Linux From Scratch

Version 5.0

Gerard Beekmans

This book describes the process of creating a Linux system from scratch, using nothing but the sources of the required software.

Dedication

This book is dedicated to my loving and supportive wife *Beverly Beekmans*.

Table of Contents

Preface

Foreword

Having used a number of different Linux distributions, I was never fully satisfied with any of them. I didn't like the arrangement of the bootscripts. I didn't like the way certain programs were configured by default. Much more of that sort of thing bothered me. Finally I realized that if I wanted full satisfaction from my Linux system I would have to build my own system from scratch, using only the source code. I resolved not to use pre-compiled packages of any kind, nor CD-ROM or boot disk that would install some basic utilities. I would use my current Linux system to develop my own.

This wild idea seemed very difficult at the time and often seemed an impossible task. After sorting out all kinds of problems, such as dependencies and compile-time errors, a custom-built Linux system was created that was fully operational. I called this system a Linux From Scratch system, or LFS for short.

I hope you will have a great time working on your own LFS!

—
Gerard Beekmans
gerard@linuxfromscratch.org

Audience

Who would want to read this book

There are many reasons why somebody would want to read this book. The principal reason being 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 to you what makes Linux tick, 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 you have more control of your system without relying on someone else's Linux implementation. With LFS, you are in the driver's seat and dictate every aspect of your 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, you are usually forced to install several programs which you are likely never to use. They're just sitting there wasting precious disk space (or worse, CPU cycles). It isn't difficult to build an LFS system less than 100 MB. Does that still sound like a lot? A few of us have been working on creating a very small embedded LFS system. We successfully built a system that was just enough to run the Apache web server with approximately 8MB of disk space used. Further stripping could bring that down to 5 MB or less. Try that with a regular distribution.

We could compare distributed Linux to a hamburger you buy at a fast-food restaurant — you have no idea what you are eating. LFS, on the other hand, doesn't give you a hamburger, but the recipe to make a hamburger. This allows you to review it, to omit unwanted ingredients, and to add your own ingredients which enhance the flavor of your burger. When you are satisfied with the recipe, you go on to preparing it. You make it just the way you like it: broil it, bake it, deep-fry it, barbecue it, or eat it tar-tar (raw).

Another analogy that we can use is that of comparing LFS with a finished house. LFS will give you the skeletal plan of a house, but it's up to you to build it. You have the freedom to adjust your plans as you go.

One last 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 you feel are needed. You don't have 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 actually fixes the problem (adequately).

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

Who would not want to read this book

There are probably some who, for whatever reason, would feel that they do not want to read this book. If you do not wish to build your own Linux system from scratch, then you probably don't want to read this book. Our goal is to help you build a complete and usable foundation-level system. If you only want to know what happens while your computer boots, then we recommend the "From Power Up To Bash Prompt" HOWTO. The HOWTO builds a bare system which is similar to that of this book, but it focuses strictly on creating a system capable of booting to a BASH prompt.

While you decide which to read, consider your objective. If you wish to build a Linux system while learning a bit along the way, then this book is probably your best choice. If your objective is strictly educational and you do not have any plans for your finished system, then the "From Power Up To Bash Prompt" HOWTO is probably a better choice.

The "From Power Up To Bash Prompt" HOWTO is located at `http://axiom.anu.edu.au/~okeefe/p2b/` or on The Linux Documentation Project's website at `http://www.tldp.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html`.

Prerequisites

This book assumes that its reader has a good deal of knowledge about using and installing Linux software. Before you begin building your LFS system, you should read the following HOWTOs:

- Software-Building-HOWTO

  This is a comprehensive guide to building and installing "generic" UNIX software distributions under Linux. This HOWTO is available at `http://www.tldp.org/HOWTO/Software-Building-HOWTO.html`.

- The Linux Users' Guide

  This guide covers the usage of assorted Linux software and is available at `http://espc22.murdoch.edu.au/~stewart/guide/guide.html`.

- The Essential Pre-Reading Hint

  This is an LFS Hint written specifically for new users of Linux. It is mostly a list of links to excellent sources of information on a wide range of topics. Any person attempting to install LFS, should at least have an understanding of many of the topics in this hint. It is available at `http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt`

Structure

This book is divided into the following four parts:

Part I - Introduction

Part I explains a few important things on how to proceed with the installation, and gives meta information about the book (version, changelog, acknowledgments, associated mailing lists, and so on).

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.

Part III - Building the LFS system

Part III guides you 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 basic Linux system is the foundation upon which you can build other software, to extend your system in the way you like.

Part IV - Appendices

Part IV consists of two appendices. The first is an alphabetical list of all the packages that are installed — for each package giving its official download location, its contents, and its installation dependencies. The second appendix lists all the programs and libraries installed by these packages in alphabetical order, so you can easily find out to which package a certain program or library belongs.

(Much of the first appendix is integrated into Parts II and III. This enlarges the book somewhat, but we believe it makes for easier reading. Now you don't have to keep referencing the appendix while doing the installation. This going back and forth would be a real chore, especially if you're reading a plain text version of the book.)

Part I - Introduction

Chapter 1 Introduction

How things are going to be done

You are going to build your LFS system 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, because you will need programs like a compiler, linker and shell to build the new system. Normally all the required tools are available if you selected "development" as one of the options when you installed your distribution.

In Chapter 3 you will first create a new Linux native partition and file system, the place where your new LFS system will be compiled and installed. Then in Chapter 4 you download all the packages and patches required to build an LFS system, and store them on the new file system.

Chapter 5 then discusses 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.

The first thing to be done in Chapter 5 is build a first pass of the toolchain, made up of Binutils and GCC. The programs from these packages will be linked statically in order for them to be usable independently of the host system. The second thing to do is build Glibc, the C library. Glibc will be compiled by the toolchain programs we just built in the first pass. The third thing to do is build a second pass of the toolchain. This time the toolchain will be dynamically linked against the newly built Glibc. The remaining Chapter 5 packages are all built using this second pass toolchain and dynamically linked against the new host-independent Glibc. When this is done, the LFS installation process will no longer depend on the host distribution, with the exception of the running kernel.

You may be thinking that "this seems like a lot of work, just to get away from my host distribution". Well, a full technical explanation is provided at the start of Chapter 5, including some notes on the differences between statically and dynamically linked programs.

In Chapter 6 your real LFS system will be 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 reason that you don't actually reboot, but instead chroot, is that creating a bootable system requires additional work which isn't necessary just yet. But the major advantage is that chrooting allows you to continue using the host while LFS is being built. While waiting for package compilation to complete, you can simply switch to a different VC (Virtual Console) or X desktop and continue using the computer as you normally would.

