/hints/downloads/files/errors.txt*.

We also have a wonderful LFS community that is willing to offer assistance through IRC and the mailing lists (see the Section 1.2, “Resources” section of this book). In order to assist with diagnosing and solving the problem, please include all relevant information in your request for help.

1.3.1. Things to Mention

Apart from a brief explanation of the problem being experienced, the essential things to include in any request for help are:

- The version of the book being used (in this case 6.0)
- The host distribution and version being used to create LFS
- The package or section the problem was encountered in
- The exact error message or symptom being received
- Note whether you have deviated from the book at all

> **Note**
>
> Deviating from this book does *not* mean that we will not help you. After all, LFS is about personal preference. Being upfront about any changes to the established procedure helps us evaluate and determine possible causes of your problem.

1.3.2. Configure Problems

If something goes wrong during the stage where the configure script is run, review the `config.log` file. This file may contain errors encountered during configure which were not printed to the screen. Include those relevant lines if you need to ask for help.

1.3.3. Compile Problems

Both the screen output and the contents of various files are useful in determining the cause of compile issues. The screen output from the **./configure** script and the **make** run can be helpful. It is not necessary to include the entire output, but do include enough of the relevant information. Below is an example of the type of information to include from the screen output from **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
```

In this case, many people would just include the bottom section:

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

This is not enough information to properly diagnose the problem because it only notes that something went wrong, not *what* went wrong. The entire section, as in the example above, is what should be saved because it includes the command that was executed and the associated error message(s).

An excellent article about asking for help on the Internet is available online at *http://catb.org/~esr/faqs/smart-questions.html*. Read and follow the hints in this document to increase the likelihood of getting the help you need.

1.3.4. Test Suite Problems

Many packages provide a test suite which, depending on the importance of the package, should be run. Sometimes packages will generate false or expected failures. If these errors are encountered, check the LFS Wiki page at *http://wiki.linuxfromscratch.org/* to see if we have noted and investigated these issues. If these issues are noted and addressed, there is no need to be concerned.

1.4. About the Included CD

For your convenience, we have included a CD with this book that contains the source packages needed for creating a Linux From Scratch system. The CD is bootable and provides a stable working environment for building LFS. This book refers to this system as the “host system.”

In addition to the tools required to build LFS, the host system on the CD has a number of other helpful tools installed:

- An HTML version of this book
- The X Window System Environment
- Web Tools
  - Wget (command line file retriever)
  - Lynx (text web browser)
  - Irssi (console IRC client)
  - Firefox (graphical web browser)
  - Xchat (X-based IRC client)
- Text Editors
  - Vim
  - Nano
- Network Tools
  - SSH Server and Client
  - NFS Server and Client
  - Smbmount (mount.cifs) for Windows shares
  - Subversion
  - Dhcpcd (DHCP client)

- Filesystem Programs
    - Reiserfsprogs
    - Xfsprogs
- nALFS - A tool for automating LFS builds

Chapter 2. Preparing a New Partition

2.1. Introduction

In this chapter, the partition which will host the LFS system is prepared. We will create the partition itself, create a file system on it, and mount it.

2.2. Creating a New Partition

In order to build a new Linux system, space is required in the form of an empty disk partition. If the computer does not have a free partition or room on any of the hard disks to make one, LFS can be built on the same partition where the current distribution is installed.

Note

This advanced procedure is not recommended for your first LFS installation, but if you are short on disk space the following document can be helpful: *http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing_systems.txt*.

A minimal system requires a partition of around 1.3 gigabytes (GB). This is enough to store all the source tarballs and compile the packages. However, if the LFS system is intended to be the primary Linux system, additional software will probably be installed which will require additional space (2 or 3 GB). The LFS system itself will not take up this much space. A large portion of this required amount of space is to provide sufficient free temporary space. Compiling packages can require a lot of disk space which will be reclaimed after the package is installed.

Because there is not always enough Random Access Memory (RAM) available for compilation processes, it is a good idea to use a small disk partition as swap space. This space is used by the kernel to store seldom-used data to make room in memory for active processes. The swap partition for an LFS system can be the same as the one used by the host system, so another swap partition will not need to be created if your host system already has one setup.

Start a disk partitioning program such as **cfdisk** or **fdisk** with a command line option naming the hard disk on which the new partition will be created—for example `/dev/hda` for the primary Integrated Drive Electronics (IDE) disk. Create a Linux native partition and a swap partition, if needed. Please refer to the man pages of **cfdisk** or **fdisk** if you do not yet know how to use the programs.

Remember the designation of the new partition (e.g., `hda5`). This book will refer to this as the LFS partition. Also remember the designation of the swap partition. These names will be needed later for the `/etc/fstab` file.

2.3. Creating a File System on the Partition

Now that a blank partition has been set up, the file system can be created. The most widely-used system in the Linux world is the second extended file system (ext2), but with the newer high-capacity hard disks, the journaling file systems are becoming increasingly popular. Here we will create an ext2 file system, but build instructions for other file systems can be found at *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/filesystems.html*.

To create an ext2 file system on the LFS partition, run the following:

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

Replace *[xxx]* with the name of the LFS partition (`hda5` in our previous example).

If a swap partition was created, it will need to be initialized as a swap partition too (also known as formatting, as described above with **mke2fs**) by running the following. If you are using an existing swap partition, there is no need to format it.

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

Replace *[yyy]* with the name of the swap partition.

2.4. Mounting the New Partition

Now that a file system has been created, the partition needs to be made accessible. In order to do this, the partition needs to be mounted at a chosen mount point. For the purposes of this book, it is assumed that the file system is mounted under `/mnt/lfs`, but the directory choice is up to you.

Choose a mount point and assign it to the `LFS` environment variable by running:

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

Next, create the mount point and mount the LFS file system by running:

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

Replace *[xxx]* with the designation of the LFS partition.

If using multiple partitions for LFS (e.g., one for `/` and another for `/usr`), mount them using:

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

Replace *[xxx]* and *[yyy]* with the appropriate partition names.

Ensure that this new partition is not mounted with permissions that are too restrictive (such as the nosuid, nodev, or noatime options). Run the **mount** command without any parameters to see what options are set for the mounted LFS partition. If *nosuid*, *nodev*, and/or *noatime* are set, the partition will need to be remounted.

Now that there is an established place to work, it is time to download the packages.

Part II. Preparing for the Build

Chapter 3. Packages and Patches

3.1. Introduction

This chapter includes a list of packages that need to be downloaded for building a basic Linux system. The listed version numbers correspond to versions of the software that are known to work, and this book is based on their use. We highly recommend not using newer versions because the build commands for one version may not work with a newer version. The newest package versions may also have problems that work-arounds have not been developed for yet.

All the URLs, when possible, refer to the package's information page at *http://www.freshmeat.net/*. The Freshmeat pages provide easy access to official download sites, as well as project websites, mailing lists, FAQ, changelogs, and more.

Download locations may not always be accessible. If a download location has changed since this book was published, Google (*http://www.google.com*) provides a useful search engine for most packages. If this search is unsuccessful, try one of the alternate means of downloading discussed at *http://www.linuxfromscratch.org/lfs/packages.html*.

Downloaded packages and patches will need to be stored somewhere that is conveniently available throughout the entire build. A working directory is also required to unpack the sources and build them. `$LFS/sources` can be used both as the place to store the tarballs and patches and as a working directory. By using this directory, the required elements will be located on the LFS partition and will be available during all stages of the building process.

To create this directory, execute, as user *root*, the following command before starting the download session:

```
mkdir $LFS/sources
```

Make this directory writable and sticky. “Sticky” means that even if multiple users have write permission on a directory, only the owner of a file can delete the file within a sticky directory. The following command will enable the write and sticky modes:

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

3.2. All Packages

Download or otherwise obtain the following packages:

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

Note

File (4.10) may no longer be available at the listed location. The site administrators of the master download location occasionally remove older versions when new ones are released. An alternate download location that may have the correct version available is *ftp://ftp.linuxfromscratch.org/pub/lfs/*.

- Findutils (4.1.20) - 760 KB:
  http://freshmeat.net/projects/findutils/
- Flex (2.5.31) - 372 KB:
  http://freshmeat.net/projects/flex/
- Gawk (3.1.4) - 1,692 KB:
  http://freshmeat.net/projects/gnuawk/
- GCC (3.4.1) - 27,000 KB:
  http://freshmeat.net/projects/gcc/
- Gettext (0.14.1) - 6,397 KB:
  http://freshmeat.net/projects/gettext/
- Glibc (2.3.4-20040701) - 13,101 KB:
  http://freshmeat.net/projects/glibc/

Note

Released packages of Glibc are not new enough for our purposes, so create a tarball of an appropriate Concurrent Versions System (CVS) snapshot with the following commands:

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

> Alternatively, the LFS team developed a tarball which can be downloaded from any of the File Transfer Protocol (FTP) mirrors listed on the LFS Website at *http://www.linuxfromscratch.org/lfs/packages.html#http*. It is located under the `/pub/lfs/packages/conglomeration/glibc` directory. The tarball is signed using GNU Privacy Guard (GPG), and it is strongly recommended that its authenticity be verified before use. Instructions for installing GPG, which enables verification, are provided in the Beyond Linux From Scratch (BLFS) book at *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/gnupg.html*.

- Grep (2.5.1) - 545 KB:
  http://freshmeat.net/projects/grep/
- Groff (1.19.1) - 2,360 KB:
  http://freshmeat.net/projects/groff/
- Grub (0.95) - 902 KB:
  ftp://alpha.gnu.org/pub/gnu/grub/
- Gzip (1.3.5) - 324 KB:
  ftp://alpha.gnu.org/gnu/gzip/
- Iana-Etc (1.01) - 161 KB:
  http://freshmeat.net/projects/iana-etc/
- Inetutils (1.4.2) - 1,019 KB:
  http://freshmeat.net/projects/inetutils/
- IPRoute2 (2.6.8-040823) - 264 KB:
  http://developer.osdl.org/dev/iproute2/download/
- Kbd (1.12) - 617 KB:
  http://freshmeat.net/projects/kbd/
- Less (382) - 259 KB:
  http://freshmeat.net/projects/less/
- LFS-Bootscripts (2.2.2) - 16 KB:
  http://downloads.linuxfromscratch.org/
- Libtool (1.5.8) - 2,602 KB:
  http://freshmeat.net/projects/libtool/

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

- Shadow (4.0.4.1) - 795 KB:
  http://freshmeat.net/projects/shadow/
- Sysklogd (1.4.1) - 80 KB:
  http://freshmeat.net/projects/sysklogd/
- Sysvinit (2.85) - 91 KB:
  http://freshmeat.net/projects/sysvinit/
- Tar (1.14) - 1,025 KB:
  http://freshmeat.net/projects/tar/
- Tcl (8.4.7) - 3,363 KB:
  http://freshmeat.net/projects/tcltk/
- Texinfo (4.7) - 1,385 KB:
  http://freshmeat.net/projects/texinfo/
- Udev (030) - 374 KB:
  ftp://ftp.kernel.org/pub/linux/utils/kernel/hotplug/
- Udev Permissions Configuration - 2 KB:
  http://downloads.linuxfromscratch.org/udev-config-2.permissions
- Udev Rules Configuration - 1 KB:
  http://downloads.linuxfromscratch.org/udev-config-1.rules
- Util-linux (2.12b) - 1,921 KB:
  http://freshmeat.net/projects/util-linux/
- Vim (6.3) - 3,612 KB:
  http://freshmeat.net/projects/vim/
- Vim (6.3) language files (optional) - 1,033 KB:
  http://freshmeat.net/projects/vim/
- Zlib (1.2.1) - 277 KB:
  http://freshmeat.net/projects/zlib/

Total size of these packages: 135 MB

3.3. Needed Patches

In addition to the packages, several patches are also required. These patches correct any mistakes in the packages that should be fixed by the maintainer. The patches also make small modifications to make the packages easier to work with. The following patches will be needed to build an LFS system:

- Bash Display Wrap Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/bash-3.0-display_wrap-1.patch
- Coreutils Suppress Uptime, Kill, Su Patch - 16 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/co /
- Coreutils Uname Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/coreutils-5.2.1-uname-2.patch
- Expect Spawn Patch - 6 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/expect-5.42.1-spawn-1.patch
- Flex Brokenness Patch - 8 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/flex-2.5.31-debian_fixes-2.patch
- GCC Linkonce Patch - 12 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-linkonce-1.patch
- GCC No-Fixincludes Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-no_fixincludes-1.patch
- GCC Specs Patch - 11 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/gcc-3.4.1-specs-1.patch
- Inetutils Kernel Headers Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/inetutils-1.4.2-kernel_headers- 1.patch
- Inetutils No-Server-Man-Pages Patch - 4 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/in /

- IPRoute2 Disable DB Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/iproute2-2.6.8_040823-remove_db-1.patch
- Man 80-Columns Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/man-1.5o-80cols-1.patch
- Mktemp Tempfile Patch - 3 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/mktemp-1.5-add_tempfile-1.patch
- Perl Libc Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/perl-5.8.5-libc-1.patch
- Readline Display Wrap Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/readline-5.0-display_wrap-1.patch
- Sysklogd Kernel Headers Patch - 3 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysklogd-1.4.1-kernel_headers- 1.patch
- Sysklogd Signal Handling Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysklogd-1.4.1-signal-1.patch
- Sysvinit /proc Title Length Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/sysvinit-2.85-proclen-1.patch
- Texinfo Segfault Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/texinfo-4.7-segfault-1.patch
- Util-Linux Sfdisk Patch - 1 KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/util-linux-2.12b-sfdisk-2.patch
- Zlib Security Patch - 1KB:
  http://www.linuxfromscratch.org/patches/lfs/6.0/zlib-1.2.1-security-1.patch

In addition to the above required patches, there exist a number of optional patches created by the LFS community. These optional patches solve minor problems or enable functionality that is not enabled by default. Feel free to peruse the patches database located at *http://www.linuxfromscratch.org/patches/* and acquire any additional patches to suit the system needs.

Chapter 4. Final Preparations

4.1. About $LFS

Throughout this book, the environment variable `LFS` will be used several times. It is paramount that this variable is always defined. It should be set to the mount point chosen for the LFS partition. Check that the `LFS` variable is set up properly with:

```
echo $LFS
```

Make sure the output shows the path to the LFS partition's mount point, which is `/mnt/lfs` if the provided example was followed. If the output is incorrect, the variable can be set with:

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

Having this variable set is beneficial in that commands such as **mkdir $LFS/tools** can be typed literally. The shell will automatically replace “$LFS” with “/mnt/lfs” (or whatever the variable was set to) when it processes the command line.

Do not forget to check that `$LFS` is set whenever you leave and reenter the current working environment (as when doing a “su” to *root* or another user).

4.2. Creating the $LFS/tools Directory

All programs compiled in Chapter 5 will be installed under `$LFS/tools` to keep them separate from the programs compiled in Chapter 6. The programs compiled here are temporary tools and will not be a part of the final LFS system. By keeping these programs in a separate directory, they can easily be discarded later after their use. This also prevents these programs from ending up in the host production directories (easy to do by accident in Chapter 5).

Create the required directory by running the following as *root*:

```
mkdir $LFS/tools
```

The next step is to create a `/tools` symlink on the host system. This will point to the newly-created directory on the LFS partition. Run this command as *root* as well:

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

Note

The above command is correct. The **ln** command has a few syntactic variations, so be sure to check the info and man pages before reporting what you may think is an error.

The created symlink enables the toolchain to be compiled so that it always refers to `/tools`, meaning that the compiler, assembler, and linker will work both in this chapter (when we are still using some tools from the host) and in the next (when we are "chrooted" to the LFS partition).

4.3. Adding the LFS User

When logged in as user *root*, making a single mistake can damage or destroy a system. Therefore, we recommend building the packages in this chapter as an unprivileged user. You could use your own user name, but to make it easier to set up a clean work environment, create a new user called *lfs* as a member of a new group (also named *lfs*) and use this user during the installation process. As *root*, issue the following commands to add the new user:

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

The meaning of the command line options:

-s /bin/bash
: This makes **bash** the default shell for user *lfs*.

-g lfs
: This option adds user *lfs* to group *lfs*.

-m
: This creates a home directory for *lfs*.

-k /dev/null
: This parameter prevents possible copying of files from a skeleton directory (default is `/etc/skel`) by changing the input location to the special null device.

lfs
: This is the actual name for the created group and user.

To log in as *lfs* (as opposed to switching to user *lfs* when logged in as *root*, which does not require the *lfs* user to have a password), give *lfs* a password:

```
passwd lfs
```

Grant *lfs* full access to `$LFS/tools` by making *lfs* the directory owner:

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

If a separate working directory was created as suggested, give user *lfs* ownership of this directory:

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

Next, login as user *lfs*. This can be done via a virtual console, through a display manager, or with the following substitute user command:

```
su - lfs
```

The “`-`” instructs **su** to start a login shell as opposed to a non-login shell. The difference between these two types of shells can be found in detail in the Bash man and info pages.

4.4. Setting Up the Environment

Set up a good working environment by creating two new startup files for the **bash** shell. While logged in as user *lfs*, issue the following command to create a new `.bash_profile`:

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

When logged on as user *lfs*, the initial shell is usually a *login* shell which reads the `/etc/profile` of the host (probably containing some settings and environment variables) and then `.bash_profile`. The **exec env -i.../bin/bash** command in the `.bash_profile` file replaces the running shell with a new one with a completely empty environment, except for the `HOME`, `TERM`, and `PS1` variables. This ensures that no unwanted and potentially hazardous environment variables from the host system leak into the build environment. The technique used here achieves the goal of ensuring a clean environment.

The new instance of the shell is a *non-login* shell, which does not read the `/etc/profile` or `.bash_profile` files, but rather reads the `.bashrc` file instead. Create the `.bashrc` file now:

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

The **set +h** command turns off **bash**'s hash function. Hashing is ordinarily a useful feature—**bash** uses a hash table to remember the full path of executable files to avoid searching the `PATH` time and again to find the same executable. However, the new tools should be used as soon as they are installed. By switching off the hash function, the shell will always search the `PATH` when a program is to be run. As such, the shell will find the newly compiled tools in `$LFS/tools` as soon as they are available without remembering a previous version of the same program in a different location.

Setting the user file-creation mask (umask) to 022 ensures that newly created files and directories are only writable by their owner, but are readable and executable by anyone (assuming default modes are used by the open(2) system call, new files will end up with permission mode 644 and directories with mode 755).

The `LFS` variable should be set to the chosen mount point.

The `LC_ALL` variable controls the localization of certain programs, making their messages follow the conventions of a specified country. If the host system uses a version of Glibc older than 2.2.4, having `LC_ALL` set to something other than “POSIX” or “C” (during this chapter) may cause issues if you exit the chroot environment and wish to return later. Setting `LC_ALL` to “POSIX” or “C” (the two are equivalent) ensures that everything will work as expected in the chroot environment.

By putting `/tools/bin` ahead of the standard `PATH`, all the programs installed in Chapter 5 are picked up by the shell immediately after their installation. This, combined with turning off hashing, limits the risk that old programs from the host are being used when they should not be used any longer.

Finally, to have the environment fully prepared for building the temporary tools, source the just-created user profile:

```
source ~/.bash_profile
```

4.5. About SBUs

Many people would like to know beforehand approximately how long it takes to compile and install each package. Because Linux From Scratch can be built on many different systems, it is impossible to provide accurate time estimates. The biggest package (Glibc) will take approximately 20 minutes on the fastest systems, but could take up to three days on slower systems! Instead of providing actual times, the Static Build Unit (SBU) measure will be used instead.

The SBU measure works as follows. The first package to be compiled from this book is the statically-linked Binutils in Chapter 5. The time it takes to compile this package is what will be referred to as the Static Build Unit or SBU. All other compile times will be expressed relative to this time.

For example, consider a package whose compilation time is 4.5 SBUs. This means that if a system took 10 minutes to compile and install the static Binutils, it will take *approximately* 45 minutes to build this example package. Fortunately, most build times are shorter than the one for Binutils.

Please note that if the system compiler on the host is GCC-2.x based, the SBUs listed may be somewhat understated. This is because the SBU is based on the very first package, compiled with the old GCC, while the rest of the system is compiled with the newer GCC-3.4.1 (which is known to be approximately 30 percent slower). SBUs are also not highly accurate for Symmetric Multi-Processor (SMP)-based machines.

To view actual timings for a number of specific machines, we recommend *http://www.linuxfromscratch.org/~bdubbs/*.

In general, SBUs are not very accurate because they depend on many factors, not just the GCC version. They are provided here to give an estimate of how long it might take to install a package, but the numbers can vary by as much as dozens of minutes in some cases.

4.6. About the Test Suites

Most packages provide a test suite. Running the test suite for a newly built package is a good idea because it can provide a “sanity check” indicating that everything compiled correctly. A test suite that passes its set of checks usually proves that the package is functioning as the developer intended. It does not, however, guarantee that the package is totally bug free.

Some test suites are more important than others. For example, the test suites for the core toolchain packages—GCC, Binutils, and Glibc—are of the utmost importance due to their central role in a properly functioning system. The test suites for GCC and Glibc can take a very long time to complete, especially on slower hardware, but are strongly recommended.

> **Note**
>
> Experience has shown that there is little to be gained from running the test suites in Chapter 5. There can be no escaping the fact that the host system always exerts some influence on the tests in that chapter, often causing inexplicable failures. Because the tools built in Chapter 5 are temporary and eventually discarded, we do not recommend running the test suites in Chapter 5 for the average reader. The instructions for running those test suites are provided for the benefit of testers and developers, but they are strictly optional.

A common issue with running the test suites for Binutils and GCC is running out of pseudo terminals (PTYs). This can result in a high number of failing tests. This may happen for several reasons, but the most likely cause is that the host system does not have the `devpts` file system set up correctly. This issue is discussed in greater detail in Chapter 5.

Sometimes package test suites will give false failures. Consult the LFS Wiki at *http://wiki.linuxfromscratch.org/* to verify that these failures are expected. This site is valid for all tests throughout this book.

Chapter 5. Constructing a Temporary System

5.1. Introduction

This chapter shows how to compile and install a minimal Linux system. This system will contain just enough tools to start constructing the final LFS system in Chapter 6 and allow a working environment with more user convenience than a minimum environment would.

There are two steps in building this minimal system. The first step is to build a new and host-independent toolchain (compiler, assembler, linker, libraries, and a few useful utilities). The second step uses this toolchain to build the other essential tools.

The files compiled in this chapter will be installed under the `$LFS/tools` directory to keep them separate from the files installed in the next chapter and the host production directories. Since the packages compiled here are temporary, we do not want them to pollute the soon-to-be LFS system.

Before issuing the build instructions for a package, the package should be unpacked as user *lfs*, and a **cd** into the created directory should be performed. The build instructions assume that the **bash** shell is in use.

Several of the packages are patched before compilation, but only when the patch is needed to circumvent a problem. A patch is often needed in both this and the next chapter, but sometimes in only one or the other. Therefore, do not be concerned if instructions for a downloaded patch seem to be missing. Warning messages about *offset* or *fuzz* may also be encountered when applying a patch. Do not worry about these warnings, as the patch was still successfully applied.

During the compilation of most packages, there will be several warnings that scroll by on the screen. These are normal and can safely be ignored. These warnings are as they appear—warnings about deprecated, but not invalid, use of the C or C++ syntax. C standards change fairly often, and some packages still use the older standard. This is not a problem, but does prompt the warning.

After installing each package, delete its source and build directories, unless specifically instructed otherwise. Deleting the sources saves space and prevents mis-configuration when the same package is reinstalled later. Only three of the packages need to retain the source and build directories in order for their contents to be used by later commands. Pay special attention to these reminders.

Check one last time that the `LFS` environment variable is set up properly:

```
echo $LFS
```

Make sure the output shows the path to the LFS partition's mount point, which is `/mnt/lfs`, using our example.

5.2. Host System Requirements

The host must be running at least a 2.6.2 kernel compiled with GCC-3.0 or higher. There are two main reasons for this high requirement. First, the Native POSIX Threading Library (NPTL) test suite will segfault if the host's kernel has not been compiled with GCC-3.0 or a later version. Secondly, the 2.6.2 or later version of the kernel is required for the use of Udev. Udev creates devices dynamically by reading from the `sysfs` file system. However, support for this filesystem has only recently been implemented in most of the kernel drivers. We must be sure that all critical system devices get created properly.

In order to determine whether the host kernel meets the requirements outlined above, run the following command:

```
cat /proc/version
```

This will produce output similar to:

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

If the results of the above command state that the host kernel was not compiled using a GCC-3.0 (or later) compiler, one will need to be compiled. The host system will then need to be rebooted to use the newly compiled kernel. Instructions for compiling the kernel and configuring the boot loader (assuming the host uses GRUB) are located in Chapter 8.

5.3. Toolchain Technical Notes

This section explains some of the rationale and technical details behind the overall build method. It is not essential to immediately understand everything in this section. Most of this information will be clearer after performing an actual build. This section can be referred back to at any time during the process.

The overall goal of Chapter 5 is to provide a temporary environment that can be chrooted into and from which can be produced a clean, trouble-free build of the target LFS system in Chapter 6. Along the way, we separate from the host system as much as possible, and in doing so, build a self-contained and self-hosted toolchain. It should be noted that the build process has been designed to minimize the risks for new readers and provide maximum educational value at the same time. In other words, more advanced techniques could be used to build the system.

> **Important**
>
> Before continuing, be aware of the name of the working platform, often referred to as the target triplet. Many times, the target triplet will probably be *i686-pc-linux-gnu*. A simple way to determine the name of the target triplet is to run the **config.guess** script that comes with the source for many packages. Unpack the Binutils sources and run the script: **`./config.guess`** and note the output.
>
> Also be aware of the name of the platform's dynamic linker, often referred to as the dynamic loader (not to be confused with the standard linker **ld** that is part of Binutils). The dynamic linker provided by Glibc finds and loads the shared libraries needed by a program, prepares the program to run, and then runs it. The name of the dynamic linker will usually be `ld-linux.so.2`. On platforms that are less prevalent, the name might be `ld.so.1`, and newer 64 bit platforms might be named something else entirely. The name of the platform's dynamic linker can be determined by looking in the `/lib` directory on the host system. A sure-fire way to determine the name is to inspect a random binary from the host system by running: **`readelf -l <name of binary> | grep interpreter`** and noting the output. The authoritative reference covering all platforms is in the `shlib-versions` file in the root of the Glibc source tree.

Some key technical points of how the Chapter 5 build method works:

- The process is similar in principle to cross-compiling, whereby tools installed in the same prefix work in cooperation, and thus utilize a little GNU "magic"
- Careful manipulation of the standard linker's library search path ensures programs are linked only against chosen libraries
- Careful manipulation of **gcc**'s `specs` file tell the compiler which target dynamic linker will be used

Binutils is installed first because the **./configure** runs of both GCC and Glibc perform various feature tests on the assembler and linker to determine which software features to enable or disable. This is more important than one might first realize. An incorrectly configured GCC or Glibc can result in a subtly broken toolchain, where the impact of such breakage might not show up until near the end of the build of an entire distribution. A test suite failure will usually alert this error before too much additional work is performed.

Binutils installs its assembler and linker in two locations, `/tools/bin` and `/tools/$TARGET_TRIPLET/bin`. The tools in one location are hard linked to the other. An important facet of the linker is its library search order. Detailed information can be obtained from **ld** by passing it the `--verbose` flag. For example, an **`ld --verbose | grep SEARCH`** will illustrate the current search paths and their order. It shows which files are linked by **ld** by compiling a dummy program and passing the `--verbose` switch to the linker. For example, **`gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded`** will show all the files successfully opened during the linking.

The next package installed is GCC. An example of what can be seen during its run of **./configure** is:

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

This is important for the reasons mentioned above. It also demonstrates that GCC's configure script does not search the PATH directories to find which tools to use. However, during the actual operation of **gcc** itself, the same search paths are not necessarily used. To find out which standard linker **gcc** will use, run: **`gcc -print-prog-name=ld`**.

Detailed information can be obtained from **gcc** by passing it the `-v` command line option while compiling a dummy program. For example, **`gcc -v dummy.c`** will show detailed information about the preprocessor, compilation, and assembly stages, including **gcc**'s included search paths and their order.

The next package installed is Glibc. The most important considerations for building Glibc are the compiler, binary tools, and kernel headers. The compiler is generally not an issue since Glibc will always use the **gcc** found in a `PATH` directory. The binary tools and kernel headers can be a bit more complicated. Therefore, take no risks and use the available configure switches to enforce the correct selections. After the run of **./configure**, check the contents of the `config.make` file in the `glibc-build` directory for all important details. Note the use of *`CC="gcc -B/tools/bin/"`* to control which binary tools are used and the use of the *`-nostdinc`* and *`-isystem`* flags to control the compiler's include search path. These items highlight an important aspect of the Glibc package—it is very self-sufficient in terms of its build machinery and generally does not rely on toolchain defaults.

After the Glibc installation, make some adjustments to ensure that searching and linking take place only within the `/tools` prefix. Install an adjusted **ld**, which has a hard-wired search path limited to `/tools/lib`. Then amend **gcc**'s specs file to point to the new dynamic linker in `/tools/lib`. This last step is vital to the whole process. As mentioned above, a hard-wired path to a dynamic linker is embedded into every Executable and Link Format (ELF)-shared executable. This can be inspected by running: **`readelf -l <name of binary> | grep interpreter`**. Amending gcc's specs file ensures that every program compiled from here through the end of this chapter will use the new dynamic linker in `/tools/lib`.

The need to use the new dynamic linker is also the reason why the Specs patch is applied for the second pass of GCC. Failure to do so will result in the GCC programs themselves having the name of the dynamic linker from the host system's `/lib` directory embedded into them, which would defeat the goal of getting away from the host.

During the second pass of Binutils, we are able to utilize the *`--with-lib-path`* configure switch to control **ld**'s library search path. From this point onwards, the core toolchain is self-contained and self-hosted. The remainder of the Chapter 5 packages all build against the new Glibc in `/tools`.

Upon entering the chroot environment in Chapter 6, the first major package to be installed is Glibc, due to its self-sufficient nature mentioned above. Once this Glibc is installed into `/usr`, perform a quick changeover of the toolchain defaults, then proceed in building the rest of the target LFS system.

5.3.1. Notes on Static Linking

Besides their specific task, most programs have to perform many common and sometimes trivial operations. These include allocating memory, searching directories, reading and writing files, string handling, pattern matching, arithmetic, and other tasks. Instead of obliging each program to reinvent the wheel, the GNU system provides all these basic functions in ready-made libraries. The major library on any Linux system is Glibc.

There are two primary ways of linking the functions from a library to a program that uses them—statically or dynamically. When a program is linked statically, the code of the used functions is included in the executable, resulting in a rather bulky program. When a program is dynamically linked, it includes a reference to the dynamic linker, the name of the library, and the name of the function, resulting in a much smaller executable. A third option is to use the programming interface of the dynamic linker (see the *dlopen* man page for more information).

Dynamic linking is the default on Linux and has three major advantages over static linking. First, only one copy of the executable library code is needed on the hard disk, instead of having multiple copies of the same code included in several programs, thus saving disk space. Second, when several programs use the same library function at the same time, only one copy of the function's code is required in core, thus saving memory space. Third, when a library function gets a bug fixed or is otherwise improved, only the one library needs to be recompiled instead of recompiling all programs that make use of the improved function.

If dynamic linking has several advantages, why then do we statically link the first two packages in this chapter? The reasons are threefold—historical, educational, and technical. The historical reason is that earlier versions of LFS statically linked every program in this chapter. Educationally, knowing the difference between static and dynamic linking is useful. The technical benefit is a gained element of independence from the host, meaning that those programs can be used independently of the host system. However, it is worth noting that an overall successful LFS build can still be achieved when the first two packages are built dynamically.

5.4. Binutils-2.15.91.0.2 - Pass 1

The Binutils package contains a linker, an assembler, and other tools for handling object files.

Approximate build time: 1.0 SBU
Required disk space: 194 MB

Binutils installation depends on: Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed, and Texinfo

5.4.1. Installation of Binutils

It is important that Binutils be the first package compiled because both Glibc and GCC perform various tests on the available linker and assembler to determine which of their own features to enable.

This package is known to have issues when its default optimization flags (including the *`-march`* and *`-mcpu`* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building Binutils.

The Binutils documentation recommends building Binutils outside of the source directory in a dedicated build directory:

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

Note

In order for the SBU values listed in the rest of the book to be of any use, measure the time it takes to build this package from the configuration, up to and including the first install. To achieve this easily, wrap the four commands in a **time** command like this: **`time { ./configure ... && ... && ... && make install; }`**.

Now prepare Binutils for compilation:

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

The meaning of the configure options:

--prefix=/tools
: This tells the configure script to prepare to install the Binutils programs in the `/tools` directory.

--disable-nls
: This disables internationalization. This is not needed for the static programs, and NLS can cause problems when linking statically.

Continue with compiling the package:

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

The meaning of the make parameters:

configure-host
: This forces all subdirectories to be configured immediately. A statically-linked build will fail without it. Use this option to work around the problem.

LDFLAGS="-all-static"
: This tells the linker that all Binutils programs should be linked statically. However, strictly speaking, *"-all-static"* is passed to the **libtool** program, which then passes *"-static"* to the linker.

Compilation is now complete. Ordinarily we would now run the test suite, but at this early stage the test suite framework (Tcl, Expect, and DejaGNU) is not yet in place. The benefits of running the tests at this point are minimal since the programs from this first pass will soon be replaced by those from the second.

Install the package:

```
make install
```

Next, prepare the linker for the “Adjusting” phase later on:

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

The meaning of the make parameters:

-C ld clean
: This tells the make program to remove all compiled files in the `ld` subdirectory.

-C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib
: This option rebuilds everything in the `ld` subdirectory. Specifying the `LIB_PATH` Makefile variable on the command line allows us to override the default value and point it to the temporary tools location. The value of this variable specifies the linker's default library search path. This preparation is used later in the chapter.

> **Warning**
>
> *Do not* remove the Binutils build and source directories yet. These will be needed again in their current state later in this chapter.

Details on this package are located in Section 6.13.2, “Contents of Binutils.”

5.5. GCC-3.4.1 - Pass 1

The GCC package contains the GNU compiler collection, which includes the C and C++ compilers.

Approximate build time: 4.4 SBU
Required disk space: 300 MB

GCC installation depends on: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, and Texinfo

5.5.1. Installation of GCC

Unpack only the gcc-core tarball because neither the C++ compiler nor the test suite will be needed here.

This package is known to have issues when its default optimization flags (including the *-march* and *-mcpu* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building GCC.

The GCC documentation recommends building GCC outside of the source directory in a dedicated build directory:

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

Prepare GCC for compilation:

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

The meaning of the configure options:

--with-local-prefix=/tools
: The purpose of this switch is to remove `/usr/local/include` from **gcc**'s include search path. This is not absolutely essential, however, it helps to minimize the influence of the host system.

--enable-shared
: This switch may seem counter-intuitive at first. However, this switch allows the building of `libgcc_s.so.1` and `libgcc_eh.a`, and having `libgcc_eh.a` available ensures that the configure script for Glibc (the next package we compile) produces the proper results. Note that the GCC binaries will still be linked statically because this is controlled by the *-static* value of the `BOOT_LDFLAGS` variable in the next step.

--enable-languages=c
: This option ensures that only the C compiler is built. This option is only needed when you have downloaded and unpacked the full GCC tarball, as opposed to just the gcc-core tarball.

Continue with compiling the package:

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

The meaning of the make parameters:

BOOT_LDFLAGS="-static"
: This tells GCC to link its programs statically.

bootstrap
: This target does not just compile GCC, but compiles it several times. It uses the programs compiled in a first round to compile itself a second time, and then again a third time. It then compares these second and third compiles to make sure it can reproduce itself flawlessly. This also implies that it was compiled correctly.

Compilation is now complete. At this point, the test suite would normally be run, but, as mentioned before, the test suite framework is not in place yet. The benefits of running the tests at this point are minimal since the programs from this first pass will soon be replaced.

Install the package:

```
make install
```

As a finishing touch, create a symlink. Many programs and scripts run **cc** instead of **gcc**, which is used to keep programs generic and therefore usable on all kinds of UNIX systems where the GNU C compiler is not always installed. Running **cc** leaves the system administrator free to decide which C compiler to install.

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

Details on this package are located in Section 6.14.2, “Contents of GCC.”

5.6. Linux-Libc-Headers-2.6.8.1

The Linux-Libc-Headers package contains the “sanitized” kernel headers.

Approximate build time: 0.1 SBU
Required disk space: 22 MB

Linux-Libc-Headers installation depends on: Coreutils

5.6.1. Installation of Linux-Libc-Headers

For years it has been common practice to use “raw” kernel headers (straight from a kernel tarball) in `/usr/include`, but over the last few years, the kernel developers have taken a strong stance that this should not be done. This gave birth to the Linux-Libc-Headers Project, which was designed to maintain an Application Programming Interface (API) stable version of the Linux headers.

Install the header files:

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

If your architecture is not i386 (compatible), adjust the first command accordingly.

Details on this package are located in Section 6.9.2, “Contents of Linux-Libc-Headers.”

5.7. Linux-2.6.8.1 Headers

The Linux kernel package contains the kernel source as well as the header files used by Glibc.

Approximate build time: 0.1 SBU
Required disk space: 186 MB

Linux Headers installation depends on: Coreutils and Make

5.7.1. Installation of the Kernel Headers

Because some packages need to refer to the kernel header files, now is a good time to unpack the kernel archive, set it up, and copy the required files to a place where **gcc** can locate them later.

Prepare for the header installation with:

```
make mrproper
```

This ensures that the kernel tree is absolutely clean. It is recommended that this command be issued prior to *each* kernel compilation. Do not assume that the source tree is automatically clean after un-tarring.

Create the `include/linux/version.h` file:

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

Create the platform-specific `include/asm` symlink:

```
make include/asm
```

Install the platform-specific header files:

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

Finally, install the cross-platform kernel header files:

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

Details on this package are located in Section 8.3.2, “Contents of Linux.”

5.8. Glibc-2.3.4-20040701

The Glibc package contains the main C library. This library provides the basic routines for allocating memory, searching directories, opening and closing files, reading and writing files, string handling, pattern matching, arithmetic, and so on.

Approximate build time: 11.8 SBU
Required disk space: 800 MB

Glibc installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed, and Texinfo

5.8.1. Installation of Glibc

This package is known to have issues when its default optimization flags (including the *-march* and *-mcpu* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building Glibc.

It should be noted that compiling Glibc in any way other than the method suggested in this book puts the stability of the system at risk.

The Glibc documentation recommends building Glibc outside of the source directory in a dedicated build directory:

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

Next, prepare Glibc for compilation:

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

The meaning of the configure options:

--disable-profile
: This builds the libraries without profiling information. Omit this option if profiling on the temporary tools is necessary.

--enable-add-ons=nptl
: This tells Glibc to use the NPTL add-on as its threading library.

--with-tls
: This tells Glibc to include support for Thread-Local Storage (TLS). This is required in order for NPTL to work.

--with-__thread
: This option tells Glibc to include thread support. It is required in order for TLS to be properly compiled.

--enable-kernel=2.6.0
: This tells Glibc to compile the library with support for 2.6.x Linux kernels.

--with-binutils=/tools/bin
: While not required, this switch ensures that there are no errors pertaining to which Binutils programs get used during the Glibc build.

--without-gd
: This prevents the build of the **memusagestat** program, which insists on linking against the host's libraries (libgd, libpng, libz, etc.).

--without-cvs
: This prevents the Makefile files from attempting automatic CVS checkouts when using a CVS snapshot. While this command is not required, it is recommended because it suppresses an annoying, but harmless, warning about a missing autoconf program.

--with-headers=/tools/glibc-kernheaders
: This tells Glibc to compile itself against the “raw” kernel headers, so that it knows exactly what features the kernel has and can optimize itself accordingly.

During this stage the following warning might appear:

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

The missing or incompatible **msgfmt** program is generally harmless, but it can sometimes cause issues when running the test suite. This **msgfmt** program is part of the Gettext package which the host distribution should provide. If **msgfmt** is present but deemed incompatible, upgrade the host system's Gettext package or continue without it and see if the test suite runs without problems regardless.

Compile the package:

```
make
```

Compilation is now complete. As mentioned earlier, running the test suites for the temporary tools installed in this chapter is not mandatory. To run the Glibc test suite (if desired), the following command will do so:

```
make check
```

For a discussion of test failures that are of particular importance, please see Section 6.11, “Glibc-2.3.4-20040701.”

In this chapter, some tests can be adversely effected by existing tools or environmental issues on the host system. Glibc test suite failures in this chapter are typically not worrisome. The Glibc installed in Chapter 6 is the one that will ultimately end up being used, so that is the one that needs to pass most tests (even in Chapter 6, some failures could still occur, for example, with the math tests).

When experiencing a failure, make a note of it, then continue by reissuing the **make check** command. The test suite should pick up where it left off and continue. This stop-start sequence can be circumvented by issuing a **make -k check** command. If using this option, be sure to log the output so that the log file can be examined for failures later.

The install stage of Glibc will issue a harmless warning at the end about the absence of `/tools/etc/ld.so.conf`. Prevent this warning with:

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

Install the package:

```
make install
```

Different countries and cultures have varying conventions for how to communicate. These conventions range from the format for representing dates and times to more complex issues, such as the language spoken. The “internationalization” of GNU programs works by locale.

Note

If the test suites are not being run in this chapter (as per the recommendation), there is no need to install the locales now. The appropriate locales will be installed in the next chapter.

To install the Glibc locales anyway, use the following command:

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

To save time, an alternative to running the previous command (which generates and installs every locale Glibc is aware of) is to install only those locales that are wanted and needed. This can be achieved by using the **localedef** command. Information on this command is located in the `INSTALL` file in the Glibc source. However, there are a number of locales that are essential in order for the tests of future packages to pass, in particular, the *libstdc++* tests from GCC. The following instructions, instead of the *`install-locales`* target used above, will install the minimum set of locales necessary for the tests to run successfully:

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

Details on this package are located in Section 6.11.4, “Contents of Glibc.”

5.9. Adjusting the Toolchain

Now that the temporary C libraries have been installed, all tools compiled in the rest of this chapter should be linked against these libraries. In order to accomplish this, the linker and the compiler's specs file need to be adjusted.

The linker, adjusted at the end of the first pass of Binutils, is installed by running the following command from within the `binutils-build` directory:

```
make -C ld install
```

From this point onwards, everything will link only against the libraries in `/tools/lib`.

Note

If the earlier warning to retain the Binutils source and build directories from the first pass was missed, ignore the above command. This results in a small chance that the subsequent testing programs will link against libraries on the host. This is not ideal, but it is not a major problem. The situation is corrected when the second pass of Binutils is installed later.

Now that the adjusted linker is installed, the Binutils build and source directories should be removed.

The next task is to amend the GCC specs file so that it points to the new dynamic linker. A simple sed script will accomplish this:

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

Alternatively, the specs file can be edited by hand. This is done by replacing every occurrence of “/lib/ld-linux.so.2” with “/tools/lib/ld-linux.so.2”

Be sure to visually inspect the specs file in order to verify the intended changes have been made.

Important

If working on a platform where the name of the dynamic linker is something other than `ld-linux.so.2`, replace “ld-linux.so.2” with the name of the platform's dynamic linker in the above commands. Refer back to Section 5.3, “Toolchain Technical Notes,” if necessary.

There is a possibility that some include files from the host system have found their way into GCC's private include dir. This can happen as a result of GCC's “fixincludes” process, which runs as part of the GCC build. This is explained in more detail later in this chapter. Run the following command to eliminate this possibility:

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

Caution

At this point, it is imperative to stop and ensure that the basic functions (compiling and linking) of the new toolchain are working as expected. To perform a sanity check, run the following commands:

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

If everything is working correctly, there should be no errors, and the output of the last command will be of the form:

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

Note that `/tools/lib` appears as the prefix of the dynamic linker.

If the output is not shown as above or there was no output at all, then something is wrong. Investigate and retrace the steps to find out where the problem is and correct it. This issue must be resolved before continuing on. First, perform the sanity check again, using **gcc** instead of **cc**. If this works, then the `/tools/bin/cc` symlink is missing. Revisit Section 5.5, “GCC-3.4.1 - Pass 1,” and install the symlink. Next, ensure that the `PATH` is correct. This can be checked by running **echo $PATH** and verifying that `/tools/bin` is at the head of the list. If the `PATH` is wrong it could mean that you are not logged in as user *lfs* or that something went wrong back in Section 4.4, “Setting Up the Environment.” Another option is that something may have gone wrong with the specs file amendment above. In this case, redo the specs file amendment.

Once all is well, clean up the test files:

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

5.10. Tcl-8.4.7

The Tcl package contains the Tool Command Language.

Approximate build time: 0.9 SBU
Required disk space: 23 MB

Tcl installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, and Sed

5.10.1. Installation of Tcl

This package and the next two (Expect and DejaGNU) are installed to support running the test suites for GCC and Binutils. Installing three packages for testing purposes may seem excessive, but it is very reassuring, if not essential, to know that the most important tools are working properly. Even if the test suites are not run in this chapter (they are not mandatory), these packages are required to run the test suites in Chapter 6.

Prepare Tcl for compilation:

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

Build the package:

```
make
```

To test the results, issue: **`TZ=UTC make test`**. The Tcl test suite is known to experience failures under certain host conditions that are not fully understood. Therefore, test suite failures here are not surprising, and are not considered critical. The *`TZ=UTC`* parameter sets the time zone to Coordinated Universal Time (UTC), also known as Greenwich Mean Time (GMT), but only for the duration of the test suite run. This ensures that the clock tests are exercised correctly. Details on the `TZ` environment variable is provided in Chapter 7.

Install the package:

```
make install
```

> **Warning**
>
> *Do not* remove the `tcl8.4.7` source directory yet, as the next package will need its internal headers.

Now make a necessary symbolic link:

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

5.10.2. Contents of Tcl

Installed programs: tclsh (link to tclsh8.4) and tclsh8.4
Installed library: libtcl8.4.so

Short Descriptions

tclsh8.4	The Tcl command shell
tclsh	A link to tclsh8.4
`libtcl8.4.so`	The Tcl library

5.11. Expect-5.42.1

The Expect package contains a program for carrying out scripted dialogues with other interactive programs.

Approximate build time: 0.1 SBU
Required disk space: 3.9 MB

Expect installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, and Tcl

5.11.1. Installation of Expect

First, fix a bug that can result in false failures during the GCC test suite run:

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

Now prepare Expect for compilation:

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

The meaning of the configure options:

--with-tcl=/tools/lib
: This ensures that the configure script finds the Tcl installation in the temporary tools location instead of possibly locating an existing one on the host system.

--with-x=no
: This tells the configure script not to search for Tk (the Tcl GUI component) or the X Window System libraries, both of which may reside on the host system.

Build the package:

```
make
```

To test the results, issue: **make test**. Note that the Expect test suite is known to experience failures under certain host conditions that are not within our control. Therefore, test suite failures here are not surprising and are not considered critical.

Install the package:

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

The meaning of the make parameter:

`SCRIPTS=""`
: This prevents installation of the supplementary expect scripts, which are not needed.

The source directories of both Tcl and Expect can now be removed.

5.11.2. Contents of Expect

Installed program: expect
Installed library: libexpect-5.42.a

Short Descriptions

expect	Communicates with other interactive programs according to a script
`libexpect-5.42.a`	Contains functions that allow Expect to be used as a Tcl extension or to be used directly from C or C++ (without Tcl)

5.12. DejaGNU-1.4.4

The DejaGNU package contains a framework for testing other programs.

Approximate build time: 0.1 SBU
Required disk space: 8.6 MB

DejaGNU installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, and Sed

5.12.1. Installation of DejaGNU

Prepare DejaGNU for compilation:

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

Build and install the package:

```
make install
```

5.12.2. Contents of DejaGNU

Installed program: runtest

Short Descriptions

runtest A wrapper script that locates the proper **expect** shell and then runs DejaGNU

5.13. GCC-3.4.1 - Pass 2

Approximate build time: 11.0 SBU
Required disk space: 274 MB

GCC installation depends on: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, and Texinfo

5.13.1. Re-installation of GCC

This package is known to have issues when its default optimization flags (including the *`-march`* and *`-mcpu`* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building GCC.

The tools required to test GCC and Binutils—Tcl, Expect and DejaGNU—are installed now. GCC and Binutils can now be rebuilt, linking them against the new Glibc and testing them properly (if running the test suites in this chapter). Please note that these test suites are highly dependent on properly functioning PTYs which are provided by the host. PTYs are most commonly implemented via the `devpts` file system. Check to see if the host system is set up correctly in this regard by performing a quick test:

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

The response might be:

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

If the above message is received, the host does not have its PTYs set up properly. In this case, there is no point in running the test suites for GCC and Binutils until this issue is resolved. Please consult the LFS Wiki at *http://wiki.linuxfromscratch.org/* for more information on how to get PTYs working.

Because the C and the C++ compilers will be built, unpack both the core and the g++ tarballs (as well as test suite, if you want to run the tests). By unpacking them in the working directory, they will all unfold into a single `gcc-3.4.1/` subdirectory.

First correct a known problem and make an essential adjustment:

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

The first patch disables the GCC **fixincludes** script. This was briefly mentioned earlier, but a more in-depth explanation of the fixincludes process is warranted here. Under normal circumstances, the GCC **fixincludes** script scans the system for header files that need to be fixed. It might find that some Glibc header files on the host system need to be fixed, and will fix them and put them in the GCC private include directory. In Chapter 6, after the newer Glibc has been installed, this private include directory will be searched before the system include directory. This may result in GCC finding the fixed headers from the host system, which most likely will not match the Glibc version used for the LFS system.

The second patch changes GCC's default location of the dynamic linker (typically `ld-linux.so.2`). It also removes `/usr/include` from GCC's include search path. Patching now rather than adjusting the specs file after installation ensures that the new dynamic linker is used during the actual build of GCC. That is, all of the final (and temporary) binaries created during the build will link against the new Glibc.

> **Important**
>
> The above patches are critical in ensuring a successful overall build. Do not forget to apply them.

Create a separate build directory again:

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

Before starting to build GCC, remember to unset any environment variables that override the default optimization flags.

Now prepare GCC for compilation:

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

The meaning of the new configure options:

--enable-clocale=gnu
: This option ensures the correct locale model is selected for the C++ libraries under all circumstances. If the configure script finds the *de_DE* locale installed, it will select the correct gnu locale model. However, if the *de_DE* locale is not installed, there is the risk of building Application Binary Interface (ABI)-incompatible C++ libraries because the incorrect generic locale model may be selected.

--enable-threads=posix
: This enables C++ exception handling for multi-threaded code.

--enable-__cxa_atexit
: This option allows use of *__cxa_atexit*, rather than *atexit*, to register C++ destructors for local statics and global objects. This option is essential for fully standards-compliant handling of destructors. It also effects the C++ ABI, and therefore results in C++ shared libraries and C++ programs that are interoperable with other Linux distributions.

--enable-languages=c,c++
: This option ensures that both the C and C++ compilers are built.

--disable-libstdcxx-pch
: Do not build the pre-compiled header (PCH) for `libstdc++`. It takes up a lot of space, and we have no use for it.

Compile the package:

```
make
```

There is no need to use the *bootstrap* target now because the compiler being used to compile this GCC was built from the exact same version of the GCC sources used earlier.

Compilation is now complete. As previously mentioned, running the test suites for the temporary tools compiled in this chapter is not mandatory. To run the GCC test suite anyway, use the following command:

```
make -k check
```

The *-k* flag is used to make the test suite run through to completion and not stop at the first failure. The GCC test suite is very comprehensive and is almost guaranteed to generate a few failures. To receive a summary of the test suite results, run:

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

For only the summaries, pipe the output through **grep -A7 Summ**.

Results can be compared to those posted to the gcc-testresults mailing list to see similar configurations to the one being built. For an example of how current GCC-3.4.1 should look on i686-pc-linux-gnu, see *http://gcc.gnu.org/ml/gcc-testresults/2004-07/msg00179.html*.

A few unexpected failures cannot always be avoided. The GCC developers are usually aware of these issues, but have not resolved them yet. Unless the test results are vastly different from those at the above URL, it is safe to continue.

Install the package:

```
make install
```

Note

At this point it is strongly recommended to repeat the sanity check we performed earlier in this chapter. Refer back to Section 5.9, “Adjusting the Toolchain,” and repeat the test compilation. If the result is wrong, the most likely reason is that the GCC Specs patch was not properly applied.

Details on this package are located in Section 6.14.2, “Contents of GCC.”

5.14. Binutils-2.15.91.0.2 - Pass 2

The Binutils package contains a linker, an assembler, and other tools for handling object files.

Approximate build time: 1.5 SBU
Required disk space: 108 MB

Binutils installation depends on: Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed, and Texinfo

5.14.1. Re-installation of Binutils

This package is known to have issues when its default optimization flags (including the *`-march`* and *`-mcpu`* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building Binutils.

Create a separate build directory again:

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

Prepare Binutils for compilation:

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

The meaning of the new configure option:

`--with-lib-path=/tools/lib`
: This tells the configure script to specify the library search path during the compilation of Binutils, resulting in `/tools/lib` being passed to the linker. This prevents the linker from searching through library directories on the host.

Compile the package:

```
make
```

Compilation is now complete. As discussed earlier, running the test suite is not mandatory for the temporary tools here in this chapter. To run the Binutils test suite anyway, issue the following command:

```
make check
```

Install the package:

```
make install
```

Now prepare the linker for the “Re-adjusting” phase in the next chapter:

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

Warning

Do not remove the Binutils source and build directories yet. These directories will be needed again in the next chapter in their current state.

Details on this package are located in Section 6.13.2, “Contents of Binutils.”

5.15. Gawk-3.1.4

The Gawk package contains programs for manipulating text files.

Approximate build time: 0.2 SBU
Required disk space: 17 MB

Gawk installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Sed

5.15.1. Installation of Gawk

Prepare Gawk for compilation:

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

Compile the package:

```
make
```

To test the results (not necessary), issue: **`make check`**.

Install the package:

```
make install
```

Details on this package are located in Section 6.20.2, “Contents of Gawk.”

5.16. Coreutils-5.2.1

The Coreutils package contains utilities for showing and setting the basic system characteristics.

Approximate build time: 0.9 SBU
Required disk space: 69 MB

Coreutils installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, and Sed

5.16.1. Installation of Coreutils

Prepare Coreutils for compilation:

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

This package has an issue when compiled against versions of Glibc later than 2.3.2. Some of the Coreutils utilities (such as **head**, **tail**, and **sort**) will reject their traditional syntax, a syntax that has been in use for approximately 30 years. This old syntax is so pervasive that compatibility should be preserved until the many places where it is used can be updated. Backwards compatibility is achieved by setting the `DEFAULT_POSIX2_VERSION` environment variable to “199209” in the above command. If you do not want Coreutils to be backwards compatible with the traditional syntax, then omit setting the `DEFAULT_POSIX2_VERSION` environment variable. It is important to remember that doing so will have consequences, including the need to patch the many packages that still use the old syntax. Therefore, it is recommended that the instructions be followed exactly as given above.

Compile the package:

```
make
```

To test the results, issue: **`make RUN_EXPENSIVE_TESTS=yes check`**. The *`RUN_EXPENSIVE_TESTS=yes`* parameter tells the test suite to run several additional tests that are considered relatively expensive (in terms of CPU power and memory usage) on some platforms, but generally are not a problem on Linux.

Install the package:

```
make install
```

Details on this package are located in Section 6.15.2, “Contents of Coreutils.”

5.17. Bzip2-1.0.2

The Bzip2 package contains programs for compressing and decompressing files. Text files yield a much better compression than with the traditional **gzip**.

Approximate build time: 0.1 SBU
Required disk space: 2.5 MB

Bzip2 installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, and Make

5.17.1. Installation of Bzip2

The Bzip2 package does not contain a **configure** script. Compile it with:

```
make
```

Install the package:

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

Details on this package are located in Section 6.40.2, "Contents of Bzip2."

5.18. Gzip-1.3.5

The Gzip package contains programs for compressing and decompressing files.

Approximate build time: 0.1 SBU
Required disk space: 2.6 MB

Gzip installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, and Sed

5.18.1. Installation of Gzip

Prepare Gzip for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Details on this package are located in Section 6.46.2, “Contents of Gzip.”

5.19. Diffutils-2.8.1

The Diffutils package contains programs that show the differences between files or directories.

Approximate build time: 0.1 SBU
Required disk space: 7.5 MB

Diffutils installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Sed

5.19.1. Installation of Diffutils

Prepare Diffutils for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Details on this package are located in Section 6.41.2, “Contents of Diffutils.”

5.20. Findutils-4.1.20

The Findutils package contains programs to find files. Processes are provided to recursively search through a directory tree and to create, maintain, and search a database (often faster than the recursive find, but unreliable if the database has not been recently updated).

Approximate build time: 0.2 SBU
Required disk space: 7.6 MB

Findutils installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make and Sed

5.20.1. Installation of Findutils

Prepare Findutils for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

Details on this package are located in Section 6.19.2, “Contents of Findutils.”

5.21. Make-3.80

The Make package contains a program for compiling large packages.

Approximate build time: 0.2 SBU
Required disk space: 8.8 MB

Make installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, and Sed

5.21.1. Installation of Make

Prepare Make for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

Details on this package are located in Section 6.48.2, “Contents of Make.”

5.22. Grep-2.5.1

The Grep package contains programs for searching through files.

Approximate build time: 0.1 SBU
Required disk space: 5.8 MB

Grep installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed, and Texinfo

5.22.1. Installation of Grep

Prepare Grep for compilation:

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

The meaning of the configure options:

--disable-perl-regexp
This makes sure that the **grep** program does not get linked against a Perl Compatible Regular Expression (PCRE) library that may be present on the host and would not be available once we enter the chroot environment.

--with-included-regex
This ensures that Grep uses its internal regular expression code. Without this switch, Grep will use the code from Glibc, which is known to be buggy.

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

Details on this package are located in Section 6.44.2, “Contents of Grep.”

5.23. Sed-4.1.2

The Sed package contains a stream editor.

Approximate build time: 0.2 SBU
Required disk space: 5.2 MB

Sed installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Texinfo

5.23.1. Installation of Sed

Prepare Sed for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

Details on this package are located in Section 6.28.2, “Contents of Sed.”

5.24. Gettext-0.14.1

The Gettext package contains utilities for internationalization and localization. These allow programs to be compiled with NLS, enabling them to output messages in the user's native language.

Approximate build time: 0.5 SBU
Required disk space: 55 MB

Gettext installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, and Sed

5.24.1. Installation of Gettext

Prepare Gettext for compilation:

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

The meaning of the configure options:

--disable-libasprintf
: This flag tells Gettext not to build the `asprintf` library. Because nothing in this chapter or the next requires this library and Gettext gets rebuilt later, exclude it to save time and space.

--disable-csharp
: This tells Gettext not to use a C# compiler, even if a C# compiler is installed on the host. This needs to be done because once we enter the chroot environment, C# will no longer be available.

Compile the package:

```
make
```

To test the results, issue: **make check**. This takes quite some time, around 7 SBUs. The Gettext test suite is known to experience failures under certain host conditions, for example when it finds a Java compiler on the host. An experimental patch to disable Java is available from the LFS Patches project at *http://www.linuxfromscratch.org/patches/*.

Install the package:

```
make install
```

Details on this package are located in Section 6.30.2, “Contents of Gettext.”

5.25. Ncurses-5.4

The Ncurses package contains libraries for terminal-independent handling of character screens.

Approximate build time: 0.7 SBU
Required disk space: 26 MB

Ncurses installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, and Sed

5.25.1. Installation of Ncurses

Prepare Ncurses for compilation:

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

The meaning of the configure options:

`--without-ada`
: This tells Ncurses not to build its Ada bindings, even if an Ada compiler is installed on the host. This needs to be done because once we enter the chroot environment, Ada will no longer be available.

`--enable-overwrite`
: This tells Ncurses to install its header files into `/tools/include`, instead of `/tools/include/ncurses`, to ensure that other packages can find the Ncurses headers successfully.

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Details on this package are located in Section 6.21.2, “Contents of Ncurses.”

5.26. Patch-2.5.4

The Patch package contains a program for modifying files.

Approximate build time: 0.1 SBU
Required disk space: 1.9 MB

Patch installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, and Sed

5.26.1. Installation of Patch

Prepare Patch for compilation:

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

The preprocessor flag *-D_GNU_SOURCE* is only needed on the PowerPC platform. It can be left out on other architectures.

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Details on this package are located in Section 6.50.2, “Contents of Patch.”

5.27. Tar-1.14

The Tar package contains an archiving program.

Approximate build time: 0.2 SBU
Required disk space: 10 MB

Tar installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Sed

5.27.1. Installation of Tar

Prepare Tar for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

Details on this package are located in Section 6.56.2, “Contents of Tar.”

5.28. Texinfo-4.7

The Texinfo package contains programs for reading, writing, and converting Info documents.

Approximate build time: 0.2 SBU
Required disk space: 16 MB

Texinfo installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, and Sed

5.28.1. Installation of Texinfo

Prepare Texinfo for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

Details on this package are located in Section 6.34.2, “Contents of Texinfo.”

5.29. Bash-3.0

The Bash package contains the Bourne-Again SHell.

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

Bash installation depends on: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, and Sed.

5.29.1. Installation of Bash

Prepare Bash for compilation:

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

The meaning of the configure option:

--without-bash-malloc
: This options turns off the use of Bash's memory allocation (malloc) function which is known to cause segmentation faults. By turning this option off, Bash will use the malloc functions from Glibc which are more stable.

Compile the package:

```
make
```

To test the results, issue: **make tests**.

Install the package:

```
make install
```

Make a link for the programs that use **sh** for a shell:

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

Details on this package are located in Section 6.37.2, “Contents of Bash.”

5.30. M4-1.4.2

The M4 package contains a macro processor.

Approximate build time: 0.1 SBU
Required disk space: 3.0 MB

M4 installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, and Sed

5.30.1. Installation of M4

Prepare M4 for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

Details on this package are located in Section 6.24.2, “Contents of M4.”

5.31. Bison-1.875a

The Bison package contains a parser generator.

Approximate build time: 0.6 SBU
Required disk space: 10.6 MB

Bison installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, and Sed

5.31.1. Installation of Bison

Prepare Bison for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

Details on this package are located in Section 6.25.2, “Contents of Bison.”

5.32. Flex-2.5.31

The Flex package contains a utility for generating programs that recognize patterns in text.

Approximate build time: 0.6 SBU
Required disk space: 10.6 MB

Flex installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, and Sed

5.32.1. Installation of Flex

Flex contains several known bugs. These can be fixed with the following patch:

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

The GNU autotools will detect that the Flex source code has been modified by the previous patch and tries to update the manual page accordingly. This does not work on many systems, and the default page is fine, so make sure it does not get regenerated:

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

Now prepare Flex for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

Details on this package are located in Section 6.29.2, “Contents of Flex.”

5.33. Util-linux-2.12b

The Util-linux package contains miscellaneous utility programs. Among them are utilities for handling file systems, consoles, partitions, and messages.

Approximate build time: 0.2 SBU
Required disk space: 16 MB

Util-linux installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, and Zlib

5.33.1. Installation of Util-linux

Util-linux does not use the freshly installed headers and libraries from the `/tools` directory. This is fixed by altering the configure script:

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

Prepare Util-linux for compilation:

```
./configure
```

Compile some support routines:

```
make -C lib
```

Since only a couple of the utilities contained in this package are needed, build only those:

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

This package does not come with a test suite.

Copy these programs to the temporary tools directory:

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

Details on this package are located in Section 6.58.3, “Contents of Util-linux.”

5.34. Perl-5.8.5

The Perl package contains the Practical Extraction and Report Language.

Approximate build time: 0.8 SBU
Required disk space: 74 MB

Perl installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, and Sed

5.34.1. Installation of Perl

First adapt some hard-wired paths to the C library by applying the following patch:

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

Prepare Perl for compilation (make sure to get the 'IO Fcntl POSIX' part of the command correct—they are all letters):

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

The meaning of the configure option:

-Dstatic_ext='IO Fcntl POSIX'
: This tells Perl to build the minimum set of static extensions needed for installing and testing the Coreutils package in the next chapter.

Compile only the required tools:

```
make perl utilities
```

Although Perl comes with a test suite, it is not recommended to run it at this point. Only part of Perl was built and running **`make test`** now will cause the rest of Perl to be built as well, which is unnecessary at this point. The test suite can be run in the next chapter if desired.

Copy these tools and their libraries:

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

Details on this package are located in Section 6.33.2, “Contents of Perl.”

5.35. Udev-030

The Udev package contains programs for dynamic creation of device nodes.

Approximate build time: 0.2 SBU
Required disk space: 5.2 MB

Udev installation depends on: Coreutils and Make

5.35.1. Installation of Udev

The **udevstart** program hardcodes the path to the **udev** program in itself, which would cause issues since **udev** was installed in a non-standard location. Fix this by running the following:

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

Also, ensure that **udev** knows the correct location to look for its configuration files:

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

Now compile Udev:

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

This package does not come with a test suite.

Install the package:

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

Udev's configuration is far from ideal by default, so install LFS-specific configuration files here:

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

Details on this package are located in Section 6.57.2, “Contents of Udev.”

5.36. Stripping

The steps in this section are optional, but if the LFS partition is rather small, it is beneficial to learn that unnecessary items can be removed. The executables and libraries built so far contain about 130 MB of unneeded debugging symbols. Remove those symbols with:

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

The last of the above commands will skip some twenty files, reporting that it does not recognize their file format. Most of these are scripts instead of binaries.

Take care *not* to use *--strip-unneeded* on the libraries. The static ones would be destroyed and the toolchain packages would need to be built all over again.

To save another 30 MB, remove the documentation:

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

There will now be at least 850 MB of free space on the LFS file system that can be used to build and install Glibc in the next phase. If you can build and install Glibc, you can build and install the rest too.

Part III. Building the LFS System

Chapter 6. Installing Basic System Software

6.1. Introduction

In this chapter, we enter the building site and start constructing the LFS system in earnest. That is, we chroot into the temporary mini Linux system, make a few final preparations, and then begin installing the packages.

The installation of this software is straightforward. Although in many cases the installation instructions could be made shorter and more generic, we have opted to provide the full instructions for every package to minimize the possibilities for mistakes. The key to learning what makes a Linux system work is to know what each package is used for and why the user (or the system) needs it. For every installed package, a summary of its contents is given, followed by concise descriptions of each program and library the package installed.

If using the compiler optimizations provided in this chapter, please review the optimization hint at *http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt*. Compiler optimizations can make a program run slightly faster, but they may also cause compilation difficulties and problems when running the program. If a package refuses to compile when using optimization, try to compile it without optimization and see if that fixes the problem. Even if the package does compile when using optimization, there is the risk it may have been compiled incorrectly because of the complex interactions between the code and build tools. The small potential gains achieved in using compiler optimizations are often outweighed by the risks. First-time builders of LFS are encouraged to build without custom optimizations. The subsequent system will still run very fast and be stable at the same time.

The order that packages are installed in this chapter needs to be strictly followed to ensure that no program accidentally acquires a path referring to `/tools` hard-wired into it. For the same reason, do not compile packages in parallel. Compiling in parallel may save time (especially on dual-CPU machines), but it could result in a program containing a hard-wired path to `/tools`, which will cause the program to stop working when that directory is removed.

Before the installation instructions, each installation page provides information about the package, including a concise description of what it contains, approximately how long it will take to build, how much disk space is required during this building process, and any other packages needed to successfully build the package. Following the installation instructions, there is a list of programs and libraries (along with brief descriptions of these) that the package installs.

To keep track of which package installs particular files, a package manager can be used. For a general overview of different styles of package managers, please refer to *http://www.linuxfromscratch.org/blfs/view/svn/introduction/important.html*. For a package management method specifically geared towards LFS, we recommend *http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt*.

Note

The remainder of this book is to be performed while logged in as user *root* and no longer as user *lfs*.

6.2. Mounting Virtual Kernel File Systems

Various file systems exported by the kernel do not exist on the hard drive, but are used to communicate to and from the kernel itself.

Begin by creating directories onto which the file systems will be mounted:

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

Now mount the file systems:

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

Remember that if for any reason you stop working on the LFS system and start again later, it is important to check that these file systems are mounted again before entering the chroot environment.

Additional file systems will soon be mounted from within the chroot environment. To keep the host up to date, perform a “fake mount” for each of these now:

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

6.3. Entering the Chroot Environment

It is time to enter the chroot environment to begin building and installing the final LFS system. As user *root*, run the following command to enter the realm that is, at the moment, populated with only the temporary tools:

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

The `-i` option given to the **env** command will clear all variables of the chroot environment. After that, only the `HOME`, `TERM`, `PS1`, and `PATH` variables are set again. The *`TERM=$TERM`* construct will set the `TERM` variable inside chroot to the same value as outside chroot. This variable is needed for programs like **vim** and **less** to operate properly. If other variables are needed, such as `CFLAGS` or `CXXFLAGS`, this is a good place to set them again.

From this point on, there is no need to use the `LFS` variable anymore, because all work will be restricted to the LFS file system. This is because the Bash shell is told that `$LFS` is now the root (`/`) directory.

Notice that `/tools/bin` comes last in the `PATH`. This means that a temporary tool will not be used anymore as soon as its final version is installed. This occurs when the shell does not “remember” the locations of executed binaries—for this reason, hashing is switched off by passing the *`+h`* option to **bash**.

It is important that all the commands throughout the remainder of this chapter and the following chapters be run from within the chroot environment. If you leave this environment for any reason (rebooting for example), remember to first mount the `proc` and `devpts` file systems (discussed in the previous section) and enter chroot again before continuing with the installations.

Note that the bash prompt will say “I have no name!” This is normal because the `/etc/passwd` file has not been created yet.

6.4. Changing Ownership

Currently, the `/tools` directory is owned by the user *lfs*, a user that exists only on the host system. Although the `/tools` directory can be deleted once the LFS system has been finished, it can be retained to build additional LFS systems. If the `/tools` directory is kept as is, the files are owned by a user ID without a corresponding account. This is dangerous because a user account created later could get this same user ID and would own the `/tools` directory and all the files therein, thus exposing these files to possible malicious manipulation.

To avoid this issue, add the *lfs* user to the new LFS system later when creating the `/etc/passwd` file, taking care to assign it the same user and group IDs as on the host system. Alternatively, assign the contents of the `/tools` directory to user *root* by running the following command:

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

The command uses *`0:0`* instead of *`root:root`*, because **chown** is unable to resolve the name “root” until the password file has been created. This book assumes you ran this **chown** command.

6.5. Creating Directories

It is time to create some structure in the LFS file system. Create a directory tree. Issuing the following commands will create a standard tree:

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

Directories are, by default, created with permission mode 755, but this is not desirable for all directories. In the commands above, two changes are made—one to the home directory of user *root*, and another to the directories for temporary files.

The first mode change ensures that not just anybody can enter the `/root` directory—the same as a normal user would do with his or her home directory. The second mode change makes sure that any user can write to the `/tmp` and `/var/tmp` directories, but cannot remove other users' files from them. The latter is prohibited by the so-called “sticky bit,” the highest bit (1) in the 1777 bit mask.

6.5.1. FHS Compliance Note

The directory tree is based on the Filesystem Hierarchy Standard (FHS) standard (available at *http://www.pathname.com/fhs/*). Besides the tree created above, this standard stipulates the existence of `/usr/local/games` and `/usr/share/games`. We do not recommend these for a base system, however, feel free to make the system FHS-compliant. The FHS is not precise as to the structure of the `/usr/local/share` subdirectory, so we created only the directories that are needed.

6.6. Creating Essential Symlinks

Some programs hard-wire paths to programs which do not yet exist. In order to satisfy these programs, create a number of symbolic links which will be replaced by real files throughout the course of this chapter after the software has been installed.

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

6.7. Creating the passwd, group, and log Files

In order for user *root* to be able to login and for the name "root" to be recognized, there need to be relevant entries in the `/etc/passwd` and `/etc/group` files.

Create the `/etc/passwd` file by running the following command:

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

The actual password for *root* (the "x" used here is just a placeholder) will be set later.

Create the `/etc/group` file by running the following command:

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

The created groups are not part of any standard—they are some of the groups that the Udev configuration will be using in the next section. The Linux Standard Base (LSB, available at *http://www.linuxbase.org*) recommends only that, besides the group "root" with a Group ID (GID) of 0, a group "bin" with a GID of 1 be present. All other group names and GIDs can be chosen freely by the system administrator since well-written packages do not depend on GID numbers, but rather use the group's name.

To remove the “I have no name!” prompt, start a new shell. Since a full Glibc was installed in Chapter 5 and the `/etc/passwd` and `/etc/group` files have been created, user name and group name resolution will now work.

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

Note the use of the *+h* directive. This tells **bash** not to use its internal path hashing. Without this directive, **bash** would remember the paths to binaries it has executed. In order to use the newly compiled binaries as soon as they are installed, turn off this function for the duration of this chapter.

The **login**, **agetty**, and **init** programs (and others) use a number of log files to record information such as who was logged into the system and when. However, these programs will not write to the log files if they do not already exist. Initialize the log files and give them proper permissions:

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

The `/var/run/utmp` file records the users that are currently logged in. The `/var/log/wtmp` file records all logins and logouts. The `/var/log/lastlog` file records when each user last logged in. The `/var/log/btmp` file records the bad login attempts.

6.8. Populating /dev

6.8.1. Creating Initial Device Nodes

When the kernel boots the system, it requires the presence of a few device nodes, in particular the `console` and `null` devices. Create these by running the following commands:

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

6.8.2. Mounting ramfs and Populating /dev

The ideal way to populate `/dev` is to mount a `ramfs` onto `/dev`, like `tmpfs`, and create the devices on there during each bootup. Since the system has not been booted, it is necessary to do what the bootscripts would otherwise do and populate `/dev`. Begin by mounting `/dev`:

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

Run the installed **udevstart** program to create the initial devices based on all the information in `/sys`:

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

There are some symlinks and directories required by LFS that are not created by Udev, so create those here:

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

Finally, mount the proper virtual (kernel) file systems on the newly-created directories:

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

The **mount** commands executed above may result in the following warning message:

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

This file—`/etc/fstab`—has not been created yet but is also not required for the file systems to be properly mounted. As such, the warning can be safely ignored.

6.9. Linux-Libc-Headers-2.6.8.1

The Linux-Libc-Headers package contains the “sanitized” kernel headers.

Approximate build time: 0.1 SBU
Required disk space: 22 MB

Linux-Libc-Headers installation depends on: Coreutils

6.9.1. Installation of Linux-Libc-Headers

For years it has been common practice to use “raw” kernel headers (straight from a kernel tarball) in `/usr/include`, but over the last few years, the kernel developers have taken a strong stance that this should not be done. This gave birth to the Linux-Libc-Headers Project, which was designed to maintain an API stable version of the Linux headers.

Install the header files:

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

Ensure that all the headers are owned by root:

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

Make sure the users can read the headers:

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

6.9.2. Contents of Linux-Libc-Headers

Installed headers: /usr/include/{asm,linux}/*.h

Short Descriptions

`/usr/include/{asm,linux}/*.h`	The Linux headers API

6.10. Man-pages-1.67

The Man-pages package contains over 1,200 manual pages.

Approximate build time: 0.1 SBU
Required disk space: 15 MB

Man-pages installation depends on: Bash, Coreutils, and Make

6.10.1. Installation of Man-pages

Install Man-pages by running:

```
make install
```

6.10.2. Contents of Man-pages

Installed files: various manual pages

Short Descriptions

`manual pages`	Describe the C and C++ functions, important device files, and significant configuration files

6.11. Glibc-2.3.4-20040701

The Glibc package contains the main C library. This library provides the basic routines for allocating memory, searching directories, opening and closing files, reading and writing files, string handling, pattern matching, arithmetic, and so on.

Approximate build time: 12.3 SBU
Required disk space: 784 MB

Glibc installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed, and Texinfo

6.11.1. Installation of Glibc

This package is known to have issues when its default optimization flags (including the *-march* and *-mcpu* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building Glibc.

The Glibc build system is self-contained and will install perfectly, even though the compiler specs file and linker are still pointing at `/tools`. The specs and linker cannot be adjusted before the Glibc install because the Glibc autoconf tests would give false results and defeat the goal of achieving a clean build.

The Glibc documentation recommends building Glibc outside of the source directory in a dedicated build directory:

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

Prepare Glibc for compilation:

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

The meaning of the new configure option:

`--libexecdir=/usr/lib/glibc`
: This changes the location of the **pt_chown** program from its default of `/usr/libexec` to `/usr/lib/glibc`.

Compile the package:

```
make
```

> **Important**
>
> In this section, the test suite for Glibc is considered critical. Do not skip it under any circumstance.

Test the results:

```
make check
```

The Glibc test suite is highly dependent on certain functions of the host system, in particular the kernel. In general, the Glibc test suite is always expected to pass. However, in certain circumstances, some failures are unavoidable. This is a list of the most common issues:

- The *math* tests sometimes fail when running on systems where the CPU is not a relatively new genuine Intel or authentic AMD. Certain optimization settings are also known to be a factor here.
- The *gettext* test sometimes fails due to host system issues. The exact reasons are not yet clear.
- The *atime* test sometimes fails when the LFS partition is mounted with the *`noatime`* option.
- The *shm* test can fail when the host system is using the `devfs` file system but does not have the `tmpfs` file system mounted at `/dev/shm`. This occurs because of a lack of support for `tmpfs` in the kernel.
- When running on older and slower hardware, some tests can fail because of test timeouts being exceeded.

Though it is a harmless message, the install stage of Glibc will complain about the absence of `/etc/ld.so.conf`. Prevent this warning with:

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

Install the package:

```
make install
```

The locales that can make the system respond in a different language were not installed by the above command. Install this with:

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

To save time, an alternative to running the previous command (which generates and installs every locale Glibc is aware of) is to install only those locales that are wanted and needed. This can be achieved by using the **localedef** command. Information on this command is located in the `INSTALL` file in the Glibc source. However, there are a number of locales that are essential in order for the tests of future packages to pass, in particular, the *libstdc++* tests from GCC. The following instructions, instead of the *`install-locales`* target used above, will install the minimum set of locales necessary for the tests to run successfully:

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

Some locales installed by the **make localedata/install-locales** command above are not properly supported by some applications that are in the LFS and BLFS books. Because of the various problems that arise due to application programmers making assumptions that break in such locales, LFS should not be used in locales that utilize multibyte character sets (including UTF-8) or right-to-left writing order. Numerous unofficial and unstable patches are required to fix these problems, and it has been decided by the LFS developers not to support such complex locales. This applies to the ja_JP and fa_IR locales as well—they have been installed only for GCC and Gettext tests to pass, and the **watch** program (part of the Procps package) does not work properly in them. Various attempts to circumvent these restrictions are documented in internationalization-related hints.

Build the linuxthreads man pages, which are a great reference on the threading API (applicable to NPTL as well):

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

Install these pages:

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

6.11.2. Configuring Glibc

The `/etc/nsswitch.conf` file needs to be created because, although Glibc provides defaults when this file is missing or corrupt, the Glibc defaults do not work well with networking. The time zone also needs to be set up.

Create a new file `/etc/nsswitch.conf` by running the following:

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

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

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

# End /etc/nsswitch.conf
EOF
```

To determine the local time zone, run the following script:

```
tzselect
```

After answering a few questions about the location, the script will output the name of the time zone (e.g., *EST5EDT* or *Canada/Eastern*). Then create the `/etc/localtime` file by running:

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

Replace `[xxx]` with the name of the time zone that the **tzselect** provided (e.g., Canada/Eastern).

The meaning of the cp option:

`--remove-destination`
: This is needed to force removal of the already existing symbolic link. The reason for copying the file instead of using a symlink is to cover the situation where `/usr` is on a separate partition. This could be important when booted into single user mode.

6.11.3. Configuring Dynamic Loader

By default, the dynamic loader (`/lib/ld-linux.so.2`) searches through `/lib` and `/usr/lib` for dynamic libraries that are needed by programs as they are run. However, if there are libraries in directories other than `/lib` and `/usr/lib`, these need to be added to the `/etc/ld.so.conf` file in order for the dynamic loader to find them. Two directories that are commonly known to contain additional libraries are `/usr/local/lib` and `/opt/lib`, so add those directories to the dynamic loader's search path.