To finish the installation, the bootscripts are set up in Chapter 7, the kernel and bootloader are set up in Chapter 8, and Chapter 9 contains some pointers to help you after you finish the book. Then, finally, you're ready to reboot your computer into your new LFS system.

This is the process in a nutshell. Detailed information on the steps you will take are discussed in the chapters and package descriptions as you progress through them. If something isn't completely clear now, don't worry, everything will fall into place soon.

Please read Chapter 2 carefully as it explains a few important things you should be aware of before you begin to work through Chapter 5 and beyond.

Conventions used in this book

To make things easy to follow, there are a number of conventions used throughout the book. Following are some examples:

```
./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) is showing screen output, probably as the result of commands issued, and 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, and to give examples of what to type.

`http://www.linuxfromscratch.org/`

This form of text is used for hyperlinks, both within the book and to external pages such as HOWTOs, download locations and websites.

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

This type of section is used mainly 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 EOF is encountered. Therefore, this whole section is generally typed as seen.

Book version

This is version 5.0 of the Linux From Scratch book, dated November 5th, 2003. If this book is more than two months old, a newer and better version is probably already available. To find out, check one of the mirrors listed on `http://www.linuxfrom-scratch .org/`.

Changelog

5.0 - November 5th, 2003

- Upgraded to:
    - automake-1.7.6
    - bash-2.05b
    - binutils-2.14
    - e2fsprogs-1.34
    - file-4.04
    - findutils-4.1.20
    - gawk-3.1.3
    - gcc-3.3.1
    - gettext-0.12.1
    - glibc-2.3.2
    - glibc-2.3.2-sscanf-1.patch
    - grep-2.5.1
    - groff-1.19
    - gzip-1.3.5
    - less-381
    - lfs-bootscripts-1.12
    - libtool-1.5
    - linux-2.4.22
    - man-1.5m2
    - man-1.5m2-80cols.patch
    - man-1.5m2-manpath.patch
    - man-1.5m2-pager.patch
    - man-pages-1.60

  - modutils-2.4.25
  - procps-3.1.11
  - procps-3.1.11.patch
  - psmisc-21.3
  - sed-4.0.7
  - sysvinit-2.85
  - tar-1.13.25
  - texinfo-4.6
  - util-linux-2.12
  - vim-6.2
- Added:
  - bash-2.05b-2.patch
  - bison-1.875-attribute.patch
  - coreutils-5.0
  - coreutils-5.0-uname.patch
  - coreutils-5.0-hostname-2.patch
  - dejagnu-1.4.3
  - expect-5.39.0
  - expect-5.39.0.patch
  - gawk-3.1.3.patch
  - gcc-2.95.3
  - gcc-2.95.3-2.patch
  - gcc-2.95.3-no-fixinc.patch
  - gcc-2.95.3-returntype-fix.patch
  - gcc-3.3.1-no_fixincludes-2.patch
  - gcc-3.3.1-specs-2.patch
  - gcc-3.3.1-suppress-libiberty.patch
  - grub-0.93
  - grub-0.93-gcc33-1.patch
  - inetutils-1.4.2
  - lfs-utils-0.3
  - ncurses-5.3-etip-2.patch

    - ncurses-5.3-vsscanf.patch
    - perl-5.8.0-libc-3.patch
    - shadow-4.0.3-newgroup-fix.patch
    - tcl-8.4.4
    - zlib-1.1.4-vsnprintf.patch
- Removed:
    - bin86-0.16.3
    - fileutils-4.1
    - fileutils-4.1.patch
    - findutils-4.1-segfault.patch
    - findutils-4.1.patch
    - glibc-2.3.1-libnss.patch
    - glibc-2.3.1-root-perl.patch
    - gzip-1.2.4b.patch
    - lilo-22.2
    - netkit-base-0.17
    - sh-utils-2.0
    - sh-utils-2.0.patch
    - sh-utils-2.0-hostname.patch
    - tar-1.13.patch
    - textutils-2.1
    - vim-6.1.patch
- November 2nd, 2003 [alex]: Appendix A - Commented out all the "last checked against" lines.
- October 28th, 2003 [greg]: Strengthened the seds in "Locking in Glibc" and "Re-adjusting the toolchain" sections.
- October 26th, 2003 [greg]: Chapter 6 - Glibc: Added command to create /etc/ld.so.conf to match Chapter 5 Glibc. Closes bug 700.
- October 24th, 2003 [alex]: Appendix A - Changed the dependencies to the concise format, based on Tushar's post .
- October 23rd, 2003 [gerard] Chapter 9 - The End: Changed the `/etc/lfs` filename to `/etc/lfs-release` to be more consistent with other distributions out there.

- October 23rd, 2003 [alex]: Changed most of the "Chapter" references to proper "xref" cross references .
- October 22nd, 2003 [alex]: Chapter 6 - Gawk and Shadow: Adjusted the text. And added some markup elsewhere.
- October 22nd, 2003 [alex]: Chapter 6 - Entering the chroot environment: Dropped the `set +h` command, as it is pointless there: it's redone several sections later.
- October 15th, 2003 [greg]: Chapter 9: Reworked final strip command. Relocated paragraphs about directory removal from Chapter 6.
- October 14th, 2003 [greg]: Chapter 8 - Making the LFS system bootable: Expanded Grub details and added a warning.
- October 14th, 2003 [alex]: Appendix A - Updated the contents of Perl and Procps.
- October 14th, 2003 [alex]: Chapter 4 and 5 - Added a suggestion to use $LFS/sources as the working and storage place.
- October 13th, 2003 [greg]: Chapter 9 - Rebooting the system: Reworked umount commands.
- October 11th, 2003 [alex]: Adapted the required disk space values and SBUs, as posted by Bruce Dubbs.
- October 11th, 2003 [alex]: Chapter 5 - Toolchain technical notes: Added and changed some markup.
- October 9th, 2003 [gerard]: Upgraded to lfs-bootscripts-1.12.
- October 9th, 2003 [greg]: Performed internal markup reworking to fix an extraneous whitespace problem in "tidy generated" web site pages. Essentially replace all occurrences of <para><screen> with <screen> (and the matching closing tags).
- October 9th, 2003 [alex]: Chapter 6 - Basic Networking: Moved one half to the Lfs-Utils section, the other half to Perl.
- October 8th, 2003 [alex]: Chapter 8 - Making bootable: Adapted the style of the screens, and reworded some paragraphs.
- October 8th, 2003 [alex]: Removed a series of unused entities.
- October 7th, 2003 [jeremy]: Added notes to the linking tests in chapter 5 and 6 stating that blank output is a bad thing.
- October 7th, 2003 [alex]: Changed the patch entities to contain the full filename instead of just the version number.
- October 7th, 2003 [jeremy]: Chapter 1 - Added a note regarding #LFS-support on IRC.

- October 7th, 2003 [greg]: Preface: Add note about the Essential Pre-Reading Hint. Closes Bug 585.