Create a new file `/etc/ld.so.conf` by running the following:

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

/usr/local/lib
/opt/lib

# End /etc/ld.so.conf
EOF
```

6.11.4. Contents of Glibc

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

Short Descriptions

catchsegv	Can be used to create a stack trace when a program terminates with a segmentation fault
gencat	Generates message catalogues
getconf	Displays the system configuration values for file system specific variables
getent	Gets entries from an administrative database
iconv	Performs character set conversion
iconvconfig	Creates fastloading **iconv** module configuration files
ldconfig	Configures the dynamic linker runtime bindings
ldd	Reports which shared libraries are required by each given program or shared library
lddlibc4	Assists **ldd** with object files
locale	Tells the compiler to enable or disable the use of POSIX locales for built-in operations
localedef	Compiles locale specifications
mtrace	Reads and interprets a memory trace file and ouputs a summary in human-readable format
nscd	A daemon that provides a cache for the most common name service requests

nscd_nischeck	Checks whether or not secure mode is necessary for NIS+ lookup
pcprofiledump	Dumps information generated by PC profiling
pt_chown	A helper program for **grantpt** to set the owner, group and access permissions of a slave pseudo terminal
rpcgen	Generates C code to implement the Remote Procecure Call (RPC) protocol
rpcinfo	Makes an RPC call to an RPC server
sln	A statically linked **ln** program
sprof	Reads and displays shared object profiling data
tzselect	Asks the user about the location of the system and reports the corresponding time zone description
xtrace	Traces the execution of a program by printing the currently executed function
zdump	The time zone dumper
zic	The time zone compiler
`ld.so`	The helper program for shared library executables
`libBrokenLocale`	Used by programs, such as Mozilla, to solve broken locales
`libSegFault`	The segmentation fault signal handler
`libanl`	An asynchronous name lookup library
`libbsd-compat`	Provides the portability needed in order to run certain Berkey Software Distribution (BSD) programs under Linux
`libc`	The main C library
`libcrypt`	The cryptography library
`libdl`	The dynamic linking interface library
`libg`	A runtime library for **g++**
`libieee`	The Institute of Electrical and Electronic Engineers (IEEE) floating point library

`libm`	The mathematical library
`libmcheck`	Contains code run at boot
`libmemusage`	Used by **memusage** to help collect information about the memory usage of a program
`libnsl`	The network services library
`libnss`	The Name Service Switch libraries, containing functions for resolving host names, user names, group names, aliases, services, protocols, etc
`libpcprofile`	Contains profiling functions used to track the amount of CPU time spent in specific source code lines
`libpthread`	The POSIX threads library
`libresolv`	Contains functions for creating, sending, and interpreting packets to the Internet domain name servers
`librpcsvc`	Contains functions providing miscellaneous RPC services
`librt`	Contains functions providing most of the interfaces specified by the POSIX.1b Realtime Extension
`libthread_db`	Contains functions useful for building debuggers for multi-threaded programs
`libutil`	Contains code for “standard” functions used in many different Unix utilities

6.12. Re-adjusting the Toolchain

Now that the new and final C libraries have been installed, it is time to adjust the toolchain again. The toolchain will be adjusted so that it will link any newly compiled program against these new libraries. This is the same process used in the “Adjusting” phase in the beginning of Chapter 5, even though it looks to be reversed. In Chapter 5, the chain was guided from the host's `/{,usr/}lib` directories to the new `/tools/lib` directory. Now, the chain will be guided from that same `/tools/lib` directory to the LFS `/{,usr/}lib` directories.

Start by adjusting the linker. The source and build directories from the second pass over Binutils were retained for this purpose. Install the adjusted linker by running the following command from within the `binutils-build` directory:

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

Note

If the earlier warning to retain the Binutils source and build directories from the second pass in Chapter 5 was missed, or if they were accidentally deleted or are inaccessible, ignore the above command. The result will be that the next package, Binutils, will link against the C libraries in `/tools` rather than in `/{,usr/}lib`. This is not ideal, however, testing has shown that the resulting Binutils program binaries should be identical.

From now on, every compiled program will link only against the libraries in `/usr/lib` and `/lib`. The extra *`INSTALL=/tools/bin/install`* option is needed because the `Makefile` file created during the second pass still contains the reference to **/usr/bin/install**, which has not been installed yet. Some host distributions contain a `ginstall` symbolic link which takes precedence in the `Makefile` file and can cause a problem. The above command takes care of this issue.

Remove the Binutils source and build directories now.

Next, amend the GCC specs file so that it points to the new dynamic linker. A sed command accomplishes this:

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

It is a good idea to visually inspect the specs file to verify the intended change was actually made.

Important

If working on a platform where the name of the dynamic linker is something other than `ld-linux.so.2`, substitute “ld-linux.so.2” with the name of the platform's dynamic linker in the above commands. Refer back to Section 5.3, “Toolchain Technical Notes,” if necessary.

Caution

It is imperative at this point to stop and ensure that the basic functions (compiling and linking) of the adjusted toolchain are working as expected. To do this, perform a sanity check:

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

If everything is working correctly, there should be no errors, and the output of the last command will be (allowing for platform-specific differences in dynamic linker name):

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

Note that `/lib` is now the prefix of our dynamic linker.

If the output does not appear as shown above or is not received at all, then something is seriously wrong. Investigate and retrace the steps to find out where the problem is and correct it. The most likely reason is that something went wrong with the specs file amendment above. Any issues will need to be resolved before continuing on with the process.

Once everything is working correctly, clean up the test files:

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

6.13. Binutils-2.15.91.0.2

The Binutils package contains a linker, an assembler, and other tools for handling object files.

Approximate build time: 1.4 SBU
Required disk space: 167 MB

Binutils installation depends on: Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed, and Texinfo

6.13.1. Installation of Binutils

This package is known to have issues when its default optimization flags (including the *-march* and *-mcpu* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building Binutils.

Verify that the PTYs are working properly inside the chroot environment. Check that everything is set up correctly by performing a simple test:

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

If the following message shows up, the chroot environment is not set up for proper PTY operation:

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

This issue needs to be resolved before running the test suites for Binutils and GCC.

The Binutils documentation recommends building Binutils outside of the source directory in a dedicated build directory:

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

Prepare Binutils for compilation:

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

Compile the package:

```
make tooldir=/usr
```

Normally, the tooldir (the directory where the executables will ultimately be located) is set to `$(exec_prefix)/$(target_alias)`, which expands into `/usr/i686-pc-linux-gnu`. Because this is a custom system, this target-specific directory in `/usr` is not required. This setup would be used if the system was used to cross-compile (for example, compiling a package on an Intel machine that generates code that can be executed on PowerPC machines).

> **Important**
>
> The test suite for Binutils in this section is considered critical. Do not skip it under any circumstances.

Test the results:

```
make check
```

Install the package:

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

Install the `libiberty` header file that is needed by some packages:

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

6.13.2. Contents of Binutils

Installed programs: addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings, and strip
Installed libraries: libiberty.a, libbfd.[a,so], and libopcodes.[a,so]

Short Descriptions

addr2line Translates program addresses to file names and line numbers; given an address and the name of an executable, it uses the debugging information in the executable to determine which source file and line number are associated with the address

ar	Creates, modifies, and extracts from archives
as	An assembler that assembles the output of **gcc** into object files
c++filt	Used by the linker to de-mangle C++ and Java symbols and to keep overloaded functions from clashing
gprof	Displays call graph profile data
ld	A linker that combines a number of object and archive files into a single file, relocating their data and tying up symbol references
nm	Lists the symbols occurring in a given object file
objcopy	Translates one type of object file into another
objdump	Displays information about the given object file, with options controlling the particular information to display; the information shown is useful to programmers who are working on the compilation tools
ranlib	Generates an index of the contents of an archive and stores it in the archive; the index lists all of the symbols defined by archive members that are relocatable object files
readelf	Displays information about ELF type binaries
size	Lists the section sizes and the total size for the given object files
strings	Outputs, for each given file, the sequences of printable characters that are of at least the specified length (defaulting to four); for object files, it prints, by default, only the strings from the initializing and loading sections while for other types of files, it scans the entire file
strip	Discards symbols from object files
`libiberty`	Contains routines used by various GNU programs, including **getopt**, **obstack**, **strerror**, **strtol**, and **strtoul**
`libbfd`	The Binary File Descriptor library
`libopcodes`	A library for dealing with opcodes—the “readable text” versions of instructions for the processor; it is used for building utilities like **objdump**.

6.14. GCC-3.4.1

The GCC package contains the GNU compiler collection, which includes the C and C++ compilers.

Approximate build time: 11.7 SBU
Required disk space: 294 MB

GCC installation depends on: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, and Texinfo

6.14.1. Installation of GCC

This package is known to have issues when its default optimization flags (including the *`-march`* and *`-mcpu`* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building GCC.

Unpack both the gcc-core and the gcc-g++ tarballs—they will unpack into the same directory. Likewise, extract the gcc-testsuite package. The full GCC package contains additional compilers. Instructions for building these can be found at *http://www.linuxfromscratch.org/blfs/view/svn/general/gcc.html*.

Apply only the No-Fixincludes patch (not the Specs patch) also used in the previous chapter:

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

GCC fails to compile some packages outside of a base Linux From Scratch install (e.g., Mozilla and kdegraphics) when used in conjunction with newer versions of Binutils. Apply the following patch to fix this issue:

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

Apply a sed substitution that will suppress the installation of `libiberty.a`. The version of `libiberty.a` provided by Binutils will be used instead:

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

The GCC documentation recommends building GCC outside of the source directory in a dedicated build directory:

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

Prepare GCC for compilation:

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

Compile the package:

```
make
```

Important

In this section, the test suite for GCC is considered critical. Do not skip it under any circumstance.

Test the results, but do not stop at errors:

```
make -k check
```

Some of the errors are known issues and were noted in the previous chapter. The test suite notes from Section 5.13, "GCC-3.4.1 - Pass 2," are still relevant here. Be sure to refer back to them as necessary.

Install the package:

```
make install
```

Some packages expect the C PreProcessor to be installed in the `/lib` directory. To support those packages, create this symlink:

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

Many packages use the name **cc** to call the C compiler. To satisfy those packages, create a symlink:

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

> **Note**
>
> At this point, it is strongly recommended to repeat the sanity check performed earlier in this chapter. Refer back to Section 6.12, “Re-adjusting the Toolchain,” and repeat the check. If the results are in error, then the most likely reason is that the GCC Specs patch from Chapter 5 was erroneously applied here.

6.14.2. Contents of GCC

Installed programs: c++, cc (link to gcc), cpp, g++, gcc, gccbug, and gcov
Installed libraries: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so], and libsupc++.a

Short Descriptions

cc	The C compiler
cpp	The C preprocessor; it is used by the compiler to expand the #include, #define, and similar statements in the source files
c++	The C++ compiler
g++	The C++ compiler
gcc	The C compiler
gccbug	A shell script used to help create useful bug reports
gcov	A coverage testing tool; it is used to analyze programs to determine where optimizations will have the most effect
`libgcc`	Contains run-time support for **gcc**
`libstdc++`	The standard C++ library
`libsupc++`	Provides supporting routines for the C++ programming language

6.15. Coreutils-5.2.1

The Coreutils package contains utilities for showing and setting the basic system characteristics.

Approximate build time: 0.9 SBU
Required disk space: 69 MB

Coreutils installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, and Sed

6.15.1. Installation of Coreutils

A known issue with the **uname** program from this package is that the *-p* switch always returns `unknown`. The following patch fixes this behavior for Intel architectures:

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

Prevent Coreutils from installing binaries that will be later be installed by other packages:

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

Now prepare Coreutils for compilation:

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

Compile the package:

```
make
```

The test suite of Coreutils makes several assumptions about the presence of files and users that are not valid this early in the LFS build. Therefore, additional items need to be set up before running the tests. Skip down to “Install the package” if not running the test suite.

Create two dummy groups and a dummy user name:

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

Now the test suite is ready to be run. First, run the tests that are meant to be run as user *root*:

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

Then run the remainder of the tests as the *dummy* user:

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

When testing is complete, remove the dummy user and groups:

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

Install the package:

```
make install
```

Move programs to the proper locations:

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

Finally, create a symlink to be FHS-compliant:

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

6.15.2. Contents of Coreutils

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

Short Descriptions

basename Strips any path and a given suffix from a file name

cat Concatenates files to standard output

chgrp Changes the group ownership of each given file to the given group; the group can either be either given a name or a numeric ID

chmod Changes the permissions of each file to the given mode; the mode can be either a symbolic representation of the changes to make or an octal number representing the new permissions

chown Changes the user and/or group ownership of each given file to the given user/group pair

chroot Runs a command with the specified directory as the `/` directory

cksum Prints the Cyclic Redundancy Check (CRC) checksum and the byte counts of each specified file

comm Compares two sorted files, outputting in three columns the lines that are unique and the lines that are common

cp Copies files

csplit Splits a given file into several new files, separating them according to given patterns or line numbers and outputting the byte count of each new file

cut Prints sections of lines, selecting the parts according to given fields or positions

date Displays the current time in the given format, or sets the system date

dd Copies a file using the given block size and count, while optionally performing conversions on it

df Reports the amount of disk space available (and used) on all mounted file systems, or only on the file systems holding the selected files

dir Lists the contents of each given directory (the same as the **ls** command)

dircolors Outputs commands to set the `LS_COLOR` environment variable to change the color scheme used by **ls**

dirname Strips the non-directory suffix from a file name

du Reports the amount of disk space used by the current directory, by each of the given directories (including all subdirectories) or by each of the given files

echo Displays the given strings

env	Runs a command in a modified environment
expand	Converts tabs to spaces
expr	Evaluates expressions
factor	Prints the prime factors of all specified integer numbers
false	Does nothing, unsuccessfully; it always exits with a status code indicating failure
fmt	Reformats the paragraphs in the given files
fold	Wraps the lines in the given files
groups	Reports a user's group memberships
head	Prints the first ten lines (or the given number of lines) of each given file
hostid	Reports the numeric identifier (in hexadecimal) of the host
hostname	Reports or sets the name of the host
id	Reports the effective user ID, group ID, and group memberships of the current user or specified user
install	Copies files while setting their permission modes and, if possible, their owner and group
join	Joins the lines that have identical join fields from two separate files
link	Creates a hard link with the given name to a file
ln	Makes hard links or soft (symbolic) links between files
logname	Reports the current user's login name
ls	Lists the contents of each given directory
md5sum	Reports or checks Message Digest 5 (MD5) checksums
mkdir	Creates directories with the given names
mkfifo	Creates First-In, First-Outs (FIFOs), a “named pipe” in UNIX parlance, with the given names
mknod	Creates device nodes with the given names; a device node is a character special file, a block special file, or a FIFO

mv	Moves or renames files or directories
nice	Runs a program with modified scheduling priority
nl	Numbers the lines from the given files
nohup	Runs a command immune to hangups, with its output redirected to a log file
od	Dumps files in octal and other formats
paste	Merges the given files, joining sequentially corresponding lines side by side, separated by tab characters
pathchk	Checks if file names are valid or portable
pinky	Is a lightweight finger client; it reports some information about the given users
pr	Paginates and columnates files for printing
printenv	Prints the environment
printf	Prints the given arguments according to the given format, much like the C printf function
ptx	Produces a permuted index from the contents of the given files, with each keyword in its context
pwd	Reports the name of the current working directory
readlink	Reports the value of the given symbolic link
rm	Removes files or directories
rmdir	Removes directories if they are empty
seq	Prints a sequence of numbers within a given range and with a given increment
sha1sum	Prints or checks 160-bit Secure Hash Algorithm 1 (SHA1) checksums
shred	Overwrites the given files repeatedly with complex patterns, making it difficult to recover the data
sleep	Pauses for the given amount of time
sort	Sorts the lines from the given files

split Splits the given file into pieces, by size or by number of lines

stat Displays file or filesystem status

stty Sets or reports terminal line settings

sum Prints checksum and block counts for each given file

sync Flushes file system buffers; it forces changed blocks to disk and updates the super block

tac Concatenates the given files in reverse

tail Prints the last ten lines (or the given number of lines) of each given file

tee Reads from standard input while writing both to standard output and to the given files

test Compares values and checks file types

touch Changes file timestamps, setting the access and modification times of the given files to the current time; files that do not exist are created with zero length

tr Translates, squeezes, and deletes the given characters from standard input

true Does nothing, successfully; it always exits with a status code indicating success

tsort Performs a topological sort; it writes a completely ordered list according to the partial ordering in a given file

tty Reports the file name of the terminal connected to standard input

uname Reports system information

unexpand Converts spaces to tabs

uniq Discards all but one of successive identical lines

unlink Removes the given file

users Reports the names of the users currently logged on

vdir Is the same as **ls -l**

wc Reports the number of lines, words, and bytes for each given file, as well as a total line when more than one file is given

who	Reports who is logged on
whoami	Reports the user name associated with the current effective user ID
yes	Repeatedly outputs “y” or a given string until killed

6.16. Zlib-1.2.1

The Zlib package contains compression and un-compression routines used by some programs.

Approximate build time: 0.1 SBU
Required disk space: 1.5 MB

Zlib installation depends on: Binutils, Coreutils, GCC, Glibc, Make, and Sed

6.16.1. Installation of Zlib

The following patch fixes a Denial of Service vulnerability in the Zlib compression library:

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

> **Note**
>
> Zlib is known to build its shared library incorrectly if `CFLAGS` is specified in the environment. If using a specified `CFLAGS` variable, be sure to add the *`-fPIC`* directive to the `CFLAGS` variable for the duration of the configure command below, then remove it afterwards.

Prepare Zlib for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the shared library:

```
make install
```

Build the static library:

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

To test the results again, issue: **`make check`**.

Install the static library:

```
make install
```

Fix the permissions on the static library:

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

It is good policy and common practice to place important libraries into the `/lib` directory. This is most important in scenarios where `/usr` is on a separate partition. Essentially, the run-time components of any libraries that are used by programs in `/bin` or `/sbin` should reside in `/lib` so that they are on the root partition and available in the event of `/usr` being inaccessible.

For the above reason, move the run-time components of the shared Zlib into `/lib`:

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

Fix the `/usr/lib/libz.so` symlink:

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

6.16.2. Contents of Zlib

Installed libraries: libz[a,so]

Short Descriptions

`libz` Contains compression and un-compression functions used by some programs

6.17. Mktemp-1.5

The Mktemp package contains programs used to create secure temporary files in shell scripts.

Approximate build time: 0.1 SBU
Required disk space: 317 KB

Mktemp installation depends on: Coreutils, Make, and Patch

6.17.1. Installation of Mktemp

Many scripts still use the deprecated **tempfile** program, which has functionality similar to **mktemp**. Patch Mktemp to include a **tempfile** wrapper:

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

Prepare Mktemp for compilation:

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

The meaning of the configure option:

--with-libc
: This causes the **mktemp** program to use the *mkstemp* and *mkdtemp* functions from the system C library.

Compile the package:

```
make
```

Install the package:

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

6.17.2. Contents of Mktemp

Installed programs: mktemp and tempfile

Short Descriptions

mktemp	Creates temporary files in a secure manner; it is used in scripts
tempfile	Creates temporary files in a less secure manner than **mktemp**; it is installed for backwards-compatibility

6.18. Iana-Etc-1.01

The Iana-Etc package provides data for network services and protocols.

Approximate build time: 0.1 SBU
Required disk space: 641 KB

Iana-Etc installation depends on: Make

6.18.1. Installation of Iana-Etc

Parse the data:

```
make
```

Install the package:

```
make install
```

6.18.2. Contents of Iana-Etc

Installed files: /etc/protocols and /etc/services

Short Descriptions

`/etc/protocols`	Describes the various DARPA Internet protocols that are available from the TCP/IP subsystem
`/etc/services`	Provides a mapping between friendly textual names for internet services, and their underlying assigned port numbers and protocol types

6.19. Findutils-4.1.20

The Findutils package contains programs to find files. Processes are provided to recursively search through a directory tree and to create, maintain, and search a database (often faster than the recursive find, but unreliable if the database has not been recently updated).

Approximate build time: 0.2 SBU
Required disk space: 7.5 MB

Findutils installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make and Sed

6.19.1. Installation of Findutils

Prepare Findutils for compilation:

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

The *localstatedir* directive above changes the location of the **locate** database to be in `/var/lib/locate`, which is FHS-compliant.

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

6.19.2. Contents of Findutils

Installed programs: bigram, code, find, frcode, locate, updatedb, and xargs

Short Descriptions

bigram	Was formerly used to produce **locate** databases
code	Was formerly used to produce **locate** databases; it is the ancestor of **frcode**.
find	Searches given directory trees for files matching the specified criteria
frcode	Is called by **updatedb** to compress the list of file names; it uses front-compression, reducing the database size by a factor of four to five.
locate	Searches through a database of file names and reports the names that contain a given string or match a given pattern
updatedb	Updates the **locate** database; it scans the entire file system (including other file systems that are currently mounted, unless told not to) and puts every file name it finds into the database
xargs	Can be used to apply a given command to a list of files

6.20. Gawk-3.1.4

The Gawk package contains programs for manipulating text files.

Approximate build time: 0.2 SBU
Required disk space: 17 MB

Gawk installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Sed

6.20.1. Installation of Gawk

Prepare Gawk for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

6.20.2. Contents of Gawk

Installed programs: awk (link to gawk), gawk, gawk-3.1.4, grcat, igawk, pgawk, pgawk-3.1.4, and pwcat

Short Descriptions

awk	A link to **gawk**
gawk	A program for manipulating text files; it is the GNU implementation of **awk**
gawk-3.1.4	A hard link to **gawk**
grcat	Dumps the group database `/etc/group`
igawk	Gives **gawk** the ability to include files
pgawk	The profiling version of **gawk**
pgawk-3.1.4	Hard link to **pgawk**
pwcat	Dumps the password database `/etc/passwd`

6.21. Ncurses-5.4

The Ncurses package contains libraries for terminal-independent handling of character screens.

Approximate build time: 0.6 SBU
Required disk space: 27 MB

Ncurses installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, and Sed

6.21.1. Installation of Ncurses

Prepare Ncurses for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Give the Ncurses libraries execute permissions:

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

Fix a library that should not be executable:

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

Move the libraries to the `/lib` directory, where they are expected to reside:

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

Because the libraries have been moved, a few symlinks are pointing to non-existent files. Recreate those symlinks:

```
ln -sf ../../lib/libncurses.so.5 /usr/lib/libncurses.so
ln -sf libncurses.so /usr/lib/libcurses.so
```

6.21.2. Contents of Ncurses

Installed programs: captoinfo (link to tic), clear, infocmp, infotocap (link to tic), reset (link to tset), tack, tic, toe, tput, and tset
Installed libraries: libcurses.[a,so] (link to libncurses.[a,so]), libform.[a,so], libmenu.[a,so], libncurses++.a, libncurses.[a,so], and libpanel.[a,so]

Short Descriptions

captoinfo	Converts a termcap description into a terminfo description
clear	Clears the screen, if possible
infocmp	Compares or prints out terminfo descriptions
infotocap	Converts a terminfo description into a termcap description
reset	Reinitializes a terminal to its default values
tack	The terminfo action checker; it is mainly used to test the accuracy of an entry in the terminfo database
tic	The terminfo entry-description compiler that translates a terminfo file from source format into the binary format needed for the ncurses library routines. A terminfo file contains information on the capabilities of a certain terminal
toe	Lists all available terminal types, giving the primary name and description for each
tput	Makes the values of terminal-dependent capabilities available to the shell; it can also be used to reset or initialize a terminal or report its long name
tset	Can be used to initialize terminals

`libcurses`	A link to `libncurses`
`libncurses`	Contains functions to display text in many complex ways on a terminal screen; a good example of the use of these functions is the menu displayed during the kernel's **make menuconfig**
`libform`	Contains functions to implement forms
`libmenu`	Contains functions to implement menus
`libpanel`	Contains functions to implement panels

6.22. Readline-5.0

The Readline package contains the Readline command-line library.

Approximate build time: 0.11 SBU
Required disk space: 3.8 MB

Readline installation depends on: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, and Sed

6.22.1. Installation of Readline

The following patch fixes a problem where Readline sometimes only shows 33 characters on a line and then wraps to the next line.

```
patch -Np1 -i ../readline-5.0-display_wrap-1.patch
```

Prepare Readline for compilation:

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

Compile the package:

```
make SHLIB_XLDFLAGS=-lncurses
```

The meaning of the make option:

SHLIB_XLDFLAGS=-lncurses
: This option forces Readline to link against the `libncurses` library.

Install the package:

```
make install
```

Give Readline's dynamic libraries more appropriate permissions:

```
chmod 755 /usr/lib/*.5.0
```

Move the dynamic libraries to a more appropriate location:

```
mv /usr/lib/lib{readline,history}.so.5* /lib
```

Because the libraries have been moved, a few symlinks are now pointing to non-existent files. Recreate those symlinks:

```
ln -sf ../../lib/libhistory.so.5 /usr/lib/libhistory.so
ln -sf ../../lib/libreadline.so.5 /usr/lib/libreadline.so
```

6.22.2. Contents of Readline

Installed libraries: libhistory.[a,so], and libreadline.[a,so]

Short Descriptions

`libhistory`	Provides a consistent user interface for recalling lines of history
`libreadline`	Aids in the consistency of user interface across discrete programs that need to provide a command line interface

6.23. Vim-6.3

The Vim package contains a powerful text editor.

Approximate build time: 0.4 SBU
Required disk space: 34 MB

Vim installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, and Sed

Alternatives to Vim

If you prefer another editor—such as Emacs, Joe, or Nano—please refer to *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html* for suggested installation instructions.

6.23.1. Installation of Vim

First, unpack both `vim-6.3.tar.bz2` and (optionally) `vim-6.3-lang.tar.gz` archives into the same directory. Then, change the default locations of the `vimrc` and `gvimrc` configuration files to `/etc`:

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
echo '#define SYS_GVIMRC_FILE "/etc/gvimrc"' >> src/feature.h
```

Prepare Vim for compilation:

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

The optional but highly recommended *`--enable-multibyte`* switch includes support for editing files in multibyte character encodings into **vim**. This is needed if using a locale with a multibyte character set. This switch is also helpful to be able to edit text files initially created in Linux distributions like Fedora Core that use UTF-8 as a default character set.

Compile the package:

```
make
```

To test the results, issue: **`make test`**. However, this test suite outputs a lot of chaotic characters to the screen, which can cause issues with the settings of the current terminal. Therefore, running the test suite here is optional.

Install the package:

```
make install
```

Many users are used to using **vi** instead of **vim**. To allow execution of **vim** when users habitually enter **vi**, create a symlink:

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

If the X Window System is going to be installed on the LFS system, it may be necessary to recompile Vim after installing X. Vim comes with a GUI version of the editor that requires X and some additional libraries to be installed. For more information on this process, refer to the Vim documentation and the Vim installation page in the BLFS book at *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html#postlfs-editors-vim*.