- October 6th, 2003 [alex]: Changed the style of the Contents subsections in Chapters 5 and 6 and Appendix A.
- October 6th, 2003 [greg]: Simplified seds in "Locking in Glibc" and "Re-adjusting the toolchain" sections. Rearranged "How things are going to be done" section.
- October 5th, 2003 [greg]: Chapter 5: Added new section "Toolchain technical notes". Integrated and scaled back the old "Why we use static linking" section. Closes Bug 658.
- October 4th, 2003 [alex]: Minor rewordings and additions of markup here and there.
- October 4th, 2003 [greg]: Chapter 5 - Binutils Pass 1: Added extra LDFLAGS to ensure static rebuild of ld.
- October 2nd, 2003 [greg]: Chapter 6: Reinstated INSTALL=/tools/bin/install for linker adjustment command due to issues on hosts where a ginstall symlink exists. This renders the "install" symlinks redundant, so removed those too.
- October 2nd, 2003 [greg]: Chapter 6 - Shadow: Enabled MD5 passwords. Closes Bug 600.
- September 27th, 2003 [greg]: Chapter 5 - Expect: Tweaked install so that redundant scripts are not installed. Chapter 6 - Creating essential symlinks: Removed redundant links. Chapter 6 - man: Removed PATH, closes Bug 574.
- September 27th, 2003 [greg]: Added Tcl, Expect and DejaGnu items to Appendix A. Closes Bug 661.
- September 26th, 2003 [jeremy]: Added new workaround for the devpts problems.
- September 24th, 2003 [greg]: Various changes across the board addressing Bug 675.
- September 24th, 2003 [alex]: Appendix A - Changed the style of the short descriptions, and the content of most of them too.
- September 22nd, 2003 [greg]: Chapter 8 - Creating the /etc/fstab file: Made mounting devpts the default.
- September 22nd, 2003 [jeremy]: Added Net-tools patch to fix mii-tool compilation.
- September 22nd, 2003 [jwrober]: Chapter 5 - Updated the Why Static page to more accurately represent the difference between statically and dynamically linked binaries. Thanks to Ian Molton for pointing this out. Fixes Bug 602.
- September 22nd, 2003 [jeremy]: Removed the make command from DejaGnu, since it performs nothing.

- September 22nd, 2003 [jeremy]: Removed the -k from Tcl's make check, since it's not expected to have failures anymore
- September 22nd, 2003 [jeremy]: Changed the reference to the man hint to a pointer to BLFS.
- September 22nd, 2003 [jeremy]: Added a note to remember to mount devpts if you exit and re-enter chroot.
- September 22nd, 2003 [jeremy]: Removed make check from Patch and Diffutils, since these tests perform no actions.
- September 22nd, 2003 [greg]: Chapter 5 - Setting up the environment: Added unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD to .bash_profile to stop accidental build breakage.
- September 20th, 2003 [greg]: Chapter 5 - GCC Pass 2: Updated to gcc-3.3.1-specs-2.patch. Ncurses: added --enable-overwrite and description.
- September 19th, 2003 [jeremy]: Corrected bash tags for proper use of the +h flag to bash.
- September 19th, 2003 [jwrober]: Various updates to the acknowledgments page.
- September 18th, 2003 [jeremy]: Chapter 5 - GCC Pass 2: Added some extra comments regarding the 3 tarballs to unpack.
- September 17th, 2003 [greg]: Chapter 6 - GCC-2.95.3: Added rationale notes.
- September 17th, 2003 [jwrober]: Updated the acknowledgments page to match the website.
- September 17th, 2003 [jeremy]: Upgraded File to 4.04.
- September 17th, 2003 [jeremy]: Chapter 6 - Changed 2 of the occurrences of exec bash --login to include the +h directive.
- September 17th, 2003 [greg]: Chapters 5 and 6 - Locking in Glibc and Re-adjusting the toolchain: Do "make -C ld install" instead of "make -C ld install-data-local" to install a whole new linker instead of just the new ldscripts.
- September 17th, 2003 [alex]: Normalized the spelling of 'Tcl' and 'DejaGnu', following their own documentation.
- September 17th, 2003 [alex]: Properly alphabetized the dependencies.
- September 16th, 2003 [alex]: Finally updated the dependencies for the new Coreutils.
- September 16th, 2003 [greg]: Chapters 5 and 6 - Locking in Glibc and Re-adjusting the toolchain: Added sanity checks.
- September 16th, 2003 [greg]: Chapters 5 and 6 - Binutils, GCC, and Glibc: Added notes on the test suites.

- September 15th, 2003 [alex]: Corrected several typos and some inconsistencies.
- September 14th, 2003 [greg]: Chapter 6 - Revised chroot command: Removed no longer needed set +h.
- September 14th, 2003 [alex]: Fixed some typos, and added some markup. Dropped the removal of program files from the Stripping section in Chapter 5.
- September 14th, 2003 [greg]: Chapter 6 - Create essential symlinks: Add symlink /usr/lib/libgcc_s.so.1 to allow GCC abi_check to run. Future NPTL needs this as well.
- September 13th, 2003 [jwrober]: Added PLFS hint text to the page in Chapter 6 for creating passwd and group: bug 596.
- September 13th, 2003 [jwrober]: Updated the "How things are going to be done" page to include more of the PLFS hint's text.
- September 13th, 2003 [jwrober]: Preface - Merged whoread and whonotread into a single audience page.
- September 13th, 2003 [greg]: Chapter 2 - Added new section about the test suites.
- September 12th, 2003 [jeremy]: Chapter 5 - Ncurses: Added description for the --without-ada configure switch.
- September 12th, 2003 [jeremy]: Chapter 5 - Gawk: Added the test suite
- September 12th, 2003 [jeremy]: Chapter 5 - Grep: Added descriptions of configure switches courtesy of Anderson Lizardo
- September 12th, 2003 [gerard]: Removed /usr/lib/locale directory creation - it's created during Chapter 6 - Glibc where it's more relevant.
- September 11th, 2003 [jwrober]: Chapter 5 - Fixed GCC Pass 2 specs patch text to be more vague, but in actuality more accurate - provided by Anderson Lizardo.
- September 11th, 2003 [jwrober]: Chapter 5 - Grammar fix in Tcl install directions provided by Anderson Lizardo.
- September 11th, 2003 [jwrober]: Chapter 5 - Small textual change in the locking in Glibc page for /lib/ld.so.1 provided by Anderson Lizardo.
- September 11th, 2003 [jeremy]: Added bootloader setup to Chapter 8, after the addition of Grub to the book.
- September 11th, 2003 [gerard]: Removed Bin86 and LILO and replaced it with Grub.
- September 11th, 2003 [jeremy]: Dropped non-toolchain tests to optional actions. Added a note to use the Wiki for failed tests.

- September 11th, 2003 [jeremy]: Added Bison patch, backported from CVS, to fix pwlib compilation problems
- September 11th, 2003 [jeremy]: Added Greg's patch to GCC to suppress the installation of libiberty, and changed Binutils to allow its libiberty to stay.
- September 11th, 2003 [jeremy]: Added caution tags around the reminder to not delete the Binutils source and build directories in Chapter 5.
- September 11th, 2003 [jeremy]: Added new perl-libc-3 patch from Anderson Lizardo
- September 9th, 2003 [jwrober]: Fixed the Findutils package download link on the packages page closing bug 578.
- September 9th, 2003 [jeremy]: Chapter 6 - GCC 2.95.3: Removed compilation of C++, added Zack's return-type patch.
- September 9th, 2003 [jeremy]: Chapter 6 - Coreutils: Added coreutils-5.0-hostname-2.patch, which suppresses the build of the hostname binary, and also suppresses its check.
- September 9th, 2003 [jeremy]: Added some notes regarding failed tests to Glibc and DejaGnu.
- September 9th, 2003 [jeremy]: Glibc - Added commands to both Chapter 5 and 6 to include minimum locales necessary for checks.
- September 9th, 2003 [jeremy]: Chapter 6 - Removed Zlib's munging of CFLAGS in favor of a note to add -fPIC.
- September 8th, 2003 [matt]: Chapter 5 - Fixed the rm command that deletes unneeded documentation from /tools/share.
- September 6th, 2003 [matt]: Chapter 6 - Removed a reference to "the static" directory in the intro.
- September 6th, 2003 [jeremy]: Chapter 4 - Updated download locations for some packages.
- September 5th, 2003 [jeremy]: Chapter 5 - GCC Pass 2: Corrected the make check error explanation
- September 5th, 2003 [jeremy]: Chapter 6 - Makedev: Changed the default device creation to generic-nopty, because we now use devpts by default.
- September 5th, 2003 [jeremy]: Chapter 6 - GCC: Corrected wording to reflect the removal of the /usr/lib/cpp symlink.
- September 5th, 2003 [jeremy]: Corrected perl libc patch to -2, changing the old /stage1 structure to /tools
- September 5th, 2003 [matt]: Chapter 6 - Updated GCC specs patch and upgraded to man-1.5m2

- September 4th, 2003 [jeremy]: Chapter 6 - Creating Directories: Eliminated the creation of /usr/tmp - Closes bug 176.
- September 4th, 2003 [jeremy]: Chapter 6 - Mounting Proc: Added mounting the devpts filesystem into chroot here. Closes bug 533.
- September 4th, 2003 [jeremy]: Chapter 6 - Mounting Proc: Added a warning at the end regarding checking that proc is still mounted if you stop and restart the lfs process.
- September 4th, 2003 [jeremy]: Chapter 6 - Gzip: Altered text to better explain the reason behind the sed command used in the gzip installation. Closes bug 551.
- September 4th, 2003 [jeremy]: Chapter 4 - Downloading patches: Added a note regarding Tushar's patches project, and a link to the patches home page.
- September 3rd, 2003 [matt]: Fixed issue with Util-linux not utilizing headers and libraries installed in /stage1.
- September 3rd, 2003 [matt]: Removed "rm /bin/pwd" instruction from Chapter 6 kernel-headers installation as the link is still required by Glibc's installation.
- September 2nd, 2003 [alex]: Adjusted all the SBUs from the values posted by Jeremy.
- September 2nd, 2003 [alex]: Finally got around to renaming /stage1 to /tools.
- September 2nd, 2003 [alex]: Merged several of the main book structure files.
- September 2nd, 2003 [alex]: Alphabetized download lists, added note to Tcl instructions.
- September 2nd, 2003 [alex]: Reworded Organization, $LFS and SBUs sections.
- September 1, 2003 [jeremy] - Chapter 6 - Groff - Added note about choice of A4 or letter for the PAGE variable.
- September 1, 2003 [jeremy] - Added in shadow newgrp patch from Greg Schafer
- August 31, 2003 [jeremy] - Chapter 6 - Inetutils - added the --disable-whois and --disable-servers flags
- August 31, 2003 [jeremy] - Added in Greg's new instructions for GCC 3.3.1 with respect to the fixincludes process. Also added extra verbiage to the Locking in Glibc and GCC Pass 2 pages on the fixincludes process.
- August 31st, 2003 [alex]: Reworded some paragraphs, added missing markup, and rearranged the changelog.
- August 31st, 2003 [alex]: Wrapped the 'Last checked' lines in parentheses. Several other small retouches.
- August 30, 2003 [jeremy] - Updated fix-includes patch to GCC 3.3.1

- August 29, 2003 [jeremy] - Glibc - updated instructions with the sscanf patch from patches.
- August 29, 2003 [jeremy] - Updated GCC to version 3.3.1, including fixes based on Zack's mini-hint for GCC 3.3, and patches from his docs.
- August 29th, 2003 [alex]: Removed obsolete Netkit-base, Fileutils, Sh-utils, and Textutils files.
- August 29th, 2003 [alex]: Added some missing markup, changed a few /static's to /stage1's.
- August 29th, 2003 [alex]: Chapter 06 - Added all the missing text lines before the make checks, and reworded other lines.
- August 28, 2003 [matt] - Updated packages to linux-2.4.22, man-pages-1.60, expect-5.39.0, findutils-4.1.20 and tcl-8.4.4
- August 28, 2003 [jeremy] - New bash-2.05b-2.patch file to include the 7 patches from ftp.gnu.org
- August 28th, 2003 [alex]: Chapter 06 - Re-adjusting toolchain: Added a forgotten backslash.
- August 28th, 2003 [alex]: Fixed a few typos and added some missing markup.
- August 28th, 2003 [alex]: Chapter 06 - Binutils and GCC: Integrated text from the pure-lfs hint.
- August 27, 2003 [jeremy] - Chapter 06 - Inetutils: Added --sysconfdir=/etc --localstatedir=/var and moved the ping binary from /usr/bin to /bin
- August 27th, 2003 [alex]: Chapter 06 - Glibc: Integrated text from the pure-lfs hint.
- August 26, 2003 [jeremy] - Chapter 07 - Creating /etc/hosts: Changed www.mydomain.org to <value of HOSTNAME>.mydomain.org
- August 26th, 2003 [alex]: Chapter 06 & 08 - Moved the installation of the kernel manpages from chapter 6 to 8.
- August 26, 2003 [jeremy] - Chapter 04 - Mounting the LFS partition: Added text regarding mounting with too restrictive permissions.
- August 26, 2003 [jeremy] - Chapter 06 - Creating Directories: Added the creation of the /dev/shm directory.
- August 26, 2003 [jeremy] - Chapter 08 - Creating fstab: Added the mount of tmpfs filesystem to /dev/shm.
- August 26, 2003 [jeremy] - Chapter 08 - Kernel Installation: Added a reminder to compile tmpfs support into the kernel.