6.23.2. Configuring Vim

By default, **vim** runs in vi-incompatible mode. This may be new to users who have used other editors in the past. The “nocompatible” setting is included below to highlight the fact that a new behavior is being used. It also reminds those who would change to “compatible” mode that it should appear first. This is necessary because it changes other settings, and overrides must come after this setting. Create a default **vim** configuration file by running the following:

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

set nocompatible
set backspace=2
syntax on
if (&term == "iterm") || (&term == "putty")
  set background=dark
endif

" End /etc/vimrc
EOF
```

The `set nocompatible` makes **vim** behave in a more useful way (the default) than the vi-compatible manner. Remove the “no” to keep the old **vi** behavior. The `set backspace=2` allows backspacing over line breaks, autoindents, and the start of insert. The `syntax on` enables vim's syntax highlighting. Finally, the *if* statement with the `set background=dark` corrects **vim**'s guess about the background color of some terminal emulators. This gives the highlighting a better color scheme for use on the black background of these programs.

Documentation for other available options can be obtained by running the following command:

```
vim -c ':options'
```

6.23.3. Contents of Vim

Installed programs: efm_filter.pl, efm_perl.pl, ex (link to vim), less.sh, mve.awk, pltags.pl, ref, rview (link to vim), rvim (link to vim), shtags.pl, tcltags, vi (link to vim), view (link to vim), vim, vim132, vim2html.pl, vimdiff (link to vim), vimm, vimspell.sh, vimtutor, and xxd

Short Descriptions

efm_filter.pl	A filter for creating an error file that can be read by **vim**
efm_perl.pl	Reformats the error messages of the Perl interpreter for use with the “quickfix” mode of **vim**
ex	Starts **vim** in ex mode
less.sh	A script that starts **vim** with less.vim
mve.awk	Processes **vim** errors
pltags.pl	Creates a tags file for Perl code for use by **vim**
ref	Checks the spelling of arguments
rview	Is a restricted version of **view**; no shell commands can be started and **view** cannot be suspended
rvim	Is a restricted version of **vim**; no shell commands can be started and **vim** cannot be suspended
shtags.pl	Generates a tag file for Perl scripts

tcltags	Generates a tag file for TCL code
view	Starts **vim** in read-only mode
vi	Is the editor
vim	Is the editor
vim132	Starts **vim** with the terminal in 132-column mode
vim2html.pl	Converts Vim documentation to HypterText Markup Language (HTML)
vimdiff	Edits two or three versions of a file with **vim** and show differences
vimm	Enables the DEC locator input model on a remote terminal
vimspell.sh	Spells a file and generates the syntax statements necessary to highlight in **vim**. This script requires the old Unix **spell** command, which is provided neither in LFS nor in BLFS
vimtutor	Teaches the basic keys and commands of **vim**
xxd	Creates a hex dump of the given file; it can also do the reverse, so it can be used for binary patching

6.24. M4-1.4.2

The M4 package contains a macro processor.

Approximate build time: 0.1 SBU
Required disk space: 3.0 MB

M4 installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, and Sed

6.24.1. Installation of M4

Prepare M4 for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

6.24.2. Contents of M4

Installed program: m4

Short Descriptions

m4 copies the given files while expanding the macros that they contain. These macros are either built-in or user-defined and can take any number of arguments. Besides performing macro expansion, **m4** has built-in functions for including named files, running Unix commands, performing integer arithmetic, manipulating text, recursion, etc. The **m4** program can be used either as a front-end to a compiler or as a macro processor in its own right.

6.25. Bison-1.875a

The Bison package contains a parser generator.

Approximate build time: 0.6 SBU
Required disk space: 10.6 MB

Bison installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, and Sed

6.25.1. Installation of Bison

Prepare Bison for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

6.25.2. Contents of Bison

Installed programs: bison and yacc
Installed library: liby.a

Short Descriptions

bison generates, from a series of rules, a program for analyzing the structure of text files; Bison is a replacement for Yacc (Yet Another Compiler Compiler)

yacc a wrapper for **bison**, meant for programs that still call **yacc** instead of **bison**; it calls **bison** with the `-y` option

`liby.a` the Yacc library containing implementations of Yacc-compatible *yyerror* and *main* functions; this library is normally not very useful, but POSIX requires it

6.26. Less-382

The Less package contains a text file viewer.

Approximate build time: 0.1 SBU
Required disk space: 3.4 MB

Less installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, and Sed

6.26.1. Installation of Less

Prepare Less for compilation:

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

The meaning of the configure option:

--sysconfdir=/etc
: This option tells the programs created by the package to look in `/etc` for the configuration files.

Compile the package:

```
make
```

Install the package:

```
make install
```

6.26.2. Contents of Less

Installed programs: less, lessecho, and lesskey

Short Descriptions

less	a file viewer or pager; it displays the contents of the given file, letting the user scroll, find strings, and jump to marks
lessecho	needed to expand meta-characters, such as * and ?, in filenames on Unix systems
lesskey	used to specify the key bindings for **less**

6.27. Groff-1.19.1

The Groff package contains programs for processing and formatting text.

Approximate build time: 0.5 SBU
Required disk space: 43 MB

Groff installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, and Sed

6.27.1. Installation of Groff

Groff expects the environment variable `PAGE` to contain the default paper size. For users in the United States, *`PAGE=letter`* is appropriate. Elsewhere, *`PAGE=A4`* may be more suitable.

Prepare Groff for compilation:

```
PAGE=[paper_size] ./configure --prefix=/usr
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Some documentation programs, such as **xman**, will not work properly without the following symlinks:

```
ln -s soelim /usr/bin/zsoelim
ln -s eqn /usr/bin/geqn
ln -s tbl /usr/bin/gtbl
```

6.27.2. Contents of Groff

Installed programs: addftinfo, afmtodit, eqn, eqn2graph, geqn (link to eqn), grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (link to tbl), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pfbtops, pic, pic2graph, post-grohtml, pre-grohtml, refer, soelim, tbl, tfmtodit, troff, and zsoelim (link to soelim)

Short Descriptions

addftinfo	Reads a troff font file and adds some additional font-metric information that is used by the **groff** system
afmtodit	Creates a font file for use with **groff** and **grops**
eqn	Compiles descriptions of equations embedded within troff input files into commands that are understood by **troff**
eqn2graph	Converts a troff EQN (equation) into a cropped image
eqn	A link to **eqn**
grn	A **groff** preprocessor for gremlin files
grodvi	A driver for **groff** that produces TeX dvi format
groff	A front-end to the groff document formatting system; normally, it runs the **troff** program and a post-processor appropriate for the selected device
groffer	Displays groff files and man pages on X and tty terminals
grog	Reads files and guesses which of the **groff** options *`-e`*, *`-man`*, *`-me`*, *`-mm`*, *`-ms`*, *`-p`*, *`-s`*, and *`-t`* are required for printing files, and reports the **groff** command including those options
grolbp	Is a **groff** driver for Canon CAPSL printers (LBP-4 and LBP-8 series laser printers)
grolj4	Is a driver for **groff** that produces output in PCL5 format suitable for an HP Laserjet 4 printer
grops	Translates the output of GNU **troff** to PostScript

grotty	Translates the output of GNU **troff** into a form suitable for typewriter-like devices
gtbl	Is the GNU implementation of **tbl**
hpftodit	Creates a font file for use with **groff -Tlj4** from an HP-tagged font metric file
indxbib	Creates an inverted index for the bibliographic databases with a specified file for use with **refer**, **lookbib**, and **lkbib**
lkbib	Searches bibliographic databases for references that contain specified keys and reports any references found
lookbib	Prints a prompt on the standard error (unless the standard input is not a terminal), reads a line containing a set of keywords from the standard input, searches the bibliographic databases in a specified file for references containing those keywords, prints any references found on the standard output, and repeats this process until the end of input
mmroff	A simple preprocessor for **groff**
neqn	Formats equations for American Standard Code for Information Interchange (ASCII) output
nroff	A script that emulates the **nroff** command using **groff**
pfbtops	Translates a PostScript font in `.pfb` format to ASCII
pic	Compiles descriptions of pictures embedded within troff or TeX input files into commands understood by TeX or **troff**
pic2graph	Converts a PIC diagram into a cropped image
post-grohtml	Translates the output of GNU **troff** to html
pre-grohtml	Translates the output of GNU **troff** to html
refer	Copies the contents of a file to the standard output, except that lines between *.[* and *.]* are interpreted as citations, and lines between *.R1* and *.R2* are interpreted as commands for how citations are to be processed
soelim	Reads files and replaces lines of the form *.so file* by the contents of the mentioned *file*

tbl	Compiles descriptions of tables embedded within troff input files into commands that are understood by **troff**
tfmtodit	Creates a font file for use with **groff -Tdvi**
troff	Is highly compatible with Unix **troff**; it should usually be invoked using the **groff** command, which will also run preprocessors and post-processors in the appropriate order and with the appropriate options
zsoelim	Is the GNU implementation of **soelim**

6.28. Sed-4.1.2

The Sed package contains a stream editor.

Approximate build time: 0.2 SBU
Required disk space: 5.2 MB

Sed installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Texinfo

6.28.1. Installation of Sed

Prepare Sed for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

6.28.2. Contents of Sed

Installed program: sed

Short Descriptions

sed Filters and transforms text files in a single pass

6.29. Flex-2.5.31

The Flex package contains a utility for generating programs that recognize patterns in text.

Approximate build time: 0.1 SBU
Required disk space: 3.4 MB

Flex installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, and Sed

6.29.1. Installation of Flex

Flex contains several known bugs. Fix these with the following patch:

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

The GNU autotools detects that the Flex source code has been modified by the previous patch and tries to update the manual page accordingly. This does not work correctly on many systems, and the default page is fine, so make sure it does not get regenerated:

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

Prepare Flex for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

There are some packages that expect to find the `lex` library in `/usr/lib`. Create a symlink to account for this:

```
ln -s libfl.a /usr/lib/libl.a
```

A few programs do not know about **flex** yet and try to run its predecessor, **lex**. To support those programs, create a wrapper script named `lex` that calls `flex` in **lex** emulation mode:

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Begin /usr/bin/lex

exec /usr/bin/flex -l "$@"

# End /usr/bin/lex
EOF
chmod 755 /usr/bin/lex
```

6.29.2. Contents of Flex

Installed programs: flex, flex++ (link to flex), and lex
Installed library: libfl.a

Short Descriptions

flex	A tool for generating programs that recognize patterns in text; it allows for the versatility to specify the rules for pattern-finding, eradicating the need to develop a specialized program
flex++	Invokes a version of **flex** that is used exclusively for C++ scanners
lex	Script that runs **flex** in **lex** emulation mode
`libfl.a`	The `flex` library

6.30. Gettext-0.14.1

The Gettext package contains utilities for internationalization and localization. These allow programs to be compiled with NLS, enabling them to output messages in the user's native language.

Approximate build time: 0.5 SBU
Required disk space: 55 MB

Gettext installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, and Sed

6.30.1. Installation of Gettext

Prepare Gettext for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**. This takes a very long time, around 7 SBUs.

Install the package:

```
make install
```

6.30.2. Contents of Gettext

Installed programs: autopoint, config.charset, config.rpath, envsubst, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, and xgettext
Installed libraries: libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] and libgettextsrc[a,so]

Short Descriptions

autopoint	Copies standard Gettext infrastructure files into a source package
config.charset	Outputs a system-dependent table of character encoding aliases
config.rpath	Outputs a system-dependent set of variables, describing how to set the runtime search path of shared libraries in an executable
envsubst	Substitutes environment variables in shell format strings
gettext	Translates a natural language message into the user's language by looking up the translation in a message catalog
gettextize	Copies all standard Gettext files into the given top-level directory of a package to begin internationalizing it
hostname	Displays a network hostname in various forms
msgattrib	Filters the messages of a translation catalog according to their attributes and manipulates the attributes
msgcat	Concatenates and merges the given `.po` files
msgcmp	Compares two `.po` files to check that both contain the same set of msgid strings
msgcomm	Finds the messages that are common to to the given `.po` files
msgconv	Converts a translation catalog to a different character encoding
msgen	Creates an English translation catalog
msgexec	Applies a command to all translations of a translation catalog
msgfilter	Applies a filter to all translations of a translation catalog
msgfmt	Generates a binary message catalog from from a translation catalog
msggrep	Extracts all messages of a translation catalog that match a given pattern or belong to some given source files
msginit	Creates a new `.po` file, initializing the meta information with values from the user's environment
msgmerge	Combines two raw translations into a single file

msgunfmt	Decompiles a binary message catalog into raw translation text
msguniq	Unifies duplicate translations in a translation catalog
ngettext	Displays native language translations of a textual message whose grammatical form depends on a number
xgettext	Extracts the translatable message lines from the given source files to make the first translation template
`libasprintf`	defines the *autosprintf* class, which makes C formatted output routines usable in C++ programs, for use with the <*string*> strings and the <*iostream*> streams
`libgettextlib`	a private library containing common routines used by the various Gettext programs; these are not intended for general use
`libgettextpo`	Used to write specialized programs that process `.po` files; this library is used when the standard applications shipped with Gettext (such as **msgcomm**, **msgcmp**, **msgattrib**, and **msgen**) will not suffice
`libgettextsrc`	A private library containing common routines used by the various Gettext programs; these are not intended for general use

6.31. Inetutils-1.4.2

The Inetutils package contains programs for basic networking.

Approximate build time: 0.2 SBU
Required disk space: 11 MB

Inetutils installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, and Sed

6.31.1. Installation of Inetutils

Inetutils has issues with the Linux 2.6 kernel series. Fix these issues by applying the following patch:

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

All programs that come with Inetutils will not be installed. However, the Inetutils build system will insist on installing all the man pages anyway. The following patch will correct this situation:

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

Prepare Inetutils for compilation:

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

The meaning of the configure options:

--disable-logger
: This option prevents Inetutils from installing the **logger** program, which is used by scripts to pass messages to the System Log Daemon. Do not install it because Util-linux installs a better version later.

--disable-syslogd
: This option prevents Inetutils from installing the System Log Daemon, which is installed with the Sysklogd package.

--disable-whois
This option disables the building of the Inetutils **whois** client, which is out of date. Instructions for a better **whois** client are in the BLFS book.

--disable-servers
This disables the installation of the various network servers included as part of the Inetutils package. These servers are deemed not appropriate in a basic LFS system. Some are insecure by nature and are only considered safe on trusted networks. More information can be found at *http://www.linuxfromscratch.org/blfs/view/svn/basicnet/inetutils.html*. Note that better replacements are available for many of these servers.

Compile the package:

```
make
```

Install the package:

```
make install
```

Move the **ping** program to its FHS-compliant place:

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

6.31.2. Contents of Inetutils

Installed programs: ftp, ping, rcp, rlogin, rsh, talk, telnet, and tftp

Short Descriptions

ftp	Is the file transfer protocol program
ping	Sends echo-request packets and reports how long the replies take
rcp	Performs remote file copy
rlogin	Performs remote login
rsh	Runs a remote shell
talk	Is used to chat with another user
telnet	An interface to the TELNET protocol
tftp	A trivial file transfer program

6.32. Iproute2-2.6.8-040823

The Iproute2 package contains programs for basic and advanced IPV4-based networking.

Approximate build time: 0.1 SBU
Required disk space: .6 MB

Iproute2 installation depends on: GCC, Glibc, Make, Linux-Headers, and Sed

6.32.1. Installation of Iproute2

The **arpd** binary included in this package is dependent on Berkeley DB. Because **arpd** is not a very common requirement on a base Linux system, remove the dependency on Berkeley DB by applying the patch using the command below. If the **arpd** binary is needed, instructions for compiling Berkeley DB can be found in the BLFS Book at *http://www.linuxfromscratch.org/blfs/view/svn/content/databases.html#db*.

```
patch -Np1 -i ../iproute2-2.6.8_040823-remove_db-1.patch
```

Prepare Iproute2 for compilation:

```
./configure
```

Compile the package:

```
make SBINDIR=/sbin
```

The meaning of the make option:

SBINDIR=/sbin
    This makes sure that the Iproute2 binaries will install into `/sbin`. This is the correct location according to the FHS, because some of the Iproute2 binaries are used in the bootscripts.

Install the package:

```
make SBINDIR=/sbin install
```

6.32.2. Contents of Iproute2

Installed programs: ifstat, ip, nstat, routef, routel, rtmon, rtstat, ss, and tc.

Short Descriptions

ifstat Shows the interfaces statistic, including the amount of transmitted and received packages by interface.

ip The main executable. It has several different functions:

ip link ***`[device]`*** allows users to look at the state of devices and to make changes.

ip addr allows users to look at addresses and their properties, add new addresses, and delete old ones.

ip neighbor allows users to look at neighbor bindings and their properties, add new neighbor entries, and delete old ones.

ip rule allows users to look at the routing policies and change them.

ip route allows users to look at the routing table and change routing table rules.

ip tunnel allows users to look at the IP tunnels and their properties, and change them.

ip maddr allows users to look at the multicast addresses and their properties, and change them.

ip mroute allows users to set, change, or delete the multicast routing.

ip monitor allows users to continously monitor the state of devices, addresses and routes.

nstat Shows network statistics.

routef A component of **ip route**. This is for flushing the routing tables.

routel A component of **ip route**. This is for listing the routing tables.

rtmon Route monitoring utility.

rtstat Route status utility

ss Similar to the **netstat** command; shows active connections

tc Traffic Controlling Executable; this is for Quality Of Service (QOS) and Class Of Service (COS) implementations

tc qdisc allows users to setup the queueing discipline

tc class allows users to setup classes based on the queuing discipline scheduling

tc estimator allows users to estimate the network flow into a network

tc filter allows users to setup the QOS/COS packet filtering

tc policy allows users to setup the QOS/COS policies

6.33. Perl-5.8.5

The Perl package contains the Practical Extraction and Report Language.

Approximate build time: 2.9 SBU
Required disk space: 143 MB

Perl installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, and Sed

6.33.1. Installation of Perl

To have full control over the way Perl is set up, run the interactive **Configure** script and hand-pick the way this package is built. If the defaults it auto-detects are suitable, prepare Perl for compilation with:

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

The meaning of the configure option:

-Dpager="/bin/less -isR"
: This corrects an error in the **perldoc** code with the invocation of the **less** program.

Compile the package:

```
make
```

To run the test suite, first create a basic `/etc/hosts` file which is needed by a couple of tests to resolve the network name localhost:

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

Now run the tests, if desired:

```
make test
```

Install the package:

```
make install
```

6.33.2. Contents of Perl

Installed programs: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.5 (link to perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (link to s2p), pstruct (link to c2ph), s2p, splain, and xsubpp
Installed libraries: Several hundred which cannot all be listed here

Short Descriptions

a2p	Translates awk to Perl
c2ph	Dumps C structures as generated from **cc -g -S**
dprofpp	Displays Perl profile data
en2cxs	Builds a Perl extension for the Encode module from either Unicode Character Mappings or Tcl Encoding Files
find2perl	Translates **find** commands to Perl
h2ph	Converts `.h` C header files to `.ph` Perl header files
h2xs	Converts `.h` C header files to Perl extensions
libnetcfg	Can be used to configure the `libnet`
perl	Combines some of the best features of C, sed, awk and sh into a single swiss-army language
perl5.8.5	A hard link to **perl**
perlbug	Used to generate bug reports about Perl, or the modules that come with it, and mail them
perlcc	Generates executables from Perl programs
perldoc	Displays a piece of documentation in pod format that is embedded in the Perl installation tree or in a Perl script
perlivp	The Perl Installation Verification Procedure; it can be used to verify that Perl and its libraries have been installed correctly
piconv	A Perl version of the character encoding converter **iconv**

pl2pm	A rough tool for converting Perl4 `.pl` files to Perl5 `.pm` modules
pod2html	Converts files from pod format to HTML format
pod2latex	Converts files from pod format to LaTeX format
pod2man	Converts pod data to formatted *roff input
pod2text	Converts pod data to formatted ASCII text
pod2usage	Prints usage messages from embedded pod docs in files
podchecker	Checks the syntax of pod format documentation files
podselect	Displays selected sections of pod documentation
psed	A Perl version of the stream editor **sed**
pstruct	Dumps C structures as generated from **cc -g -S** stabs
s2p	Translates sed to Perl
splain	Is used to force verbose warning diagnostics in Perl
xsubpp	Converts Perl XS code into C code

6.34. Texinfo-4.7

The Texinfo package contains programs for reading, writing, and converting Info documents.

Approximate build time: 0.2 SBU
Required disk space: 17 MB

Texinfo installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, and Sed

6.34.1. Installation of Texinfo

The following patch fixes a problem where the **info** program sometimes crashes when hitting the *Delete* key on the keyboard:

```
patch -Np1 -i ../texinfo-4.7-segfault-1.patch
```

Prepare Texinfo for compilation:

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

Compile the package:

```
make
```

To test the results, issue: `make check`.

Install the package:

```
make install
```

Optionally, install the components belonging in a TeX installation:

```
make TEXMF=/usr/share/texmf install-tex
```

The meaning of the make parameter:

`TEXMF=/usr/share/texmf`
: The `TEXMF` makefile variable holds the location of the root of the TeX tree if, for example, a TeX package will be installed later.

The Info documentation system uses a plain text file to hold its list of menu entries. The file is located at `/usr/share/info/dir`. Unfortunately, due to occasional problems in the Makefiles of various packages, it can sometimes get out of step with the Info manuals installed on the system. If the `/usr/share/info/dir` file ever needs to be recreated, the following optional commands will accomplish the task:

```
cd /usr/share/info
rm dir
for f in *
do install-info $f dir 2>/dev/null
done
```

6.34.2. Contents of Texinfo

Installed programs: info, infokey, install-info, makeinfo, texi2dvi, and texindex

Short Descriptions

info	Used to read Info documents which are similar to man pages, but often go much deeper than just explaining all the command line options. For example, compare **man bison** and **info bison**.
infokey	Compiles a source file containing Info customizations into a binary format
install-info	Used to install Info files; it updates entries in the Info index file
makeinfo	Translates the given Texinfo source documents into info files, plain text, or HTML
texi2dvi	Used to format the given Texinfo document into a device-independent file that can be printed
texindex	Used to sort Texinfo index files

6.35. Autoconf-2.59

The Autoconf package contains programs for producing shell scripts that can automatically configure source code.

Approximate build time: 0.5 SBU
Required disk space: 7.7 MB

Autoconf installation depends on: Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, and Sed

6.35.1. Installation of Autoconf

Prepare Autoconf for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**. This takes a long time, about 2 SBUs.

Install the package:

```
make install
```

6.35.2. Contents of Autoconf

Installed programs: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate, and ifnames

Short Descriptions

autoconf Produces shell scripts that automatically configure software source code packages to adapt to many kinds of Unix-like systems. The configuration scripts it produces are independent—running them does not require the **autoconf** program.

autoheader A tool for creating template files of C *#define* statements for configure to use

autom4te A wrapper for the M4 macro processor

autoreconf Automatically runs **autoconf**, **autoheader**, **aclocal**, **automake**, **gettextize**, and **libtoolize** in the correct order to save time when changes are made to **autoconf** and **automake** template files

autoscan Helps to create a `configure.in` file for a software package; it examines the source files in a directory tree, searching them for common portability issues, and creates a `configure.scan` file that serves as as a preliminary `configure.in` file for the package

autoupdate Modifies a `configure.in` file that still calls **autoconf** macros by their old names to use the current macro names

ifnames Helps when writing `configure.in` files for a software package; it prints the identifiers that the package uses in C preprocessor conditionals. If a package has already been set up to have some portability, this program can help determine what **configure** needs to check for. It can also fill in gaps in a `configure.in` file generated by **autoscan**

6.36. Automake-1.9.1

The Automake package contains programs for generating Makefiles for use with Autoconf.

Approximate build time: 0.2 SBU
Required disk space: 6.8 MB

Automake installation depends on: Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, and Sed

6.36.1. Installation of Automake

Prepare Automake for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**. This takes a long time, about 5 SBUs.

Install the package:

```
make install
```

6.36.2. Contents of Automake

Installed programs: acinstall, aclocal, aclocal-1.9.1, automake, automake-1.9.1, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, symlink-tree, and ylwrap

Short Descriptions

acinstall	A script that installs aclocal-style M4 files
aclocal	Generates `aclocal.m4` files based on the contents of `configure.in` files
aclocal-1.9.1	A hard link to **aclocal**

automake	A tool for automatically generating `Makefile.in` files from `Makefile.am` files. To create all the `Makefile.in` files for a package, run this program in the top-level directory. By scanning the `configure.in` file, it automatically finds each appropriate `Makefile.am` file and generate the corresponding `Makefile.in` file
automake-1.9.1	A hard link to **automake**
compile	A wrapper for compilers
config.guess	A script that attempts to guess the canonical triplet for the given build, host, or target architecture
config.sub	A configuration validation subroutine script
depcomp	A script for compiling a program so that dependency information is generated in addition to the desired output
elisp-comp	Byte-compiles Emacs Lisp code
install-sh	A script that installs a program, script, or data file
mdate-sh	A script that prints the modification time of a file or directory
missing	A script acting as a common stub for missing GNU programs during an installation
mkinstalldirs	A script that creates a directory tree
py-compile	Compiles a Python program
symlink-tree	A script to create a symlink tree of a directory tree
ylwrap	A wrapper for **lex** and **yacc**

6.37. Bash-3.0

The Bash package contains the Bourne-Again SHell.

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

Bash installation depends on: Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, and Sed.

6.37.1. Installation of Bash

The following patch is only necessary if Readline was not installed as suggested. This patch fixes a problem where Bash sometimes limits 33 characters to a line before wrapping to the next line. If Readline has been installed per the instructions, this patch is not necessary because the patch applied to the Readline package already resolves this issue.

```
patch -Np1 -i ../bash-3.0-display_wrap-1.patch
```

Prepare Bash for compilation:

```
./configure --prefix=/usr --bindir=/bin \
    --without-bash-malloc --with-installed-readline
```

The meaning of the configure option:

--with-installed-readline
: This options tells Bash to use the `readline` library that is already installed on the system rather than using its own readline version.

Compile the package:

```
make
```

To test the results, issue: **make tests**.

Install the package:

```
make install
```

Run the newly compiled **bash** program (replacing the one that is currently being executed):

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

> **Note**
>
> The parameters used make the **bash** process an interactive login shell and continue to disable hashing so that new programs are found as they become available.

6.37.2. Contents of Bash

Installed programs: bash, bashbug, and sh (link to bash)

Short Descriptions

bash A widely-used command interpreter; it performs many types of expansions and substitutions on a given command line before executing it, thus making this interpreter a powerful tool

bashbug A shell script to help the user compose and mail bug reports concerning **bash** in a standard format

sh A symlink to the **bash** program; when invoked as **sh**, **bash** tries to mimic the startup behavior of historical versions of **sh** as closely as possible, while conforming to the POSIX standard as well

6.38. File-4.10

The File package contains a utility for determining the type of files.

Approximate build time: 0.1 SBU
Required disk space: 6.3 MB

File installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, and Zlib

6.38.1. Installation of File

Prepare File for compilation:

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

Compile the package:

```
make
```

Install the package:

```
make install
```

6.38.2. Contents of File

Installed programs: file
Installed library: libmagic.[a,so]

Short Descriptions

file	Tries to classify each given file; it does this by performing several tests—file system tests, magic number tests, and language tests
`libmagic`	Contains routines for magic number recognition, used by the **file** program

6.39. Libtool-1.5.8

The Libtool package contains the GNU generic library support script. It wraps the complexity of using shared libraries in a consistent, portable interface.

Approximate build time: 1.5 SBU
Required disk space: 20 MB

Libtool installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, and Sed

6.39.1. Installation of Libtool

Prepare Libtool for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

6.39.2. Contents of Libtool

Installed programs: libtool and libtoolize
Installed libraries: libltdl.[a,so]

Short Descriptions

libtool	Provides generalized library-building support services
libtoolize	Provides a standard way to add **libtool** support to a package
libltdl	Hides the various difficulties of dlopening libraries

6.40. Bzip2-1.0.2

The Bzip2 package contains programs for compressing and decompressing files. Text files yield a much better compression than with the traditional **gzip**.

Approximate build time: 0.1 SBU
Required disk space: 3.0 MB

Bzip2 installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, and Make

6.40.1. Installation of Bzip2

Prepare Bzip2 for compilation with:

```
make -f Makefile-libbz2_so
make clean
```

The *-f* flag will cause Bzip2 to be built using a different `Makefile` file, in this case the `Makefile-libbz2_so` file, which creates a dynamic `libbz2.so` library and links the Bzip2 utilities against it.

Compile the package:

```
make
```

If reinstalling Bzip2, perform **`rm -f /usr/bin/bz*`** first, otherwise the following **make install** will fail.

Install the programs:

```
make install
```

Install the shared **bzip2** binary into the `/bin` directory, make some necessary symbolic links, and clean up:

```
cp bzip2-shared /bin/bzip2
cp -a libbz2.so* /lib
ln -s ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm /usr/bin/{bunzip2,bzcat,bzip2}
ln -s bzip2 /bin/bunzip2
ln -s bzip2 /bin/bzcat
```

6.40.2. Contents of Bzip2

Installed programs: bunzip2 (link to bzip2), bzcat (link to bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless, and bzmore
Installed libraries: libbz2.a, libbz2.so (link to libbz2.so.1.0), libbz2.so.1.0 (link to libbz2.so.1.0.2), and libbz2.so.1.0.2

Short Descriptions

bunzip2	Decompresses bzipped files
bzcat	Decompresses to standard output
bzcmp	Runs **cmp** on bzipped files
bzdiff	Runs **diff** on bzipped files
bzgrep	Runs **grep** on bzipped files
bzegrep	Runs **egrep** on bzipped files
bzfgrep	Runs **fgrep** on bzipped files
bzip2	Compresses files using the Burrows-Wheeler block sorting text compression algorithm with Huffman coding; the compression rate is better than that achieved by more conventional compressors using “Lempel-Ziv” algorithms, like **gzip**
bzip2recover	Tries to recover data from damaged bzipped files
bzless	Runs **less** on bzipped files
bzmore	Runs **more** on bzipped files
`libbz2*`	The library implementing lossless, block-sorting data compression, using the Burrows-Wheeler algorithm

6.41. Diffutils-2.8.1

The Diffutils package contains programs that show the differences between files or directories.

Approximate build time: 0.1 SBU
Required disk space: 7.5 MB

Diffutils installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Sed

6.41.1. Installation of Diffutils

Prepare Diffutils for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

6.41.2. Contents of Diffutils

Installed programs: cmp, diff, diff3, and sdiff

Short Descriptions

cmp	Compares two files and reports whether or in which bytes they differ
diff	Compares two files or directories and reports which lines in the files differ
diff3	Compares three files line by line
sdiff	Merges two files and interactively outputs the results

6.42. Kbd-1.12

The Kbd package contains key-table files and keyboard utilities.

Approximate build time: 0.1 SBU
Required disk space: 12 MB

Kbd installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, Gzip, M4, Make, and Sed

6.42.1. Installation of Kbd

Prepare Kbd for compilation:

```
./configure
```

Compile the package:

```
make
```

Install the package:

```
make install
```

6.42.2. Contents of Kbd

Installed programs: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, getunimap, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (link to psfxtable), psfgettable (link to psfxtable), psfstriptable (link to psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setlogcons, setmetamode, setvesablank, showconsolefont, showkey, unicode_start, and unicode_stop

Short Descriptions

chvt	Changes the foreground virtual terminal
deallocvt	Deallocates unused virtual terminals
dumpkeys	Dumps the keyboard translation tables
fgconsole	Prints the number of the active virtual terminal

getkeycodes	Prints the kernel scancode-to-keycode mapping table
getunimap	Prints the currently used unimap
kbd_mode	Reports or sets the keyboard mode
kbdrate	Sets the keyboard repeat and delay rates
loadkeys	Loads the keyboard translation tables
loadunimap	Loads the kernel unicode-to-font mapping table
mapscrn	An obsolete program that used to load a user-defined output character mapping table into the console driver; this is now done by **setfont**
openvt	Starts a program on a new virtual terminal (VT)
psfaddtable	A link to **psfxtable**
psfgettable	A link to **psfxtable**
psfstriptable	A link to **psfxtable**
psfxtable	Handle Unicode character tables for console fonts
resizecons	Changes the kernel idea of the console size
setfont	Changes the Enhanced Graphic Adapter (EGA) and Video Graphics Array (VGA) fonts on the console
setkeycodes	Loads kernel scancode-to-keycode mapping table entries; this is useful if there are unusual keys on the keyboard
setleds	Sets the keyboard flags and Light Emitting Diodes (LEDs)
setlogcons	Sends kernel messages to the console
setmetamode	Defines the keyboard meta-key handling
setvesablank	Lets the user adjust the built-in hardware screensaver (a blank screen)
showconsolefont	Shows the current EGA/VGA console screen font
showkey	Reports the scancodes, keycodes, and ASCII codes of the keys pressed on the keyboard

unicode_start Puts the keyboard and console in UNICODE mode. Never use it on LFS, because applications are not configured to support UNICODE.

unicode_stop Reverts keyboard and console from UNICODE mode

6.43. E2fsprogs-1.35

The E2fsprogs package contains the utilities for handling the `ext2` file system. It also supports the `ext3` journaling file system.

Approximate build time: 0.6 SBU
Required disk space: 4.9 MB

E2fsprogs installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, and Texinfo

6.43.1. Installation of E2fsprogs

It is recommended that E2fsprogs be built in a subdirectory of the source tree:

```
mkdir build
cd build
```

Prepare E2fsprogs for compilation:

```
../configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs --disable-evms
```

The meaning of the configure options:

`--with-root-prefix=""`
Certain programs (such as the **e2fsck** program) are considered essential programs. When, for example, `/usr` is not mounted, these essential programs need to be available. They belong in directories like `/lib` and `/sbin`. If this option is not passed to E2fsprogs' configure, the programs are installed into the `/usr` directory, which is not where they should be.

`--enable-elf-shlibs`
This creates the shared libraries which some programs in this package use.

`--disable-evms`
This disables the building of the Enterprise Volume Management System (EVMS) plugin. This plugin is not up-to-date with the latest EVMS internal interfaces and EVMS is not installed as part of a base LFS system, so the plugin is not required. See the EVMS website at *http://evms.sourceforge.net/* for more information regarding EVMS.

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install most of the package:

```
make install
```

Install the shared libraries:

```
make install-libs
```

6.43.2. Contents of E2fsprogs

Installed programs: 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, and uuidgen.
Installed libraries: libblkid.[a,so], libcom_err.[a,so], libe2p.[a,so], libext2fs.[a,so], libss.[a,so], and libuuid.[a,so]

Short Descriptions

badblocks	Searches a device (usually a disk partition) for bad blocks
blkid	A command line utility to locate and print block device attributes
chattr	Changes the attributes of files on an `ext2` file system; it also changes `ext3` file systems, the journaling version of `ext2` file systems
compile_et	An error table compiler; it converts a table of error-code names and messages into a C source file suitable for use with the `com_err` library
debugfs	A file system debugger; it can be used to examine and change the state of an `ext2` file system
dumpe2fs	Prints the super block and blocks group information for the file system present on a given device

e2fsck	Is used to check, and optionally repair `ext2` file systems and `ext3` file systems
e2image	Is used to save critical `ext2` file system data to a file
e2label	Displays or changes the file system label on the `ext2` file system present on a given device
findfs	Finds a file system by label or Universally Unique Identifier (UUID)
fsck	Is used to check, and optionally repair, file systems
fsck.ext2	By default checks `ext2` file systems
fsck.ext3	By default checks `ext3` file systems
logsave	Saves the output of a command in a log file
lsattr	Lists the attributes of files on a second extended file system
mk_cmds	Converts a table of command names and helps messages into a C source file suitable for use with the `libss` subsystem library
mke2fs	Is used to create a second extended file system on the given device
mkfs.ext2	By default creates `ext2` file systems
mkfs.ext3	By default creates `ext3` file systems
mklost+found	Used to create a `lost+found` directory on an `ext2` file system; it pre-allocates disk blocks to this directory to lighten the task of **e2fsck**
resize2fs	Can be used to enlarge or shrink an `ext2` file system
tune2fs	Adjusts tunable file system parameters on an `ext2` file system
uuidgen	Creates new UUIDs. Each new UUID can reasonably be considered unique among all UUIDs created, on the local system and on other systems, in the past and in the future
`libblkid`	Contains routines for device identification and token extraction
`libcom_err`	The common error display routine
`libe2p`	Used by **dumpe2fs**, **chattr**, and **lsattr**

`libext2fs`	Contains routines to enable user-level programs to manipulate an `ext2` file system
`libss`	Used by **debugfs**
`libuuid`	Contains routines for generating unique identifiers for objects that may be accessible beyond the local system

6.44. Grep-2.5.1

The Grep package contains programs for searching through files.

Approximate build time: 0.1 SBU
Required disk space: 5.8 MB

Grep installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed, and Texinfo

6.44.1. Installation of Grep

Prepare Grep for compilation:

```
./configure --prefix=/usr --bindir=/bin --with-included-regex
```

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

6.44.2. Contents of Grep

Installed programs: egrep (link to grep), fgrep (link to grep), and grep

Short Descriptions

egrep Prints lines matching an extended regular expression

fgrep Prints lines matching a list of fixed strings

grep Prints lines matching a basic regular expression

6.45. Grub-0.95

The Grub package contains the Grand Unified Bootloader.

Approximate build time: 0.2 SBU
Required disk space: 10 MB

Grub installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, and Sed

6.45.1. Installation of Grub

This package is known to have issues when its default optimization flags (including the *`-march`* and *`-mcpu`* options) are changed. If any environment variables that override default optimizations have been defined, such as `CFLAGS` and `CXXFLAGS`, unset them when building Grub.

Prepare Grub for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Note that the test results will always show the error “ufs2_stage1_5 is too big.” This is due to a compiler issue, but can be ignored unless you plan to boot from an UFS partition. The partitions are normally only used by Sun workstations.

Install the package:

```
make install
mkdir /boot/grub
cp /usr/share/grub/i386-pc/stage{1,2} /boot/grub
```

Replace `i386-pc` with whatever directory is appropriate for the hardware in use.

The `i386-pc` directory contains a number of `*stage1_5` files, different ones for different file systems. Review the files available and copy the appropriate ones to the `/boot/grub` directory. Most users will copy the `e2fs_stage1_5` and/or `reiserfs_stage1_5` files.

6.45.2. Contents of Grub

Installed programs: grub, grub-install, grub-md5-crypt, grub-terminfo, and mbchk

Short Descriptions

grub	The Grand Unified Bootloader's command shell
grub-install	Installs GRUB on the given device
grub-md5-crypt	Encrypts a password in MD5 format
grub-terminfo	Generates a terminfo command from a terminfo name; it can be employed if an unknown terminal is being used
mbchk	Checks the format of a multi-boot kernel

6.46. Gzip-1.3.5

The Gzip package contains programs for compressing and decompressing files.

Approximate build time: 0.1 SBU
Required disk space: 2.6 MB

Gzip installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, and Sed

6.46.1. Installation of Gzip

Prepare Gzip for compilation:

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

The **gzexe** script has the location of the **gzip** binary hard-wired into it. Because the location of the binary is changed later, the following command ensures that the new location gets placed into the script:

```
sed -i 's@"BINDIR"@/bin@g' gzexe.in
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Move the programs to the `/bin` directory:

```
mv /usr/bin/gzip /bin
rm /usr/bin/{gunzip,zcat}
ln -s gzip /bin/gunzip
ln -s gzip /bin/zcat
ln -s gunzip /bin/uncompress
```

6.46.2. Contents of Gzip

Installed programs: gunzip (link to gzip), gzexe, gzip, uncompress (link to gunzip), zcat (link to gzip), zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore, and znew

Short Descriptions

gunzip	Decompresses gzipped files
gzexe	Creates self-uncompressing executable files
gzip	Compresses the given files using Lempel-Ziv (LZ77) coding
uncompress	Decompresses compressed files
zcat	Uncompresses the given gzipped files to standard output
zcmp	Runs **cmp** on gzipped files
zdiff	Runs **diff** on gzipped files
zegrep	Runs **egrep** on gzipped files
zfgrep	Runs **fgrep** on gzipped files
zforce	Forces a `.gz` extension on all given files that are gzipped files, so that **gzip** will not compress them again; this can be useful when file names were truncated during a file transfer
zgrep	Runs **grep** on gzipped files
zless	Runs **less** on gzipped files
zmore	Runs **more** on gzipped files
znew	Re-compresses files from **compress** format to **gzip** format—`.Z` to `.gz`

6.47. Man-1.5o

The Man package contains programs for finding and viewing manual pages.

Approximate build time: 0.1 SBU
Required disk space: 1.9MB

Man installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, and Sed

6.47.1. Installation of Man

Three adjustments need to be made to the sources of Man.

The first is a patch which allows Man to work better with recent releases of Groff. In particular, man pages will now display using the full terminal width instead of being limited to 80 characters:

```
patch -Np1 -i ../man-1.5o-80cols-1.patch
```

The second is a sed substitution to add the *-R* switch to the `PAGER` variable so that escape sequences are properly handled by Less:

```
sed -i 's@-is@&R@g' configure
```

The third is also a sed substitution to comment out the “MANPATH /usr/man” line in the `man.conf` file to prevent redundant results when using programs such as **whatis**:

```
sed -i 's@MANPATH./usr/man@#&@g' src/man.conf.in
```

Prepare Man for compilation:

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

The meaning of the configure options:

-confdir=/etc
: This tells the **man** program to look for the `man.conf` configuration file in the `/etc` directory.

Compile the package:

```
make
```

Install the package:

```
make install
```

Note

To disable Select Graphic Rendition (SGR) escape sequences, edit the `man.conf` file and add the *`-c`* switch to the `NROFF` variable.

If the character set uses 8-bit characters, search for the line beginning with “NROFF” in `/etc/man.conf`, and verify that it looks as follows:

```
NROFF  /usr/bin/nroff -Tlatin1 -mandoc
```

Note that “latin1” should be used even if it is not the character set of the locale. The reason is that, according to the specification, **groff** has no means of typesetting characters outside International Organization for Standards (ISO) 8859-1 without some strange escape codes. When formatting manual pages, **groff** thinks that they are in the ISO 8859-1 encoding and this *`-Tlatin1`* switch tells **groff** to use the same encoding for output. Since **groff** does no recoding of input characters, the formatted result is really in the same encoding as input, and therefore it is usable as the input for a pager.

This does not solve the problem of a non-working **man2dvi** program for localized manual pages in non-ISO 8859-1 locales. Also, it does not work with multibyte character sets. The first problem does not currently have a solution. The second issue is not of concern because the LFS installation does not support multibyte character sets.

Additional information with regards to the compression of man and info pages can be found in the BLFS book at *http://www.linuxfromscratch.org/blfs/view/cvs/postlfs/compressdoc.html*.

6.47.2. Contents of Man

Installed programs: apropos, makewhatis, man, man2dvi, man2html, and whatis

Short Descriptions

apropos	Searches the whatis database and displays the short descriptions of system commands that contain a given string
makewhatis	Builds the whatis database; it reads all the manual pages in the manpath and writes the name and a short description in the whatis database for each page
man	Formats and displays the requested on-line manual page
man2dvi	Converts a manual page into dvi format
man2html	Converts a manual page into HTML
whatis	Searches the whatis database and displays the short descriptions of system commands that contain the given keyword as a separate word

6.48. Make-3.80

The Make package contains a program for compiling large packages.

Approximate build time: 0.2 SBU
Required disk space: 8.8 MB

Make installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, and Sed

6.48.1. Installation of Make

Prepare Make for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **`make check`**.

Install the package:

```
make install
```

6.48.2. Contents of Make

Installed program: make

Short Descriptions

make Automatically determines which pieces of a large package need to be recompiled and then issues the relevant commands

6.49. Module-Init-Tools-3.0

The Module-Init-Tools package contains programs for handling kernel modules in Linux kernels greater than or equal to version 2.5.47.

Approximate build time: 0.1 SBU
Required disk space: 650 KB

Module-Init-Tools installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Glibc, Grep, M4, Make, and Sed

6.49.1. Installation of Module-Init-Tools

Prepare Module-Init-Tools for compilation:

```
./configure --prefix="" --enable-zlib
```

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

6.49.2. Contents of Module-Init-Tools

Installed programs: depmod, genksyms, insmod, insmod_ksymoops_clean, kallsyms (link to insmod), kernelversion, ksyms (link to insmod), lsmod (link to insmod), modinfo, modprobe (link to insmod), and rmmod (link to insmod)

Short Descriptions

depmod	Creates a dependency file based on the symbols it finds in the existing set of modules; this dependency file is used by **modprobe** to automatically load the required modules
genksyms	Generates symbol version information

insmod	Installs a loadable module in the running kernel
insmod_ksymoops_clean	Deletes saved ksyms and modules not accessed for two days
kallsyms	Extracts all kernel symbols for debugging
kernelversion	Reports the major version of the running kernel
ksyms	Displays exported kernel symbols
lsmod	Lists currently loaded modules
modinfo	Examines an object file associated with a kernel module and displays any information that it can glean
modprobe	Uses a dependency file, created by **depmod**, to automatically load relevant modules
rmmod	Unloads modules from the running kernel

6.50. Patch-2.5.4

The Patch package contains a program for modifying files.

Approximate build time: 0.1 SBU
Required disk space: 1.9 MB

Patch installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, and Sed

6.50.1. Installation of Patch

Prepare Patch for compilation. The preprocessor flag *-D_GNU_SOURCE* is only needed on the PowerPC platform. It can be left it out on other architectures:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

6.50.2. Contents of Patch

Installed program: patch

Short Descriptions

patch Modifies files according to a patch file. A patch file is normally a difference listing created with the **diff** program. By applying these differences to the original files, **patch** creates the patched versions.

6.51. Procps-3.2.3

The Procps package contains programs for monitoring processes.

Approximate build time: 0.1 SBU
Required disk space: 6.2 MB

Procps installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make, and Ncurses

6.51.1. Installation of Procps

Compile the package:

```
make
```

Install the package:

```
make install
```

6.51.2. Contents of Procps

Installed programs: free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w, and watch
Installed library: libproc.so

Short Descriptions

free	Reports the amount of free and used memory (both physical and swap memory) in the system
kill	Sends signals to processes
pgrep	Looks up processes based on their name and other attributes
pkill	Signals processes based on their name and other attributes
pmap	Reports the memory map of the given process
ps	Lists the current running processes
skill	Sends signals to processes matching the given criteria

snice	Changes the scheduling priority of processes matching the given criteria
sysctl	Modifies kernel parameters at run time
tload	Prints a graph of the current system load average
top	Displays the top CPU processes; it provides an ongoing look at processor activity in real time
uptime	Reports how long the system has been running, how many users are logged on, and the system load averages
vmstat	Reports virtual memory statistics, giving information about processes, memory, paging, block Input/Output (IO), traps, and CPU activity
w	Shows which users are currently logged on, where, and since when
watch	Runs a given command repeatedly, displaying the first screen-full of its output; this allows a user to watch the output change over time
`libproc`	Contains the functions used by most programs in this package

6.52. Psmisc-21.5

The Psmisc package contains programs for displaying information on processes.

Approximate build time: 0.1 SBU
Required disk space: 2.2 MB

Psmisc installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, and Sed

6.52.1. Installation of Psmisc

Prepare Psmisc for compilation:

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

The meaning of the configure option:

`--exec-prefix=""`
: This causes the binaries to be installed in `/bin` instead of `/usr/bin`. Because the Psmisc programs are often used in bootscripts, they should be available when the `/usr` file system is not mounted.

Compile the package:

```
make
```

Install the package:

```
make install
```

There is no reason for the **pstree** and **pstree.x11** programs to reside in `/bin`. Therefore, move them to `/usr/bin`. Also, there is no need for **pstree.x11** to exist as a separate program. Make it a symbolic link to **pstree** instead:

```
mv /bin/pstree* /usr/bin
ln -sf pstree /usr/bin/pstree.x11
```

By default, Psmisc's **pidof** program is not installed. This usually is not a problem because it is installed later in the Sysvinit package, which provides a better **pidof** program. If Sysvinit will not be used for a particular system, complete the installation of Psmisc by creating the following symlink:

```
ln -s killall /bin/pidof
```

6.52.2. Contents of Psmisc

Installed programs: fuser, killall, pstree, and pstree.x11 (link to pstree)

Short Descriptions

fuser	Reports the Process IDs (PIDs) of processes that use the given files or file systems
killall	Kills processes by name; it sends a signal to all processes running any of the given commands
pstree	Displays running processes as a tree
pstree.x11	Same as **pstree**, except that it waits for confirmation before exiting

6.53. Shadow-4.0.4.1

The Shadow package contains programs for handling passwords in a secure way.

Approximate build time: 0.4 SBU
Required disk space: 11 MB

Shadow installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Sed

6.53.1. Installation of Shadow

Prepare Shadow for compilation:

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

Work around a problem that prevents Shadow's internationalization from working:

```
echo '#define HAVE_SETLOCALE 1' >> config.h
```

Shadow incorrectly declares the malloc() function, causing compilation failure. Fix this:

```
sed -i '/extern char/d' libmisc/xmalloc.c
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Shadow uses two files to configure authentication settings for the system. Install these two config files:

```
cp etc/{limits,login.access} /etc
```

Instead of using the default *crypt* method, use the more secure *MD5* method of password encryption, which also allows passwords longer than 8 characters. It is also necessary to change the obsolete `/var/spool/mail` location for user mailboxes that Shadow uses by default to the `/var/mail` location used currently. Both of these can be accomplished by changing the relevant configuration file while copying it to its destination:

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

Move some misplaced symlinks/programs to their proper locations:

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

Move Shadow's dynamic libraries to a more appropriate location:

```
mv /usr/lib/lib{shadow,misc}.so.0* /lib
```

Because some packages expect to find the just-moved libraries in `/usr/lib`, create the following symlinks:

```
ln -sf ../../lib/libshadow.so.0 /usr/lib/libshadow.so
ln -sf ../../lib/libmisc.so.0 /usr/lib/libmisc.so
```

The *`-D`* option of the **useradd** program requires the `/etc/default` directory for it to work properly:

```
mkdir /etc/default
```

Coreutils has already installed a better **groups** program in `/usr/bin`. Remove the one installed by Shadow:

```
rm /bin/groups
```

6.53.2. Configuring Shadow

This package contains utilities to add, modify, and delete users and groups; set and change their passwords; and perform other administrative tasks. For a full explanation of what *password shadowing* means, see the `doc/HOWTO` file within the unpacked source tree. If using Shadow support, keep in mind that programs which need to verify passwords (display managers, FTP programs, pop3 daemons, etc.) must be shadow-compliant. That is, they need to be able to work with shadowed passwords.

To enable shadowed passwords, run the following command:

```
pwconv
```

To enable shadowed group passwords, run:

```
grpconv
```

Under normal circumstances, passwords will not have been created yet. However, if returning to this section later to enable shadowing, reset any current user passwords with the **passwd** command or any group passwords with the **gpasswd** command.

6.53.3. Setting the root password

Choose a password for user *root* and set it by running:

```
passwd root
```

6.53.4. Contents of Shadow

Installed programs: chage, chfn, chpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, groups, grpck, grpconv, grpunconv, lastlog, login, logoutd, mkpasswd, newgrp, newusers, passwd, pwck, pwconv, pwunconv, sg (link to newgrp), useradd, userdel, usermod, vigr (link to vipw), and vipw
Installed libraries: libshadow[.a,so]

Short Descriptions

chage	Used to change the maximum number of days between obligatory password changes
chfn	Used to change a user's full name and other info
chpasswd	Used to update the passwords of an entire series of user accounts
chsh	Used to change a user's default login shell
expiry	Checks and enforces the current password expiration policy
faillog	Is used to examine the log of login failures, to set a maximum number of failures before an account is blocked, or to reset the failure count
gpasswd	Is used to add and delete members and administrators to groups
groupadd	Creates a group with the given name
groupdel	Deletes the group with the given name
groupmod	Is used to modify the given group's name or GID
groups	Reports the groups of which the given users are members
grpck	Verifies the integrity of the group files `/etc/group` and `/etc/gshadow`
grpconv	Creates or updates the shadow group file from the normal group file
grpunconv	Updates `/etc/group` from `/etc/gshadow` and then deletes the latter
lastlog	Reports the most recent login of all users or of a given user
login	Is used by the system to let users sign on
logoutd	Is a daemon used to enforce restrictions on log-on time and ports

mkpasswd	Generates random passwords
newgrp	Is used to change the current GID during a login session
newusers	Is used to create or update an entire series of user accounts
passwd	Is used to change the password for a user or group account
pwck	Verifies the integrity of the password files `/etc/passwd` and `/etc/shadow`
pwconv	Creates or updates the shadow password file from the normal password file
pwunconv	Updates `/etc/passwd` from `/etc/shadow` and then deletes the latter
sg	Executes a given command while the user's GID is set to that of the given group
su	Runs a shell with substitute user and group IDs
useradd	Creates a new user with the given name, or updates the default new-user information
userdel	Deletes the given user account
usermod	Is used to modify the given user's login name, User Identification (UID), shell, initial group, home directory, etc.
vigr	Edits the `/etc/group` or `/etc/gshadow` files
vipw	Edits the `/etc/passwd` or `/etc/shadow` files
`libshadow`	Contains functions used by most programs in this package

6.54. Sysklogd-1.4.1

The Sysklogd package contains programs for logging system messages, such as those given by the kernel when unusual things happen.

Approximate build time: 0.1 SBU
Required disk space: 0.5 MB

Sysklogd installation depends on: Binutils, Coreutils, GCC, Glibc, and Make

6.54.1. Installation of Sysklogd

Sysklogd has issues with the Linux 2.6 kernel series. Fix these issues by applying the following patch:

```
patch -Np1 -i ../sysklogd-1.4.1-kernel_headers-1.patch
```

There is also a race condition in the signal handling logic, and this sometimes confuses the **sysklogd** initscript. Fix this bug by applying another patch:

```
patch -Np1 -i ../sysklogd-1.4.1-signal-1.patch
```

Compile the package:

```
make
```

Install the package:

```
make install
```

6.54.2. Configuring Sysklogd

Create a new file `/etc/syslog.conf` by running the following:

```
cat > /etc/syslog.conf << "EOF"
# Begin /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 *

# End /etc/syslog.conf
EOF
```

6.54.3. Contents of Sysklogd

Installed programs: klogd and syslogd

Short Descriptions

klogd	A system daemon for intercepting and logging kernel messages
syslogd	Logs the messages that system programs offer for logging

6.55. Sysvinit-2.85

The Sysvinit package contains programs for controlling the startup, running, and shutdown of the system.

Approximate build time: 0.1 SBU
Required disk space: 0.9 MB

Sysvinit installation depends on: Binutils, Coreutils, GCC, Glibc, and Make

6.55.1. Installation of Sysvinit

Sysvinit-2.85 contains a “buffer overflow” bug. Under some conditions, it modifies the values of environment variables. Fix this with:

```
patch -Np1 -i ../sysvinit-2.85-proclen-1.patch
```

When run-levels are changed (for example, when halting the system), **init** sends termination signals to those processes that **init** itself started and that should not be running in the new run-level. While doing this, **init** outputs messages like “Sending processes the TERM signal” which seem to imply that it is sending these signals to all currently running processes. To avoid this misinterpretation, modify the source so that these messages read like “Sending processes started by init the TERM signal” instead:

```
sed -i 's@Sending processes@& started by init@g' \
    src/init.c
```

Compile the package:

```
make -C src
```

Install the package:

```
make -C src install
```

6.55.2. Configuring Sysvinit

Create a new file `/etc/inittab` by running the following:

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty -I '\033(K' tty1 9600
2:2345:respawn:/sbin/agetty -I '\033(K' tty2 9600
3:2345:respawn:/sbin/agetty -I '\033(K' tty3 9600
4:2345:respawn:/sbin/agetty -I '\033(K' tty4 9600
5:2345:respawn:/sbin/agetty -I '\033(K' tty5 9600
6:2345:respawn:/sbin/agetty -I '\033(K' tty6 9600

# End /etc/inittab
EOF
```

The *-I '\033(K'* option tells **agetty** to send this escape sequence to the terminal before doing anything else. This escape sequence switches the console character set to a user-defined one, which can be modified by running the **setfont** program. The **console** initscript from the LFS-Bootscripts package calls the **setfont** program during system startup. Sending this escape sequence is necessary for people who use non-ISO 8859-1 screen fonts, but it does not effect native English speakers.

6.55.3. Contents of Sysvinit

Installed programs: halt, init, killall5, last, lastb (link to last), mesg, pidof (link to killall5), poweroff (link to halt), reboot (link to halt), runlevel, shutdown, sulogin, telinit (link to init), utmpdump, and wall

Short Descriptions

halt	Normally invokes **shutdown** with the `-h` option, except when already in run-level 0, then it tells the kernel to halt the system; it notes in the file `/var/log/wtmp` that the system is being brought down
init	The first process to be started when the kernel has initialized the hardware which takes over the boot process and starts all the proceses it is instructed to
killall5	Sends a signal to all processes, except the processes in its own session so it will not kill the shell running the script that called it
last	Shows which users last logged in (and out), searching back through the `/var/log/wtmp` file; it also shows system boots, shutdowns, and run-level changes
lastb	Shows the failed login attempts, as logged in `/var/log/btmp`
mesg	Controls whether other users can send messages to the current user's terminal
pidof	Reports the PIDs of the given programs
poweroff	Tells the kernel to halt the system and switch off the computer (see **halt**)
reboot	Tells the kernel to reboot the system (see **halt**)
runlevel	Reports the previous and the current run-level, as noted in the last run-level record in `/var/run/utmp`
shutdown	Brings the system down in a secure way, signaling all processes and notifying all logged-in users
sulogin	Allows *root* to log in; it is normally invoked by **init** when the system goes into single user mode

telinit	Tells **init** which run-level to change to
utmpdump	Displays the content of the given login file in a more user-friendly format
wall	Writes a message to all logged-in users

6.56. Tar-1.14

The Tar package contains an archiving program.

Approximate build time: 0.2 SBU
Required disk space: 10 MB

Tar installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, and Sed

6.56.1. Installation of Tar

Prepare Tar for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**.

Install the package:

```
make install
```

6.56.2. Contents of Tar

Installed programs: rmt and tar

Short Descriptions

rmt Remotely manipulates a magnetic tape drive through an interprocess communication connection

tar Creates and extracts files from archives, also known as tarballs

6.57. Udev-030

The Udev package contains programs for dynamic creation of device nodes.

Approximate build time: 0.2 SBU
Required disk space: 5.2 MB

Udev installation depends on: Coreutils and Make

6.57.1. Installation of Udev

Compile the package:

```
make udevdir=/dev
```

udevdir=/dev
This tells **udev** in which directory devices nodes are to be created.

This package does not come with a test suite.