- August 25th, 2003 [alex]: Chapter 06 - Rewrote the installation text of Shadow and Util-Linux while correcting some typos.

- August 25th, 2003 [alex]: Chapter 05 & 06 - Made the "Locking in" and "Re-adjusting" look similar.
- August 24th, 2003 [alex]: Chapter 04 - Merged the many little files into one file. Gave packages and patches a separate page.
- August 17th, 2003 [alex]: Chapter 05 - From Bash to Perl: put text in between commands. Added a section on stripping unneeded symbols to decrease the size of the tools.
- August 16th, 2003 [alex]: Chapter 05 - From Make to Texinfo: put text in between commands.
- August 11th, 2003 [alex]: Chapter 05 - From Binutils Pass 1 to Findutils: several small textual adjustments. For the second passes not giving the contents and dependencies.
- August 11th, 2003 [alex]: Chapter 04 - Listed separate core, g++, and test suite tarballs for GCC.
- August 11th, 2003 [alex]: Chapter 04 - Suppressed the mention of a wget script.
- August 9th, 2003 [alex]: Chapter 05 - Binutils Pass 2 and GCC Pass 2: integrated some text from the pure-lfs hint.
- August 8th, 2003 [alex]: Chapter 05 - Tcl, Expect, and DejaGnu: added some text.
- August 6th, 2003 [gerard]: Applied Alex Groenewoud's patch that adds Appendix B, providing a list of all installed programs and libraries plus references to the installation pages.
- July 30th, 2003 [gerard]: Chapter 06 - Vim: Changed the way the global `vimrc` and `gvimrc` locations are defined.
- July 30th, 2003 [gerard]: Chapter 05 - Binutils Pass 2: removed the lib patch, it's no longer needed with the binutils-2.14 upgrade.
- July 30th, 2003 [gerard]: Chapter 05 Binutils Pass 1: Added `make configure-host`.
- July 30th, 2003 [gerard]: Upgraded to binutils-2.14, linux-2.4.21, expect-5.38.4, gawk-3.1.3, texinfo-4.6, util-linux-2.12, man-pages-1.58, lfs-utils-0.3, vim-6.2, gettext-0.12.1, automake-1.7.6, file-4.03, e2fsprogs-1.34, procps-3.1.11, psmisc-21.3
- June 3rd, 2003 [gerard]: Chapter 06 - Gawk: removed the removal of `/bin/awk`. This symlink isn't created anymore.
- May 21st, 2003 [gerard]: Chapter 06 - GCC-2.95.3: Added /opt/gcc-2.95.3/lib to the /etc/ld.so.conf file so the libraries can be found during run-time.
- May 21st, 2003 [gerard]: Chapter 05 - Gzip: Simplified commands.

- May 21st, 2003 [gerard]: Chapter 05 - Bzip2: Simplified commands.
- May 21st, 2003 [gerard]: Chapter 06 - Shadow: Added the `grpconv` command to complement the enabling of all shadowed passwords.
- May 21st, 2003 [winkie]: Chapter 06 - Creating Files: All those `ln` commands can be made into a few long ln commands.
- May 21st, 2003 [winkie]: Chapter 05 - Installing Glibc: Create an ld.so.conf file before building Glibc, to prevent an (harmless) error.
- May 21st, 2003 [winkie]: Chapter 06 - Installing Glibc: Don't bother doing the 'exec /stage1/bin/bash' stuff, it doesn't do anything now that we use PLFS.
- May 21st, 2003 [winkie]: Chapter 05 & 06 - Installing Coreutils: Only test the non-root stuff in Chapter 05, but test everything in Chapter 06.
- May 21st, 2003 [winkie]: Chapter 05 - Installing Expect: Don't bother passing anything more than --prefix=/stage1. None of it is needed.
- May 16th, 2003 [gerard]: Chapter 06: Net-tools: Changed `make install` to `make update`.
- May 15th, 2003 [timothy]: Chapter 05: Installing Patch: Added `CPPFLAGS=-D_GNU_SOURCE` before `./configure` to fix patch build on PPC.
- May 13th, 2003 [gerard]: Chapter 06: When we `exec /path/to/bash --login`, also run `set +h` to keep the hashing option turned off. Fixes bug #531
- May 13th, 2003 [gerard]: Chapter 06 - Basic Network: Changed the single quotes to double quotes in the echo command. Without it, $(hostname) won't expand which defeats the sole purpose of this command - to make Perl's hostname check work.
- May 13th, 2003 [winkie]: Removed all occurrences &&. Updated bug syntax. Added "make check/test" where necessary in Chapter 6.
- May 13th, 2003 [winkie]: Chapter 06: Applied "Changing ownership" patch to polish the text. Closes bug #511.
- May 13th, 2003 [winkie]: Chapter 06: Applied "Configuring system components" patch to polish the text. Closes bug #510.
- May 13th, 2003 [gerard]: Chapter 06: Removed Tcl, Expect and DejaGnu. Nothing uses this once past GCC in chapter 6. The versions in /stage1/bin do the job just fine.
- May 13th, 2003 [winkie]: Chapter 06 - Installing Shadow: Touching the /usr/bin/passwd file before installation. Not doing so results in Shadow thinking passwd will be in /bin/passwd.
- May 13th, 2003 [winkie]: Chapter 06 - Installing Procps: Remove the /lib/libproc.so symlink. No package outside of Procps itself uses this library, and none should.

- May 13th, 2003 [winkie]: Chapter 06 - Installing Net-tools: Run "make config" before doing make. Fixes bugs #462 and #497.
- May 13th, 2003 [gerard]: Chapter 06 - Ncurses: Added the vsscanf patch.
- May 12th, 2003 [gerard]: Chapter 05 - Gzip: Removed `make check`. It doesn't do anything.
- May 12th, 2003 [winkie]: Chapter 05 - Installing Texinfo: Don't install the texmf data. It's not used by anything.
- May 12th, 2003 [winkie]: Chapter 05 & 06 - Installing Ncurses: In Chapter 6, symlink creation has been updated to include libcurses.*, and libncurses++.a has its properties changed to 644. Chapter 5 doesn't need any libcurses.* so those are removed.
- May 12th, 2003 [gerard]: Chapter 06 - Basic Network: Added $(hostname) to /etc/hosts, without it Perl's hostname test doesn't pass.
- May 12th, 2003 [gerard]: Chapter 06 - Installing GCC: Don't try to remove /usr/include/libiberty.h. It isn't installed in the first place.
- May 12th, 2003 [winkie]: Upgraded to findutils-4.1.7, gzip-1.3.5, and tar-1.13.25.
- May 12th, 2003 [winkie]: Chapter 05 - Installing Perl: Added extra commands to build certain modules into Perl. This is to accommodate the Coreutils "make check". Partially fixes bug #528.
- May 12th, 2003 [winkie]: Chapter 05 - Installing Gzip: Nothing in Chapter 6 checks for or uses the uncompress command, therefore we shouldn't create it.
- May 12th, 2003 [winkie]: Chapter 05 - Installing Bzip2: Running "make" implies "make check", therefore there is no reason whatsoever for us to run it manually.
- May 12th, 2003 [winkie]: Chapter 05 - Installing Lfs-Utils: Removed. The only package that checks for mktemp before it is installed is GCC, and that's only for gccbug.