Install the package:

```
make udevdir=/dev install
```

Udev's configuration is far from ideal by default, so install the configuration files here:

```
cp ../udev-config-2.permissions \
    /etc/udev/permissions.d/25-lfs.permissions
cp ../udev-config-1.rules /etc/udev/rules.d/25-lfs.rules
```

6.57.2. Contents of Udev

Installed programs: udev, udevd, udevsend, udevstart, udevinfo, and udevtest
Installed directory: /etc/udev

Short Descriptions

udev	Creates device nodes in `/dev` or renames network interfaces (not in LFS) in response to hotplug events

udevd A daemon that reorders hotplug events before submitting them to **udev**, thus avoiding various race conditions

udevsend Delivers hotplug events to **udevd**

udevstart Creates device nodes in `/dev` that correspond to drivers compiled directly into the kernel; it performs that task by simulating hotplug events presumably dropped by the kernel before invocation of this program (e.g., because the root filesystem has not been mounted) and submitting such synthetic hotplug events to **udev**

udevinfo Allows users to query the **udev** database for information on any device currently present on the system; it also provides a way to query any device in the `sysfs` tree to help create udev rules

udevtest Simulates a **udev** run for the given device, and prints out the name of the node the real **udev** would have created or (not in LFS) the name of the renamed network interface

`/etc/udev` Contains **udev** configuation files, device permissions, and rules for device naming

6.58. Util-linux-2.12b

The Util-linux package contains miscellaneous utility programs. Among them are utilities for handling file systems, consoles, partitions, and messages.

Approximate build time: 0.2 SBU
Required disk space: 16 MB

Util-linux installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, and Zlib

6.58.1. FHS compliance notes

The FHS recommends using the `/var/lib/hwclock` directory instead of the usual `/etc` directory as the location for the `adjtime` file. To make the **hwclock** program FHS-compliant, run the following:

```
sed -i 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
    hwclock/hwclock.c
mkdir -p /var/lib/hwclock
```

6.58.2. Installation of Util-linux

GCC-3.4.1 does not properly compile **sfdisk** if the default optimization level is used. The following patch corrects the problem:

```
patch -Np1 -i ../util-linux-2.12b-sfdisk-2.patch
```

Prepare Util-linux for compilation:

```
./configure
```

Compile the package:

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

The meaning of the make parameters:

HAVE_KILL=yes
: This prevents the **kill** program (already installed by Procps) from being built and installed again.

HAVE_SLN=yes
: This prevents the **sln** program (a statically linked version of **ln** already installed by Glibc) from being built and installed again.

This package does not come with a test suite.

Install the package:

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

6.58.3. Contents of Util-linux

Installed programs: agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, ipcrm, ipcs, isosize, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, pg, pivot_root, ramsize (link to rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (link to rdev), script, setfdprm, setsid, setterm, sfdisk, swapdev, swapoff (link to swapon), swapon, tunelp, ul, umount, vidmode (link to rdev), whereis, and write

Short Descriptions

agetty	Opens a tty port, prompts for a login name, and then invokes the **login** program
arch	Reports the machine's architecture
blockdev	Allows users to call block device ioctls from the command line
cal	Displays a simple calendar
cfdisk	Manipulates the partition table of the given device
chkdupexe	Finds duplicate executables
col	Filters out reverse line feeds

colcrt	Filters **nroff** output for terminals that lack some capabilities, such as overstriking and half-lines
colrm	Filters out the given columns
column	Formats a given file into multiple columns
ctrlaltdel	Sets the function of the Ctrl+Alt+Del key combination to a hard or a soft reset
cytune	Tunes the parameters of the serial line drivers for Cyclades cards
ddate	Gives the Discordian date or converts the given Gregorian date to a Discordian one
dmesg	Dumps the kernel boot messages
elvtune	Tunes the performance and interactivity of a block device
fdformat	Low-level formats a floppy disk
fdisk	Manipulates the partition table of the given device
fsck.cramfs	Performs a consistency check on the Cramfs file system on the given device
fsck.minix	Performs a consistency check on the Minix file system on the given device
getopt	Parses options in the given command line
hexdump	Dumps the given file in hexadecimal or in another given format
hwclock	Reads or sets the system's hardware clock, also called the Real-Time Clock (RTC)) or Basic Input-Output System (BIOS) clock
ipcrm	Removes the given Inter-Process Communication (IPC) resource
ipcs	Provides IPC status information
isosize	Reports the size of an iso9660 file system
line	Copies a single line
logger	Enters the given message into the system log
look	Displays lines that begin with the given string
losetup	Sets up and controls loop devices

mcookie	Generates magic cookies (128-bit random hexadecimal numbers) for **xauth**
mkfs	Builds a file system on a device (usually a hard disk partition)
mkfs.bfs	Creates an Santa Cruz Operations (SCO) bfs file system
mkfs.cramfs	Creates a cramfs file system
mkfs.minix	Creates a Minix file system
mkswap	Initializes the given device or file to be used as a swap area
more	A filter for paging through text one screen at a time
mount	Attaches the file system on the given device to a specified directory in the file-system tree
namei	Shows the symbolic links in the given pathnames
pg	Displays a text file one screen full at a time
pivot_root	Makes the given file system the new root file system of the current process
ramsize	Sets the size of the RAM disk in a bootable image
raw	Used to bind a Linux raw character device to a block device
rdev	Queries and sets the root device, among other things, in a bootable image
readprofile	Reads kernel profiling information
rename	Renames the given files, replacing a given string with another
renice	Alters the priority of running processes
rev	Reverses the lines of a given file
rootflags	Sets the rootflags in a bootable image
script	Makes a typescript of a terminal session
setfdprm	Sets user-provided floppy disk parameters
setsid	Runs the given program in a new session
setterm	Sets terminal attributes

sfdisk	A disk partition table manipulator
swapdev	Sets the swap device in a bootable image
swapoff	Disables devices and files for paging and swapping
swapon	Enables devices and files for paging and swapping
tunelp	Tunes the parameters of the line printer
ul	A filter for translating underscores into escape sequences indicating underlining for the terminal in use
umount	Disconnects a file system from the system's file tree
vidmode	Sets the video mode in a bootable image
whereis	Reports the location of binary, the source, and the manual page for the given command
write	Sends a message to the given user *if* that user has not disabled receipt of such messages

6.59. About Debugging Symbols

Most programs and libraries are, by default, compiled with debugging symbols included (with **gcc**'s *-g* option). This means that when debugging a program or library that was compiled with debugging information included, the debugger can provide not only memory addresses, but also the names of the routines and variables.

However, the inclusion of these debugging symbols enlarges a program or library significantly. The following is an example of the amount of space these symbols occupy:

- a bash binary with debugging symbols: 1200 KB
- a bash binary without debugging symbols: 480 KB
- Glibc and GCC files (`/lib` and `/usr/lib`) with debugging symbols: 87 MB
- Glibc and GCC files without debugging symbols: 16 MB

Sizes may vary depending on which compiler and C library were used, but when comparing programs with and without debugging symbols, the difference will usually be a factor between two and five.

Because most users will never use a debugger on their system software, a lot of disk space can be regained by removing these symbols. The next section shows how to strip all debugging symbols from the programs and libraries. Additional information on system optimization can be found at *http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt*.

6.60. Stripping Again

If the intended user is not a programmer and does not plan to do any debugging on the system software, the system size can be decreased by about 200 MB by removing the debugging symbols from binaries and libraries. This causes no inconvenience other than not being able to debug the software fully anymore.

Most people who use the command mentioned below do not experience any difficulties. However, it is easy to make a typo and render the new system unusable, so before running the **strip** command, it is a good idea to make a backup of the current situation.

Before performing the stripping, take special care to ensure that none of the binaries that are about to be stripped are running. If unsure whether the user entered chroot with the command given in Section 6.3, “Entering the Chroot Environment,” first exit from chroot:

```
logout
```

Then reenter it with:

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

Now the binaries and libraries can be safely stripped:

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

A large number of files will be reported as having their file format not recognized. These warnings can be safely ignored. These warnings indicate that those files are scripts instead of binaries.

If disk space is very tight, the `--strip-all` option can be used on the binaries in `/{,usr/}{bin,sbin}` to gain several more megabytes. Do not use this option on libraries—they will be destroyed.

6.61. Cleaning Up

From now on, when reentering the chroot environment after exiting, use the following modified chroot command:

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

The reason for this is that, since the programs in `/tools` are no longer needed, the directory can be deleted to regain space. Before actually deleting the directory, exit from chroot and reenter it with the above command. Also, before removing `/tools`, tar it up and store it in a safe place in case another LFS system will be built.

> **Note**
>
> Removing `/tools` will also remove the temporary copies of Tcl, Expect, and DejaGNU which were used for running the toolchain tests. To use these programs later on, they will need to be recompiled and re-installed. The installation instructions are the same as in Chapter 5, apart from changing the prefix from `/tools` to `/usr`. The BLFS book discusses a slightly different approach to installing Tcl (see *http://www.linuxfromscratch.org/blfs/*).

The packages and patches stored in `/sources` can also be moved to a more usual location, such as `/usr/src/packages`. The entire directory can also be deleted if its contents have been burned to a CD.

Chapter 7. Setting Up System Bootscripts

7.1. Introduction

This chapter details how to install the bootscripts and set them up properly. Most of these scripts will work without modification, but a few require additional configuration files because they deal with hardware-dependent information.

System-V style init scripts are employed in this book because they are widely used. For additional options, a hint detailing the BSD style init setup is available at *http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt*. Searching the LFS mailing lists for “depinit” will also offer additional choices.

If using an alternate style of init scripts, skip this chapter and move on to Chapter 8.

7.2. LFS-Bootscripts-2.2.2

The LFS-Bootscripts package contains a set of bootscripts.

Approximate build time: 0.1 SBU
Required disk space: 0.3 MB

LFS-Bootscripts installation depends on: Bash and Coreutils

7.2.1. Installation of LFS-Bootscripts

Install the package:

```
make install
```

7.2.2. Contents of LFS-bootscripts

Installed scripts: checkfs, cleanfs, console, functions, halt, ifdown, ifup, localnet, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysklogd, template, and udev

Short Descriptions

checkfs	Checks the file systems before they are mounted (with the exception of journal and network based file systems)
cleanfs	Removes files that should not be preserved between reboots, such as those in `/var/run/` and `/var/lock/`; it re-creates `/var/run/utmp` and removes the possibly present `/etc/nologin`, `/fastboot`, and `/forcefsck` files
console	Loads the keymap table specified as proper for the keyboard layout; it also sets the screen font
functions	Contains functions shared among different scripts, such as error and status checking
halt	Halts the system
ifdown	Assists the network script with network devices

ifup	Assists the network script with network devices
localnet	Sets up the system's hostname and local loopback device
mountfs	Mounts all file systems, except ones that are marked *noauto* or are network based
mountkernfs	Is used to mount kernel-provided file systems, such as `proc`
network	Sets up network interfaces, such as network cards, and sets up the default gateway (where applicable)
rc	The master run-level control script; it is responsible for running all other scripts one-by-one, in a sequence determined by the name of the symbolic links being processed
reboot	Reboots the system
sendsignals	Makes sure every process is terminated before the system reboots or halts
setclock	Resets the kernel clock to local time in case the hardware clock is not set to UTC time
static	Provides the functionality needed to assign a static Internet Protocol (IP) address to a network interface
swap	Enables and disables swap files and partitions
sysklogd	Starts and stops the system and kernel log daemons
template	A template to create custom bootscripts for other daemons
udev	Sets up udev and create the devices nodes in `/dev`

7.3. How Do These Bootscripts Work?

Linux uses a special booting facility named SysVinit that is based on a concept of *run-levels*. It can be quite different from one system to another, so it cannot be assumed that because things worked in <insert distro name>, they should work the same in LFS too. LFS has its own way of doing things, but it respects generally accepted standards.

SysVinit (which will be referred to as “init” from now on) works using a run-levels scheme. There are seven (from 0 to 6) run-levels (actually, there are more run-levels, but they are for special cases and are generally not used. The init man page describes those details), and each one of those corresponds to the actions the computer is supposed to perform when it starts up. The default run-level is 3. Here are the descriptions of the different run-levels as they are implemented:

```
0: halt the computer
1: single-user mode
2: multi-user mode without networking
3: multi-user mode with networking
4: reserved for customization, otherwise does the same as 3
5: same as 4, it is usually used for GUI login (like X's xdm or KDE's kdm)
6: reboot the computer
```

The command used to change run-levels is **init *`[runlevel]`***, where *`[runlevel]`* is the target run-level. For example, to reboot the computer, a user would issue the **init 6** command. The **reboot** command is an alias for it, as is the **halt** command an alias for **init 0**.

There are a number of directories under `/etc/rc.d` that look like `rc?.d` (where ? is the number of the run-level) and `rcsysinit.d`, all containing a number of symbolic links. Some begin with a *K*, the others begin with an *S*, and all of them have two numbers following the initial letter. The K means to stop (kill) a service and the S means to start a service. The numbers determine the order in which the scripts are run, from 00 to 99—the lower the number the earlier it gets executed. When init switches to another run-level, the appropriate services get killed and others get started.

The real scripts are in `/etc/rc.d/init.d`. They do the actual work, and the symlinks all point to them. Killing links and starting links point to the same script in `/etc/rc.d/init.d`. This is because the scripts can be called with different parameters like *`start`*, *`stop`*, *`restart`*, *`reload`*, and *`status`*. When a K link is encountered, the appropriate script is run with the *`stop`* argument. When an S link is encountered, the appropriate script is run with the *`start`* argument.

There is one exception to this explanation. Links that start with an *S* in the `rc0.d` and `rc6.d` directories will not cause anything to be started. They will be called with the parameter *`stop`* to stop something. The logic behind this is that when a user is going to reboot or halt the system, nothing needs to be started. The system only needs to be stopped.

These are descriptions of what the arguments make the scripts do:

`start`
: The service is started.

`stop`
: The service is stopped.

`restart`
: The service is stopped and then started again.

`reload`
: The configuration of the service is updated. This is used after the configuration file of a service was modified, when the service does not need to be restarted.

`status`
: Tells if the service is running and with which PIDs.

Feel free to modify the way the boot process works (after all, it is your own LFS system). The files given here are an example of how it can be done.

7.4. Device and Module Handling on an LFS System

In Chapter 6, we installed the Udev package. Before we go into the details regarding how this works, a brief history of previous methods of handling devices is in order.

Linux systems in general traditionally use a static device creation method, whereby a great many device nodes are created under `/dev` (sometimes literally thousands of nodes), regardless of whether the corresponding hardware devices actually exist. This is typically done via a **MAKEDEV** script, which contains a number of calls to the **mknod** program with the relevant major and minor device numbers for every possible device that might exist in the world. Using the udev method, only those devices which are detected by the kernel get device nodes created for them. Because these device nodes will be created each time the system boots, they will be stored on a `ramfs` (a file system that resides entirely in memory and does not take up any disk space). Device nodes do not require much disk space, so the memory that is used in negligable.

7.4.1. History

In February 2000, a new filesystem called `devfs` was merged into the 2.3.46 kernel and was made available during the 2.4 series of stable kernels. Although it was present in the kernel source itself, this method of creating devices dynamically never received overwhelming support from the core kernel developers.

The main problem with the approach adopted by `devfs` was the way it handled device detection, creation, and naming. The latter issue, that of device node naming, was perhaps the most critical. It is generally accepted that if device names are allowed to be configurable, then the device naming policy should be up to a system administrator, not imposed on them by any particular developer(s). The `devfs` file system also suffers from race conditions that are inherent in its design and cannot be fixed without a substantial revision to the kernel. It has also been marked as deprecated due to a lack of recent maintenance.

With the development of the unstable 2.5 kernel tree, later released as the 2.6 series of stable kernels, a new virtual filesystem called `sysfs` came to be. The job of `sysfs` is to export a view of the system's structure to userspace processes. With this userspace visible representation, the possibility of seeing a userspace replacement for `devfs` became much more realistic.

7.4.2. Udev Implementation

The `sysfs` filesystem was mentioned briefly above. One may wonder how `sysfs` knows about the devices present on a system and what device numbers should be used. Drivers that have been compiled into the kernel directly register their objects with `sysfs` as they are detected by the kernel. For drivers compiled as modules, this will happen when the module is loaded. Once the `sysfs` filesystem is mounted (on `/sys`), the data which the built-in drivers registered with `sysfs` are available to userspace processes and to **udev** for device node creation.

The **S10udev** initscript takes care of creating these device nodes when Linux is booted. This script starts with registering **/sbin/udev** as a hotplug event handler. Hotplug events (discussed below) should not be generated during this stage, but **udev** is registered just in case they do occur. The **udevstart** program then walks through the `/sys` filesystem and creates devices under `/dev` that match the descriptions. For example, `/sys/class/tty/vcs/dev` contains the string “7:0” This string is used by **udevstart** to create `/dev/vcs` with major number *7* and minor *0*. The permissions of each and every device that **udevstart** creates are set using files from the `/etc/udev.d/permissions.d/` directory. These are numbered in a similar fashion to the LFS bootscripts. If **udev** cannot find a permissions file for the device it is creating, it will default permissions to *600* and ownership to *root:root*. The names of the nodes created under the `/dev` directory are configured according to the rules specified in the files within the `/etc/udev/rules.d/` directory.

Once the above stage is complete, all devices that were already present and have compiled-in drivers will be available for use. What about those devices that have modular drivers?

Earlier, we mentioned the concept of a “hotplug event handler.” When a new device connection is detected by the kernel, the kernel will generate a hotplug event and look at the file `/proc/sys/kernel/hotplug` to find out the userspace program that handles the device's connection. The **udev** initscript registered **udev** as this handler. When these hotplug events are generated, the kernel will tell **udev** to check the `/sys` filesystem for the information pertaining to this new device and create the `/dev` entry for it.

This brings us to one problem that exists with **udev**, and likewise with `devfs` before it. It is commonly referred to as the “chicken and egg” problem. Most Linux distrubtions handle loading modules via entries in `/etc/modules.conf`. Access to a device node causes the appropriate kernel module to load. With **udev**, this method will not work because the device node does not exist until the module is loaded. To solve this, the **S05modules** bootscript was added to the lfs-bootscripts package, along with the `/etc/sysconfig/modules` file. By adding module names to the `modules` file, these modules will be loaded when the computer is starting up. This allows **udev** to detect the devices and create the appropriate device nodes.

Note that on slower machines or for drivers that create a lot of device nodes, the process of creating devices may take a few seconds to complete. This means that some device nodes may not be immediately accessible.

7.4.3. Handling Hotpluggable/Dynamic Devices

When you plug in a device, such a Universal Serial Bus (USB) MP3 player, the kernel recognizes that the device is now connected and generates a hotplug event. If the driver is already loaded (either because it was compiled into the kernel or because it was loaded via the **S05modules** bootscript), **udev** will be called upon to create the relevant device node(s) according to the `sysfs` data available in `/sys`. If the driver for the just plugged in device is available as a module but currently unloaded, then attaching the device to the system will only cause the kernel's bus driver to generate a hotplug event that notifies userspace of the new device connection and it not being attached to a driver. In effect, nothing happens and the device itself is not usable yet.

If building a system that has a lot of drivers compiled as modules rather than directly built into the kernel, using the **S05modules** may not be practical. The Hotplug package (see *http://linux-hotplug.sourceforge.net/*) can be beneficial in these cases. When the Hotplug package is installed, it will respond to the aforementioned kernel's bus driver hotplug events. The Hotplug package will load the appropriate module and make this device available by creating the device node(s) for it.

7.4.4. Problems with Creating Devices

There are a few known problems when it comes to automatically creating devices nodes:

1) A kernel driver may not export its data to `sysfs`.

This is most common with third party drivers from outside the kernel tree. These drivers will not end up having their device nodes created. Use the `/etc/sysconfig/createfiles` configuration file to manually create the devices. Consult the `devices.txt` file inside the kernel documentation or the documentation for that driver to find the proper major/minor numbers.

2) A non-hardware device is required. This is most common with the Advanced Linux Sound Architecture (ALSA) project's Open Sound System (OSS) compatibility module. These types of devices can be handled in one of two ways:

- Adding the module names to `/etc/sysconfig/modules`
- Using an "install" line in `/etc/modprobe.conf`. This tells the **modprobe** command "when loading this module, also load this other module, at the same time." For example:

  ```
  install snd-pcm modprobe -i snd-pcm ; modprobe \
      snd-pcm-oss ; true
  ```

  This will cause the system to load both the *snd-pcm* and *snd-pcm-oss* modules when any request is made to load the driver *snd-pcm*.

7.4.5. Useful Reading

Additional helpful documentation is available at the following sites:

- A Userspace Implementation of devfs *http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf*
- udev FAQ *http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ*
- The Linux Kernel Driver Model *http://public.planetmirror.com/pub/lca/2003/proceedings/papers/Patrick_Mochel/Patrick_Mochel.pdf*

7.5. Configuring the setclock Script

The **setclock** script reads the time from the hardware clock, also known as BIOS or the Complementary Metal Oxide Semiconductor (CMOS) clock. If the hardware clock is set to UTC, this script will convert the hardware clock's time to the local time using the `/etc/localtime` file (which tells the **hwclock** program which timezone the user is in). There is no way to detect whether or not the hardware clock is set to UTC time, so this needs to be manually configured.

If you cannot remember whether or not the hardware clock is set to UTC time, find out by running the **`hwclock --localtime --show`** command. This will tell what the current time is according to the hardware clock. If this time matches whatever your watch says, then the hardware clock is set to local time. If the output from **hwclock** is not local time, chances are it is set to UTC time. Verify this by adding or subtracting the proper amount of hours for the timezone to this **hwclock** time. For example, if you live in the MST timezone, which is also known as GMT -0700, add seven hours to the local time. Then, account for Daylight Savings Time, which requires subtracting an hour (or only add six in the first place) during the summer months.

Change the value of the `UTC` variable below to a value of *`0`* (zero) if the hardware clock is *not* set to UTC time.

Create a new file `/etc/sysconfig/clock` by running the following:

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock

UTC=1

# End /etc/sysconfig/clock
EOF
```

A good hint explaining how to deal with time on LFS is available at *http://www.linuxfromscratch.org/hints/downloads/files/time.txt*. It explains issues such as time zones, UTC, and the `TZ` environment variable.

7.6. Configuring the Linux Console

This section discusses how to configure the **console** initscript that sets up the keyboard map and the console font. If non-ASCII characters (British pound and Euro character are examples of non-ASCII characters) will not be used and the keyboard is a U.S. one, skip this section. Without the configuration file, the console initscript will do nothing.

The **console** script uses the `/etc/sysconfig/console` as a configuration file. Decide which keymap and screen font will be used. The language-specific HOWTO can help with this. A pre-made `/etc/sysconfig/console` file with known settings for several countries was installed with the LFS-Bootscripts package, so the relevant section can be uncommented if the country is supported. If still in doubt, look in the `/usr/share/kbd` directory for valid keymaps and screen fonts. Read the loadkeys and setfont manual pages and determine the correct arguments for these programs. Once decided, create the configuration file with the following command:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="[arguments for loadkeys]"
FONT="[arguments for setfont]"
EOF
```

For example, for Spanish users who also want to use the Euro character (accessible by pressing AltGr+E), the following settings are correct:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="es euro2"
FONT="lat9-16 -u iso01"
EOF
```

Note

The `FONT` line above is correct only for the ISO 8859-15 character set. If using ISO 8859-1 and, therefore, a pound sign instead of Euro, the correct `FONT` line would be:

```
FONT="lat1-16"
```

If the `KEYMAP` or `FONT` variable is not set, the **console** initscript will not run the corresponding program.

In some keymaps, the Backspace and Delete keys send characters different from ones in the default keymap built into the kernel. This confuses some applications. For example, Emacs displays its help (instead of erasing the character before the cursor) when Backspace is pressed. To check if the keymap in use is effected (this works only for i386 keymaps):

```
zgrep '\W14\W' [/path/to/your/keymap]
```

If the keycode 14 is Backspace instead of Delete, create the following keymap snippet to fix this issue:

```
mkdir -p /etc/kbd && cat > /etc/kbd/bs-sends-del <<"EOF"
                  keycode  14 = Delete Delete Delete Delete
              alt keycode  14 = Meta_Delete
        altgr alt keycode  14 = Meta_Delete
                  keycode 111 = Remove
    altgr control keycode 111 = Boot
      control alt keycode 111 = Boot
altgr control alt keycode 111 = Boot
EOF
```

Tell the **console** script to load this snippet after the main keymap:

```
cat >>/etc/sysconfig/console <<"EOF"
KEYMAP_CORRECTION="/etc/kbd/bs-sends-del"
EOF
```

To compile the keymap directly into the kernel instead of setting it every time from the **console** bootscript, follow the instructions given in Section 8.3, “Linux-2.6.8.1.” Doing this ensures that the keyboard will always work as expected, even when booting into maintenance mode (by passing *`init=/bin/sh`* to the kernel), because the **console** bootscript will not be run in that situation. Additionally, the kernel will not set the screen font automatically. This should not pose many problems because ASCII characters will be handled correctly, and it is unlikely that a user would need to rely on non-ASCII characters while in maintenance mode.

Since the kernel will set up the keymap, it is possible to omit the `KEYMAP` variable from the `/etc/sysconfig/console` configuration file. It can also be left in place, if desired, without consequence. Keeping it could be beneficial if running several different kernels where it is difficult to ensure that the keymap is compiled into every one of them.

7.7. Creating the /etc/inputrc File

The `/etc/inputrc` file deals with mapping the keyboard for specific situations. This file is the start-up file used by Readline, the input-related library used by Bash and most other shells.

For more information, see the bash info page, section *Readline Init File*. The readline info page is also a good source of information.

Global values are set in `/etc/inputrc`. Personal user values are set in `~/.inputrc`. The `~/.inputrc` file will override the global settings file. A later page sets up Bash to use `/etc/inputrc` if there is no `.inputrc` for a user when `/etc/profile` is read (usually at login). To make the system use both, or to negate global keyboard handling, it is a good idea to place a default `.inputrc` into the `/etc/skel` directory for use with new users.

Below is a base `/etc/inputrc`, along with comments to explain what the various options do. Note that comments cannot be on the same line as commands.

To create the `.inputrc` in `/etc/skel` using the command below, change the command's output to `/etc/skel/.inputrc` and be sure to check/set permissions afterward. Copy that file to `/etc/inputrc` and the home directory of any user already existing on the system, including *root*, that needs a private version of the file. Be certain to use the *-p* parameter of **cp** to maintain permissions and be sure to change owner and group appropriately.