- May 11th, 2003 [gerard]: Chapter 06 - GCC-2.95.3: Added --enable-threads=posix as well to complete the C++ addition.
- May 11th, 2003 [gerard]: Chapter 06 - GCC-2.95.3: Added --enable-languages=c,c++ to fix that GCC's version bug with regards to -Wreturn-type. Fixes bug #525
- May 11th, 2003 [gerard]: Chapter 05 - Bash: Removed the --without-bash-malloc configure option.
- May 11th, 2003 [gerard]: Updated to gcc-3.2.3-specs-4.patch.
- May 11th, 2003 [winkie]: Chapter 06 - Setting up Basic Networking: Added section. Create a basic /etc/hosts files, and create /etc/services and /etc/protocols from IANA. Fixes bugs #359 & #515.

- May 11th, 2003 [winkie]: Upgrading to lfs-utils-0.2.2. This adds two files needed for proper networking configuration.
- May 11th, 2003 [winkie]: Removed Netkit-base 0.17. Added Inetutils 1.4.2. Fixes bug #490.
- May 11th, 2003 [winkie]: Added lfs-utils-0.2.1. Fixes bug #493.
- May 11th, 2003 [winkie]: Chapter 06 - Installing Ncurses: Fix up the symlinks so that they follow suit of other library symlinks. No more weirdness here.
- May 11th, 2003 [winkie]: Chapter 06 - Installing Procps: Removed XSCPT="" cruft and its corresponding paragraph. This stuff isn't needed anymore.
- May 11th, 2003 [winkie]: Chapter 06 - Installing Ncurses: Pass --without-debug to the configure script. It seems to have gotten lost at some point.
- May 11th, 2003 [timothy]: Chapter 5 & 6 - Installing Bzip2, Installing Zlib: Modified build commands per bug #524.
- May 11th, 2003 [winkie]: Chapter 06 - Installing Glibc: Install the linuxthreads man pages, too. This got lost somewhere.
- May 11th, 2003 [winkie]: Chapter 06 - Installing Grep: Added --with-included-regex to prevent Grep from using Glibc's somewhat bugged regex.
- May 11th, 2003 [winkie]: Chapter 06 - Installing Coreutils: Fix some functionality of the uname command with a patch.
- May 11th, 2003 [winkie]: Chapter 06 - Installing Net-tools: Just do regular old "make install" instead of "make update". The latter works fine now.
- May 11th, 2003 [winkie]: Chapter 06 - Installing GCC: After installation, remove /usr/include/libiberty.h. It is not used outside of the GCC build tree.
- May 11th, 2003 [winkie]: Upgraded to Bash 2.05b and added its patch.
- May 11th, 2003 [winkie]: Chapter 06 - Installing Zlib: Apply a patch to fix the buffer overflow in gzprintf().
- May 11th, 2003 [winkie]: Chapter 06 - Configuring system components: Moved the creation of the btmp, wtmp, lastlog and utmp to just before Shadow, so that they are detected at their proper locations.
- May 10th, 2003 [winkie]: Chapter 06 - Installing Automake: Run "make" before installing. This is needed now with the newer releases of Automake.
- May 10th, 2003 [winkie]: Chapter 06 - Installing Vim: Removed the patch. It hasn't been required since GCC 3.2.1.
- May 10th, 2003 [winkie]: Chapter 06 - Creating the mtab file: Removed. Mounting /proc has the side effect of creating /etc/mtab for us.
- May 10th, 2003 [winkie]: Chapter 06 - Installing Make: Removed modification of /usr/bin/make file. It is no longer mistakenly installed with strange ownership or permissions.

- May 10th, 2003 [winkie]: Chapter 06 - Installing Glibc: Made /etc/localtime a file instead of a symlink. The symlink method breaks on systems where /usr is a separate partition.
- May 10th, 2003 [winkie]: Chapter 06 - Installing E2fsprogs: Removed install-info commands for e2fsprogs. The "make install" target handles this for us.
- May 10th, 2003 [gerard]: Removed all CFLAGS and LDFLAGS variables where they are not essential (so, not including static binutils, GCC and compiling Zlib with -fPIC).
- May 10th, 2003 [gerard]: Chapter 05 - Binutils (pass1, pass2), locking in Glibc and adjusting toolchain: Changed tooldir to /stage1 (likewise we use tooldir=/usr in Chapter 6).
- May 10th, 2003 [gerard]: Chapter 05 - Kernel headers: Removed the usage of `cp -H` because there are distributions out there that do not know about the `-H` option.
- May 10th, 2003 [gerard]: New gcc-3.2.3-specs-3.patch.
- May 10th, 2003 [gerard]: Chapter 06 - Adjusting toolchain: Made it more architecture-independent.
- May 10th, 2003 [gerard]: Chapter 05 - Locking in Glibc: Made it more architecture-independent.
- May 7th, 2003 [gerard]: Removed GCC No Debug patches. No longer assume gcc-core and gcc-g++ packages are downloaded, so added appropriate --enable-languages options.
- May 7th, 2003 [gerard]: Removed Chapter 6 - Glibc-Pass2. It's not needed anymore with the pure-lfs integration.
- May 7th, 2003 [gerard]: Downgraded to flex-2.5.4a again. Newer versions just don't work properly.
- May 5th, 2003 [gerard]: Removed zlib installation from chapter 5 (its inclusion was a mistake).
- May 5th, 2003 [gerard]: Various bug fixes that were introduced during the pure-lfs integration.
- May 2nd, 2003 [gerard]: Upgraded to: automake-1.7.4, e2fsprogs-1.33, file-4.02, flex-2.5.31, gawk-3.1.2, gcc-3.2.3, glibc-2.3.2, grep-2.5.1, groff-1.19, less-381, libtool-1.5, man-1.5l, man-pages-1.56, modutils-2.4.25, procps-3.1.8, sed-4.0.7, sysvinit-2.85, texinfo-4.5, util-linux-2.11z
- May 2nd, 2003 [gerard]: Removed fileutils-4.1, sh-utils-2.0, textutils-2.1 (all replaced with coreutils-5.0).
- May 2nd, 2003 [gerard]: Added binutils-2.13.2-libc.patch, coreutils-5.0, dejagnu-1.4.3, expect-5.38, gawk-3.1.2, gcc-2.95.3, tcl-8.4.2

- May 2nd, 2003 [gerard] - Integrated new installation method from the Pure LFS hint written by Greg Schafer and Ryan Oliver.

Release of version 4.1 on April 28th, 2003.

Resources

FAQ

If during the building of your LFS system you encounter any errors, or have any questions, or think you found a typo in the book, then please first consult the FAQ (Frequently Asked Questions) at `http://www.linuxfromscratch.org/faq/`.

IRC

Several members of the LFS community offer assistance on our community IRC server. Before you utilize this mode of support, we ask that you've at least checked the LFS FAQ and the mailing list archives for the answer to your question. You can find the IRC server at *irc.linuxfromscratch.org* port 6667. The support channel is named #LFS-support.

Mailing lists

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

For information on which lists are available, how to subscribe to them, their archive locations, and so on, visit `http://www.linuxfromscratch.org/mail.html`.

News server

All the mailing lists hosted at *linuxfromscratch.org* are also accessible via the NNTP server. All messages posted to a mailing list will be copied to the correspondent newsgroup, and vice versa.

The news server can be reached at *news.linuxfromscratch.org*.

Mirror sites

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

Contact information

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

But if you need to reach Gerard Beekmans personally, send an email to gerard@linuxfromscratch.org.

Acknowledgments

We would like to thank the following people and organizations for their contributions to the Linux From Scratch Project.

Current Project Team Members

- Gerard Beekmans <gerard@linuxfromscratch.org> — Linux-From-Scratch initiator, LFS Project organizer.
- Matthew Burgess <matthew@linuxfromscratch.org> — LFS General Package maintainer, LFS Book editor.
- Craig Colton <meerkats@bellsouth.net> — LFS, ALFS, BLFS and Hints Project logo creator.
- Jeroen Coumans <jeroen@linuxfromscratch.org> — Website developer, FAQ maintainer.
- Bruce Dubbs <bdubbs@linuxfromscratch.org> — LFS Quality Assurance Team leader, BLFS Book editor.
- Alex Groenewoud <alex@linuxfromscratch.org> — LFS Book editor.
- Mark Hymers <markh@linuxfromscratch.org> — CVS maintainer, BLFS Book creator, former LFS Book editor.
- James Iwanek <iwanek@linuxfromscratch.org> — System Administration Team member.
- Nicholas Leippe <nicholas@linuxfromscratch.org> — Wiki maintainer.
- Anderson Lizardo <lizardo@linuxfromscratch.org> — Website backend scripts creator and maintainer.
- Bill Maltby <bill@linuxfromscratch.org> — LFS Project organizer.
- Scot Mc Pherson <scot@linuxfromscratch.org> — LFS NNTP gateway maintainer.
- Ryan Oliver <ryan@linuxfromscratch.org> — Testing Team leader, co-creator of PLFS.
- James Robertson <jwrober@linuxfromscratch.org> — Bugzilla maintainer, Wiki developer, LFS Book editor.

- Greg Schafer <greg@linuxfromscratch.org> — Toolchain maintainer, LFS Book editor, co-creator of PLFS.
- Tushar Teredesai <tushar@linuxfromscratch.org> — BLFS Book editor, Hints and Patches Projects maintainer.
- Jeremy Utley <jeremy@linuxfromscratch.org> — LFS Book editor, Bugzilla maintainer.
- Countless other people on the various LFS and BLFS mailing lists who are making this book happen 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.

Mirror Maintainers

- Jason Andrade <jason@dstc.edu.au> — au.linuxfromscratch.org mirror.
- William Astle <lost@l-w.net> — ca.linuxfromscratch.org mirror.
- Baque <baque@cict.fr> — lfs.cict.fr mirror.
- Stephan Brendel <stevie@stevie20.de> — lfs.netservice-neuss.de mirror.
- Ian Chilton <ian@ichilton.co.uk> — us.linuxfromscratch.org, linuxfromscratch.co.uk mirrors.
- Fredrik Danerklint <fredan-lfs@fredan.org> — se.linuxfromscratch.org mirror.
- David D.W. Downey <pgpkeys@aeternamtech.com> — lfs.learnbyexample.com mirror.
- Eduardo B. Fonseca <ebf@aedsolucoes.com.br> — br.linuxfromscratch.org mirror.
- Hagen Herrschaft <hrx@hrxnet.de> — de.linuxfromscratch.org mirror.
- Tim Jackson <tim@idge.net> — linuxfromscratch.idge.net mirror.
- Barna Koczka <barna@siker.hu> — hu.linuxfromscratch.org mirror.
- Roel Neefs — linuxfromscratch.rave.org mirror.

- Simon Nicoll <sime@dot-sime.com> — uk.linuxfromscratch.org mirror.
- Ervin S. Odisho <ervin@activalink.net> — lfs.activalink.net mirror.
- Guido Passet <guido@primerelay.net> — nl.linuxfromscratch.org mirror.
- Mikhail Pastukhov <miha@xuy.biz> — lfs.130th.net mirror.
- Jeremy Polen <jpolen@rackspace.com> — us2.linuxfromscratch.org mirror.
- UK Mirror Service — linuxfromscratch.mirror.co.uk mirror.
- Thomas Skyt <thomas@sofagang.dk> — dk.linuxfromscratch.org mirror.
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> — at.linuxfromscratch.org mirror.
- Dag Stenstad <dag@stenstad.net> for providing no.linuxfromscratch.org and Ian Chilton for running it.
- Parisian sysadmins <archive@doc.cs.univ-paris8.fr> — www2.fr.linuxfromscratch.org mirror.
- Jesse Tie-Ten-Quee <highos@linuxfromscratch.org> for providing and running the linuxfromscratch.org server.
- Alexander Velin <velin@zadnik.org> — bg.linuxfromscratch.org mirror.
- Martin Voss <Martin.Voss@ada.de> — lfs.linux-matrix.net mirror.
- Pui Yong <pyng@spam.averse.net> — sg.linuxfromscratch.org mirror.

Donators

- Dean Benson <dean@vipersoft.co.uk> for several monetary contributions.
- DREAMWVR.COM for their past sponsorship of donating various resources to the LFS and related sub projects.
- Hagen Herrschaft <hrx@hrxnet.de> for donating a 2.2 GHz P4 system, now running under the name of *lorien*.
- O'Reilly for donating books on SQL and PHP.
- VA Software who, on behalf of Linux.com, donated a VA Linux 420 (former StartX SP2) workstation.
- Mark Stone for donating *shadowfax*, the first linuxfromscratch.org server, a 750 MHz P3 with 512 MB RAM and two 9 GB SCSI drives. When the server moved it was renamed to *belgarath*.
- Jesse Tie-Ten-Quee <highos@linuxfromscratch.org> for donating a Yamaha CDRW 8824E CD-writer.
- Countless other people on the various LFS mailing lists who are making this book better by giving their suggestions, submitting bug reports, and throwing in their criticism.

Former Team Members and Contributors

- Timothy Bauscher <timothy@linuxfromscratch.org> — LFS Book editor, Hints Project maintainer.
- Robert Briggs for originally donating the *linuxfromscratch.org* and *linuxfromscratch.com* domain names.
- Ian Chilton <ian@ichilton.co.uk> for maintaining the Hints project.
- Marc Heerdink <gimli@linuxfromscratch.org> — LFS Book editor.
- Seth W. Klein <sklein@linuxfromscratch.org> — LFS FAQ creator.
- Garrett LeSage <garrett@linuxart.com> — Original LFS banner creator.
- Simon Perreault <nomis80@videotron.ca> — Hints Project maintainer.
- Geert Poels <Geert.Poels@skynet.be> — Original BLFS banner creator; based on the LFS banner by Garrett LeSage.
- Frank Skettino <bkenoah@oswd.org> for the initial design of the old website — have a look at http://www.oswd.org/.
- Jesse Tie-Ten-Quee <highos@linuxfromscratch.org> for answering countless questions on IRC and having a great deal of patience.

Chapter 2
Important information

About $LFS