```
cat > /etc/inputrc << "EOF"
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>

# Make sure we don't output everything on the 1 line
set horizontal-scroll-mode Off

# Enable 8bit input
set meta-flag On
set input-meta On

# Turns off 8th bit stripping
set convert-meta Off

# Keep the 8th bit for display
set output-meta On
```

```
# none, visible or audible
set bell-style none

# All of the following map the escape sequence of the
# value contained inside the 1st argument to the
# readline specific functions

"\eOd": backward-word
"\eOc": forward-word

# for linux console
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert

# for xterm
"\eOH": beginning-of-line
"\eOF": end-of-line

# for Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line

# End /etc/inputrc
EOF
```

7.8. The Bash Shell Startup Files

The shell program **/bin/bash** (hereafter referred to as "the shell") uses a collection of startup files to help create an environment to run in. Each file has a specific use and may effect login and interactive environments differently. The files in the `/etc` directory provide global settings. If an equivalent file exists in the home directory, it may override the global settings.

An interactive login shell is started after a successful login, using **/bin/login**, by reading the `/etc/passwd` file. An interactive non-login shell is started at the command-line (e.g., `[prompt]$`**/bin/bash**). A non-interactive shell is usually present when a shell script is running. It is non-interactive because it is processing a script and not waiting for user input between commands.

For more information, see **info bash** - Nodes: Bash Startup Files and Interactive Shells.

The files `/etc/profile` and `~/.bash_profile` are read when the shell is invoked as an interactive login shell.

A base `/etc/profile` below sets some environment variables necessary for native language support. Setting them properly results in:

- The output of programs translated into the native language
- Correct classification of characters into letters, digits and other classes. This is necessary for Bash to properly accept non-ASCII characters in command lines in non-English locales
- The correct alphabetical sorting order for the country
- Appropriate default paper size
- Correct formatting of monetary, time, and date values

This script also sets the `INPUTRC` environment variable that makes Bash and Readline use the `/etc/inputrc` file created earlier.

Replace *[ll]* below with the two-letter code for the desired language (e.g., “en”) and *[CC]* with the two-letter code for the appropriate country (e.g., “GB”). It may also be necessary to specify (and this is actually the preferred form) the character encoding (e.g. “iso8859-1”) after a dot (so that the result is “en_GB.iso8859-1”). Issue the following command for more information:

```
man 3 setlocale
```

The list of all locales supported by Glibc can be obtained by running the following command:

```
locale -a
```

Once the proper locale settings have been determined, create the `/etc/profile` file:

```
cat > /etc/profile << "EOF"
# Begin /etc/profile

export LC_ALL=[ll]_[CC]
export LANG=[ll]_[CC]
export INPUTRC=/etc/inputrc

# End /etc/profile
EOF
```

Note

The “C” (default) and “en_US” (the recommended one for United States English users) locales are different.

Setting the keyboard layout, screen font, and locale-related environment variables are the only internationalization steps needed to support locales that use ordinary single-byte encodings and left-to-right writing direction. More complex cases (including UTF-8 based locales) require additional steps and additional patches because many applications tend to not work properly under such conditions. These steps and patches are not included in the LFS book and such locales are not supported by LFS in any way.

7.9. Configuring the sysklogd Script

The `sysklogd` script invokes the **syslogd** program with the *`-m 0`* option. This option turns off the periodic timestamp mark that **syslogd** writes to the log files every 20 minutes by default. To turn on this periodic timestamp mark, edit the `sysklogd` script and make the changes accordingly. See **`man syslogd`** for more information.

7.10. Configuring the localnet Script

Part of the localnet script is setting up the system's hostname. This needs to be configured in the `/etc/sysconfig/network`.

Create the `/etc/sysconfig/network` file and enter a hostname by running:

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

[lfs] needs to be replaced with the name the computer is to be called. Do not enter the Fully Qualified Domain Name (FQDN) here. That information will be put in the `/etc/hosts` file later.

7.11. Creating the /etc/hosts File

If a network card is to be configured, decide on the IP-address, FQDN, and possible aliases for use in the `/etc/hosts` file. The syntax is:

```
<IP address> myhost.example.org aliases
```

Unless the computer is to be visible to the Internet (e.g., there is a registered domain and a valid block of assigned IP addresses—most users do not have this), make sure that the IP address is in the private network IP address range. Valid ranges are:

```
        Class Networks
        A     10.0.0.0
        B     172.16.0.0 through 172.31.0.0
        C     192.168.0.0 through 192.168.255.0
```

A valid IP address could be 192.168.1.1. A valid FQDN for this IP could be www.linuxfromscratch.org (not recommended because this is a valid registered domain address and could cause domain name server issues).

Even if not using a network card, an FQDN is still required. This is necessary for certain programs to operate correctly.

Create the `/etc/hosts` file by running:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
[192.168.1.1] [<HOSTNAME>.example.org] [HOSTNAME]

# End /etc/hosts (network card version)
EOF
```

The *`[192.168.1.1]`* and *`[<HOSTNAME>.example.org]`* values need to be changed for specific users or requirements (if assigned an IP address by a network/system administrator and the machine will be connected to an existing network).

If a network card is not going to be configured, create the `/etc/hosts` file by running:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)

127.0.0.1 [<HOSTNAME>.example.org] [HOSTNAME] localhost

# End /etc/hosts (no network card version)
EOF
```

7.12. Configuring the network Script

This section only applies if a network card is to be configured.

If a network card will not be used, there is likely no need to create any configuration files relating to network cards. If that is the case, remove the `network` symlinks from all run-level directories (`/etc/rc.d/rc*.d`).

7.12.1. Creating Network Interface Configuration Files

Which interfaces are brought up and down by the network script depends on the files in the `/etc/sysconfig/network-devices` directory. This directory should contain files in the form of `ifconfig.xyz`, where “xyz” is a network interface name (such as eth0 or eth0:1).

If the `/etc/sysconfig/network-devices` directory is to be renamed or moved, make sure to edit the `/etc/sysconfig/rc` file and update the “network_devices” option by providing it with the new path.

New files are created in this directory. The following command creates a sample `ipv4` file for the *eth0* device:

```
cd /etc/sysconfig/network-devices &&
mkdir ifconfig.eth0 &&
cat > ifconfig.eth0/ipv4 << "EOF"
ONBOOT=yes
SERVICE=ipv4-static
IP=192.168.1.1
GATEWAY=192.168.1.2
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

The values of these variables must be changed in every file to match the proper setup. If the `ONBOOT` variable is set to “yes” the network script will bring up the Network Interface Card (NIC) during booting of the system. If set to anything but “yes” the NIC will be ignored by the network script and not brought up.

The `SERVICE` variable defines the method of obtaining the IP address. The LFS bootscripts have a modular IP assignment format, and creating additional files in the `/etc/sysconfig/network-devices/services` directory allows other IP assignment methods. This is commonly used for Dynamic Host Configuration Protocol (DHCP), which is addressed in the BLFS book.

The `GATEWAY` variable should contain the default gateway IP address, if one is present. If not, then comment out the variable entirely.

The `PREFIX` variable needs to contain the number of bits used in the subnet. Each octet in an IP address is 8 bits. If the subnet's netmask is 255.255.255.0, then it is using the first three octets (24 bits) to specify the network number. If the netmask is 255.255.255.240, it would be using the first 28 bits. Prefixes longer than 24 bits are commonly used by DSL- and cable-based Internet Service Providers (ISPs). In this example (PREFIX=24), the netmask is 255.255.255.0. Adjust according to the specific subnet.

7.12.2. Creating the /etc/resolv.conf File

If the system is going to be connected to the Internet, it will need some means of Domain Name Service (DNS) name resolution to resolve Internet domain names to IP addresses, and vice versa. This is best achieved by placing the IP address of the DNS server, available from the ISP or network administrator, into `/etc/resolv.conf`. Create the file by running the following:

```
cat > /etc/resolv.conf << "EOF"
# Begin /etc/resolv.conf

domain {[Your Domain Name]}
nameserver [IP address of your primary nameserver]
nameserver [IP address of your secondary nameserver]

# End /etc/resolv.conf
EOF
```

Replace *`[IP address of the nameserver]`* with the IP address of the DNS most appropriate for the setup. There will often be more than one entry (requirements demand secondary servers for fallback capability). If you only need or want one DNS server, remove the second *nameserver* line from the file. The IP address may also be a router on the local network.

Chapter 8. Making the LFS System Bootable

8.1. Introduction

It is time to make the LFS system bootable. This chapter discusses creating an `fstab` file, building a kernel for the new LFS system, and installing the Grub boot loader so that the LFS system can be selected for booting at startup.

8.2. Creating the /etc/fstab File

The `/etc/fstab` file is used by some programs to determine where file systems are to be mounted by default, which must be checked, and in which order. Create a new file systems table like this:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab

# file system  mount-point  type   options         dump  fsck
#                                                        order

/dev/[xxx]     /            [fff]  defaults        1     1
/dev/[yyy]     swap         swap   pri=1           0     0
proc           /proc        proc   defaults        0     0
sysfs          /sys         sysfs  defaults        0     0
devpts         /dev/pts     devpts gid=4,mode=620  0     0
shm            /dev/shm     tmpfs  defaults        0     0
# End /etc/fstab
EOF
```

Replace *[xxx]*, *[yyy]*, and *[fff]* with the values appropriate for the system, for example, `hda2`, `hda5`, and `ext2`. For details on the six fields in this file, see **man 5 fstab**.

When using a journalling file system, the *1 1* at the end of the line should be replaced with *0 0* because such a partition does not need to be dumped or checked.

The `/dev/shm` mount point for `tmpfs` is included to allow enabling POSIX-shared memory. The kernel must have the required support built into it for this to work (more about this is in the next section). Please note that very little software currently uses POSIX-shared memory. Therefore, consider the `/dev/shm` mount point optional. For more information, see `Documentation/filesystems/tmpfs.txt` in the kernel source tree.

There are other lines which may be added to the `/etc/fstab` file. One example is a line for USB devices:

```
usbfs        /proc/bus/usb usbfs   devgid=14,devmode=0660 0 0
```

This option will only work if “Support for Host-side USB” and “USB device filesystem” are compiled into the kernel (not as a module).

8.3. Linux-2.6.8.1

The Linux package contains the kernel and the header files.

Approximate build time: 4.20 SBU
Required disk space: 181 MB

Linux installation depends on: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, and Sed

8.3.1. Installation of the kernel

Building the kernel involves a few steps—configuration, compilation, and installation. Read the `README` file in the kernel source tree for alternate methods to the way this book configures the kernel.

Prepare for compilation by running the following command:

```
make mrproper
```

This ensures that the kernel tree is absolutely clean. The kernel team recommends that this command be issued prior to each kernel compilation. Do not rely on the source tree being clean after un-tarring.

Also, ensure that the kernel does not attempt to pass hotplugging events to userspace until userspace specifies that it is ready:

```
sed -i 's@/sbin/hotplug@/bin/true@' kernel/kmod.c
```

If, in Section 7.6, “Configuring the Linux Console,” it was decided to compile the keymap into the kernel, issue the command below:

```
loadkeys -m /usr/share/kbd/keymaps/[path to  keymap] > \
    drivers/char/defkeymap.c
```

For example, if using a Dutch keyboard, use `/usr/share/kbd/keymaps/i386/qwerty/nl.map.gz`.

Configure the kernel via a menu-driven interface:

```
make menuconfig
```

Alternatively, **make oldconfig** may be more appropriate in some situations. See the `README` file for more information.

> **Note**
>
> When configuring the kernel, be sure to enable the “Support for hot-pluggable devices” option under the “General Setup” menu. This enables hotplug events that are used by **udev** to populate the `/dev` directory with device nodes.

If desired, skip kernel configuration by copying the kernel config file, `.config`, from the host system (assuming it is available) to the unpacked `linux-2.6.8.1` directory. However, we do not recommend this option. It is often better to explore all the configuration menus and create the kernel configuration from scratch.

For POSIX-shared memory support, ensure that the kernel config option “Virtual memory file system support” is enabled. It resides within the “File systems” menu and is normally enabled by default.

LFS bootscripts make the assumption that either both “Support for Host-side USB” and “USB device filesystem” have been compiled directly into the kernel, or that neither is compiled at all. Bootscripts will not work properly if it is a module (usbcore.ko).

> **Note**
>
> NPTL requires the kernel to be compiled with GCC 3.x, in this case 3.4.1. Compiling with 2.95.x is known to cause failures in the glibc test suite, so it is not recommended to compile the kernel with gcc 2.95.x.

Compile the kernel image and modules:

```
make
```

If using kernel modules, an `/etc/modprobe.conf` file may be needed. Information pertaining to modules and kernel configuration is located in the kernel documentation in the `linux-2.6.8.1/Documentation` directory. The *modprobe.conf* man page may also be of interest.

Be very careful when reading other documentation because it usually applies to 2.4.x kernels only. As far as we know, kernel configuration issues specific to Hotplug and Udev are not documented. The problem is that Udev will create a device node only if Hotplug or a user-written script inserts the corresponding module into the kernel, and not all modules are detectable by Hotplug. Note that statements like the one below in the `/etc/modprobe.conf` file do not work with Udev:

```
alias char-major-XXX some-module
```

Because of the complications with Hotplug, Udev, and modules, we strongly recommend starting with a completely non-modular kernel configuration, especially if this is the first time using Udev.

Install the modules, if the kernel configuration uses them:

```
make modules_install
```

If there are many modules and very little space, consider stripping and compressing the modules. For most users, such compression is not worth the time, but if the system is pressed for space, see *http://www.linux-mips.org/archives/linux-mips/2002-04/msg00031.html*.

After kernel compilation is complete, additional steps are required to complete the installation. Some files need to be copied to the `/boot` directory.

The path to the kernel image may vary depending on the platform being used. Issue the following command to install the kernel:

```
cp arch/i386/boot/bzImage /boot/lfskernel-2.6.8.1
```

`System.map` is a symbol file for the kernel. It maps the function entry points of every function in the kernel API, as well as the addresses of the kernel data structures for the running kernel. Issue the following command to install the map file:

```
cp System.map /boot/System.map-2.6.8.1
```

The kernel configuration file `.config` produced by the **make menuconfig** step above contains all the configuration selections for the kernel that was just compiled. It is a good idea to keep this file for future reference:

```
cp .config /boot/config-2.6.8.1
```

It is important to note that the files in the kernel source directory are not owned by *root*. Whenever a package is unpacked as user *root* (like we did inside chroot), the files have the user and group IDs of whatever they were on the packager's computer. This is usually not a problem for any other package to be installed because the source tree is removed after the installation. However, the Linux source tree is often retained for a long time. Because of this, there is a chance that whatever user ID the packager used will be assigned to somebody on the machine. That person would then have write access to the kernel source.

If the kernel source tree is going to retained, run **chown -R 0:0** on the `linux-2.6.8.1` directory to ensure all files are owned by user *root*.

8.3.2. Contents of Linux

Installed files: kernel, kernel headers, and System.map

Short Descriptions

`kernel`	The engine of the Linux system. When turning on the computer, the kernel is the first part of the operating system that gets loaded. It detects and initializes all components of the computer's hardware, then makes these components available as a tree of files to the software and turns a single CPU into a multitasking machine capable of running scores of programs seemingly at the same time.
`kernel headers`	Defines the interface to the services that the kernel provides. The headers in the system's `include` directory should *always* be the ones against which Glibc was compiled and therefore, should *not* be replaced when upgrading the kernel.
`System.map`	A list of addresses and symbols; it maps the entry points and addresses of all the functions and data structures in the kernel

8.4. Making the LFS System Bootable

Your shiny new LFS system is almost complete. One of the last things to do is to ensure that the system can be properly booted. The instructions below apply only to computers of IA-32 architecture, meaning mainstream PCs. Information on “boot loading” for other architectures should be available in the usual resource-specific locations for those architectures.

Boot loading can be a complex area, so a few cautionary words are in order. Be familiar with the current boot loader and any other operating systems present on the hard drive(s) that need to be bootable. Make sure that an emergency boot disk is ready to “rescue” the computer if the computer becomes unusable (un-bootable).

Earlier, we compiled and installed the Grub boot loader software in preparation for this step. The procedure involves writing some special Grub files to specific locations on the hard drive. We highly recommend creating a Grub boot floppy diskette as a backup. Insert a blank floppy diskette and run the following commands:

```
dd if=/boot/grub/stage1 of=/dev/fd0 bs=512 count=1
dd if=/boot/grub/stage2 of=/dev/fd0 bs=512 seek=1
```

Remove the diskette and store it somewhere safe. Now, run the **grub** shell:

```
grub
```

Grub uses its own naming structure for drives and partitions in the form of *(hdn,m)*, where *n* is the hard drive number and *m* is the partition number, both starting from zero. For example, partition `hda1` is *(hd0,0)* to Grub and `hdb3` is *(hd1,2)*. In contrast to Linux, Grub does not consider CD-ROM drives to be hard drives. For example, if using a CD on `hdb` and a second hard drive on `hdc`, that second hard drive would still be *(hd1)*.

Using the above information, determine the appropriate designator for the root partition (or boot partition, if a separate one is used). For the following example, it is assumed that the root (or separate boot) partition is `hda4`.

Tell Grub where to search for its `stage{1,2}` files. The Tab key can be used everywhere to make Grub show the alternatives:

```
root (hd0,3)
```

> **Warning**
>
> The following command will overwrite the current boot loader. Do not run the command if this is not desired, for example, if using a third party boot manager to manage the Master Boot Record (MBR). In this scenario, it would make more sense to install Grub into the “boot sector” of the LFS partition. In this case, this next command would become **`setup (hd0,3)`**.

Tell Grub to install itself into the MBR of `hda`:

```
setup (hd0)
```

If all went well, Grub will have reported finding its files in `/boot/grub`. That's all there is to it. Quit the **grub** shell:

```
quit
```

Create a “menu list” file defining Grub's boot menu:

```
cat > /boot/grub/menu.lst << "EOF"
# Begin /boot/grub/menu.lst

# By default boot the first menu entry.
default 0

# Allow 30 seconds before booting the default.
timeout 30

# Use prettier colors.
color green/black light-green/black

# The first entry is for LFS.
title LFS 6.0
root (hd0,3)
kernel /boot/lfskernel-2.6.8.1 root=/dev/hda4
EOF
```

Add an entry for the host distribution if desired. It might look like this:

```
cat >> /boot/grub/menu.lst << "EOF"
title Red Hat
root (hd0,2)
kernel /boot/kernel-2.4.20 root=/dev/hda3
initrd /boot/initrd-2.4.20
EOF
```

If dual-booting Windows, the following entry will allow booting it:

```
cat >> /boot/grub/menu.lst << "EOF"
title Windows
rootnoverify (hd0,0)
chainloader +1
EOF
```

If **info grub** does not provide all necessary material, additional information regarding Grub is located on its website at: *http://www.gnu.org/software/grub/*.

Chapter 9. The End

9.1. The End

Well done! The new LFS system is installed! We wish you much success with your shiny new custom-built Linux system.

It may be a good idea to create an `/etc/lfs-release` file. By having this file, it is very easy for you (and for us if you need to ask for help at some point) to find out which LFS version is installed on the system. Create this file by running:

```
echo 6.0 > /etc/lfs-release
```

9.2. Get Counted

Now that you have finished the book, do you want to be counted as an LFS user? Head over to *http://www.linuxfromscratch.org/cgi-bin/lfscounter.cgi* and register as an LFS user by entering your name and the first LFS version you have used.

Let's reboot into LFS now.

9.3. Rebooting the System

Now that all of the software has been installed, it is time to reboot the computer. First exit from the chroot environment:

```
logout
```

Then unmount the virtual files systems:

```
umount $LFS/dev/pts
umount $LFS/dev/shm
umount $LFS/dev
umount $LFS/proc
umount $LFS/sys
```

Unmount the LFS file system itself:

```
umount $LFS
```

If multiple partitions were created, unmount the other partitions before unmounting the main one, like this:

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

Now, reboot the system with:

```
shutdown -r now
```

Assuming the Grub boot loader was set up as outlined earlier, the menu is set to boot *LFS 6.0* automatically.

When the reboot is complete, the LFS system is ready for use and software can be added.

9.4. What Now?

Thank you for reading this LFS book. We hope that you have found this book helpful and have learned more about the system creation process.

Now that the LFS system is installed, you may be wondering “What next?” To answer that question, we have compiled a list of resources for you.

- Beyond Linux From Scratch

  The Beyond Linux From Scratch book covers installation procedures for a wide range of software beyond the scope of the LFS Book. The BLFS project is located at *http://www.linuxfromscratch.org/blfs/*.

- LFS Hints

  The LFS Hints are a collection of educational documents submitted by volunteers in the LFS community. The hints are available at *http://www.linuxfromscratch.org/hints/list.html*.

- Mailing lists

  There are several LFS mailing lists you may subscribe to if you are in need of help, want to stay current with the latest developments, want to contribute to the project, and more. See Chapter 1 - Mailing Lists for more information.

- The Linux Documentation Project

  The goal of The Linux Documentation Project (TLDP) is to collaborate on all of the issues of Linux documentation. The TLDP features a large collection of HOWTOs, guides, and man pages. It is located at *http://www.tldp.org/*.

Part IV. Appendices

Appendix A. Acronyms and Terms

ABI	Application Binary Interface
ALFS	Automated Linux From Scratch
ALSA	Advanced Linux Sound Architecture
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BIOS	Basic Input/Output System
BLFS	Beyond Linux From Scratch
BSD	Berkeley Software Distribution
chroot	change root
CMOS	Complementary Metal Oxide Semiconductor
COS	Class Of Service
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CVS	Concurrent Versions System
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Service
EGA	Enhanced Graphics Adapter
ELF	Executable and Linkable Format
EOF	End of File
EQN	equation
EVMS	Enterprise Volume Management System
ext2	second extended file system
FAQ	Frequently Asked Questions

FHS	Filesystem Hierarchy Standard
FIFO	First-In, First Out
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
GB	Gibabytes
GCC	GNU Compiler Collection
GID	Group Identifier
GMT	Greenwich Mean Time
GPG	GNU Privacy Guard
HTML	Hypertext Markup Language
IDE	Integrated Drive Electronics
IEEE	Institute of Electrical and Electronic Engineers
IO	Input/Output
IP	Internet Protocol
IPC	Inter-Process Communication
IRC	Internet Relay Chat
ISO	International Organization for Standardization
ISP	Internet Service Provider
KB	Kilobytes
LED	Light Emitting Diode
LFS	Linux From Scratch
LSB	Linux Standards Base
MB	Megabytes
MBR	Master Boot Record
MD5	Message Digest 5

NIC	Network Interface Card
NLS	Native Language Support
NNTP	Network News Transport Protocol
NPTL	Native POSIX Threading Library
OSS	Open Sound System
PCH	Pre-Compiled Headers
PCRE	Perl Compatible Regular Expression
PID	Process Identifier
PLFS	Pure Linux From Scratch
PTY	pseudo terminal
QA	Quality Assurance
QOS	Quality Of Service
RAM	Random Access Memory
RPC	Remote Procedure Call
RTC	Real Time Clock
SBU	Static Binutils Unit
SCO	The Santa Cruz Operation
SGR	Select Graphic Rendition
SHA1	Secure-Hash Algorithm 1
SMP	Symmetric Multi-Processor
TLDP	The Linux Documentation Project
TFTP	Trivial File Transfer Protocol
TLS	Thread-Local Storage
UID	User Identifier
umask	user file-creation mask

USB	Universal Serial Bus
UTC	Coordinated Universal Time
UUID	Universally Unique Identifier
VC	Virtual Console
VGA	Video Graphics Array
VT	Virtual Terminal

Appendix B. Acknowledgments

We would like to thank the following people and organizations for their contributions to the Linux From Scratch Project.

Project Team Members

- *Gerard Beekmans* <gerard@linuxfromscratch.org> – Linux From Scratch initiator, LFS Project organizer
- *Christine Barczak* <theladyskye@linuxfromscratch.org> – LFS Book Editor
- *Matthew Burgess* <matthew@linuxfromscratch.org> – LFS Project Co-Leader, LFS general package maintainer, LFS Technical Writer.
- *Craig Colton* <meerkats@bellsouth.net> – LFS, Automated Linux From Scratch (ALFS), BLFS and hints project logo creator
- *Nathan Coulson* <nathan@linuxfromscratch.org> – LFS bootscripts maintainer
- *Jeroen Coumans* <jeroen@linuxfromscratch.org> – Website developer, FAQ maintainer
- *Bruce Dubbs* <bdubbs@linuxfromscratch.org> – LFS Quality Assurance (QA) Team leader, BLFS Book Editor
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – LFS XML/XSL maintainer
- *Jim Gifford* <jim@linuxfromscratch.org> – LFS Technical Writer, Patches maintainer
- *Nicholas Leippe* <nicholas@linuxfromscratch.org> – Wiki maintainer
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Website backend scripts maintainer
- *Scot Mc Pherson* <scot@linuxfromscratch.org> – LFS NNTP gateway maintainer.
- *Ryan Oliver* <ryan@linuxfromscratch.org> – Testing Team leader, Toolchain maintainer, co-creator of Pure LFS (PLFS)
- *Alexander Patrakov* <semzx@newmail.ru> – Former LFS Technical Writer

- *James Robertson* <jwrober@linuxfromscratch.org> – Bugzilla maintainer, Wiki developer, LFS Tecnical Writer
- *Tushar Teredesai* <tushar@linuxfromscratch.org> – BLFS Book Editor, hints and patches projects maintainer
- *Jeremy Utley* <jeremy@linuxfromscratch.org> – LFS Technical Writer, Bugzilla maintainer, LFS bootscripts maintainer, LFS Server co-administrator
- *Zack Winkles* <zwinkles@gmail.com> – Former LFS Technical Writer
- Countless other people on the various LFS and BLFS mailing lists who helped make this book possible by giving their suggestions, testing the book, and submitting bug reports, instructions, and their experiences with installing various packages.

Translators

- *Manuel Canales Esparcia* <macana@lfs-es.org> – Spanish LFS translation project
- *Johan Lenglet* <johan@linuxfromscratch.org> – French LFS translation project
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Portuguese LFS translation project
- *Thomas Reitelbach* <tr@erdfunkstelle.de> – German LFS translation project

Mirror Maintainers

North American Mirrors

- *Scott Kveton* <scott@osuosl.org> – lfs.oregonstate.edu mirror
- *Mikhail Pastukhov* <miha@xuy.biz> – lfs.130th.net mirror
- *William Astle* <lost@l-w.net> – ca.linuxfromscratch.org mirror
- *Jeremy Polen* <jpolen@rackspace.com> – us2.linuxfromscratch.org mirror
- *Tim Jackson* <tim@idge.net> – linuxfromscratch.idge.net mirror
- *Jeremy Utley* <jeremy@linux-phreak.net> – lfs.linux-phreak.net mirror

South American Mirrors

- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.org mirror
- *Andres Meggiotto* <sysop@mesi.com.ar> – lfs.mesi.com.ar mirror
- *Eduardo B. Fonseca* <ebf@aedsolucoes.com.br> – br.linuxfromscratch.org mirror

European Mirrors

- *Barna Koczka* <barna@siker.hu> – hu.linuxfromscratch.org mirror
- *UK Mirror Service* – linuxfromscratch.mirror.ac.uk mirror
- *Martin Voss* <Martin.Voss@ada.de> – lfs.linux-matrix.net mirror
- *Guido Passet* <guido@primerelay.net> – nl.linuxfromscratch.org mirror
- *Bastiaan Jacques* <baafie@planet.nl> – lfs.pagefault.net mirror
- *Roel Neefs* <lfs-mirror@linuxfromscratch.rave.org> – linuxfromscratch.rave.org mirror
- *Justin Knierim* <justin@jrknierim.de> – www.lfs-matrix.de mirror
- *Stephan Brendel* <stevie@stevie20.de> – lfs.netservice-neuss.de mirror
- *Antonin Sprinzl* <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org mirror
- *Fredrik Danerklint* <fredan-lfs@fredan.org> – se.linuxfromscratch.org mirror
- *Parisian sysadmins* <archive@doc.cs.univ-paris8.fr> – www2.fr.linuxfromscratch.org mirror
- *Alexander Velin* <velin@zadnik.org> – bg.linuxfromscratch.org mirror
- *Dirk Webster* <dirk@securewebservices.co.uk> – lfs.securewebservices.co.uk mirror
- *Thomas Skyt* <thomas@sofagang.dk> – dk.linuxfromscratch.org mirror
- *Simon Nicoll* <sime@dot-sime.com> – uk.linuxfromscratch.org mirror

Asian Mirrors

- *Pui Yong* <pyng@spam.averse.net> – sg.linuxfromscratch.org mirror
- *Stuart Harris* <stuart@althalus.me.uk> – lfs.mirror.intermedia.com.sg mirror

Australian Mirrors

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

A very special thank you to our donators

- *Dean Benson* <dean@vipersoft.co.uk> for several monetary contributions
- *Hagen Herrschaft* <hrx@hrxnet.de> for donating a 2.2 GHz P4 system, now running under the name of Lorien
- *VA Software* who, on behalf of *Linux.com*, donated a VA Linux 420 (former StartX SP2) workstation
- Mark Stone for donating Belgarath, the linuxfromscratch.org server

Index

Packages

Programs

Libraries

Scripts

Others