Please read the following paragraphs carefully. Throughout this book the variable LFS will be used frequently. $LFS must at all times be replaced with the directory where the partition that contains the LFS system is mounted. How to create and where to mount the partition will be explained in full detail in Chapter 3. For the moment let's assume that the LFS partition is mounted on `/mnt/lfs`.

When you are told to run a command like `./configure --prefix=$LFS/tools`, you actually have to execute `./configure --prefix=/mnt/lfs/tools`.

It's important that this is done no matter where it is read; be it in commands entered in a shell, or in a file edited or created.

A possible solution is to set the environment variable LFS. This way $LFS can be entered literally instead of replacing it with /mnt/lfs. This is accomplished by running:

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

Now, if you are told to run a command such as `./configure --prefix=$LFS/tools`, then you may type it literally. Your shell will replace "$LFS" with "/mnt/lfs" when it processes the command line (that is, when you hit Enter after having typed the command).

About SBUs

Most people would like to know beforehand how long it approximately takes to compile and install each package. But "Linux from Scratch" is built on so many different systems, it is not possible to give actual times that are anywhere near accurate: the biggest package (Glibc) won't take more than twenty minutes on the fastest systems, but will take something like three days on the slowest — no kidding. So instead of giving actual times, we've come up with the idea of using the *Static Binutils Unit* (abbreviated to *SBU*).

It works like this: the first package you compile in this book is the statically linked Binutils in Chapter 5, and the time it takes to compile this package is what we call the "Static Binutils Unit" or "SBU". All other compile times will be expressed relative to this time.

For example, the time it takes to build the static version of GCC is 4.4 SBUs. This means that if on your system it took 10 minutes to compile and install the static Binutils, then you know it will take approximately 45 minutes to build the static GCC. Fortunately, most build times are much shorter than the one of Binutils.

Note that if the system compiler on your host is GCC-2 based, the SBUs listed may end up being 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-3.3.1 which is known to be approximately 30% slower.

Also note that SBUs don't work well for SMP-based machines. But if you're so lucky as to have multiple processors, chances are that your system is so fast that you won't mind.

About the test suites

Most packages provide a test suite. Running the test suite for a newly built package is generally a good idea as it can provide a nice sanity check that everything compiled correctly. A test suite that passes its set of checks usually proves that the package is functioning mostly 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 (the C library) — are of the utmost importance due to their central role in a properly functioning system. But be warned, the test suites for GCC and Glibc can take a very long period of time to complete, especially on slower hardware.

As you progress through the book and encounter the build commands to run the various test suites, we'll guide you on the relative importance of the test suite in question so that you can decide for yourself whether to run it or not.

A common problem when running the test suites for Binutils and GCC is running out of pseudo terminals (PTYs for short). The symptom is an unusually high number of failing tests. This can happen for any number of reasons. Most likely is that the host system doesn't have the *devpts* file system set up correctly. We'll discuss this in more detail later on in Chapter 5.

How to ask for help

If you encounter a problem while using this book, and your problem is not listed in the FAQ (`http://www.linuxfromscratch.org/faq`), you will find that most of the people on Internet Relay Chat (IRC) and on the mailing lists are willing to help you. An overview of the LFS mailing lists can be found in Chapter 1 - Mailing lists. To assist us in diagnosing and solving your problem, include as much relevant information as possible in your request for help.

Things to mention

Apart from a brief explanation of the problem you're having, the essential things to include in your request are:

- the version of the book you are using (being 5.0),
- the host distribution and version you are using to create LFS from,
- the package or section giving you problems,
- the exact error message or symptom you are receiving,
- whether you have deviated from the book at all.

(Note that saying that you've deviated from the book doesn't mean that we won't help you. After all, LFS is about choice. It'll just help us to see other possible causes of your problem.)

Configure problems

When something goes wrong during the stage where the configure script is run, look through the `config.log` file. This file may contain errors encountered during configure which weren't printed to the screen. Include those relevant lines if you decide to ask for help.

Compile problems

To help us find the cause of the problem, both screen output and the contents of various files are useful. The screen output from both the ./configure script and the make run can be useful. Don't blindly include the whole thing but on the other hand, don't include too little. As an example, here is some 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 just include the bottom section where it says:

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

and onwards. This isn't enough for us to diagnose the problem because it only tells us that *something* went wrong, not *what* went wrong. The whole section, as in the example above, is what should be included to be helpful, because it includes the command that was executed and the command's error message(s).

An excellent article on asking for help on the Internet in general has been written by Eric S. Raymond. It is available online at `http://catb.org/~esr/faqs/smart-questions.html`. Read and follow the hints in that document and you are much more likely to get a response to start with and also to get the help you actually need.

Test suite problems

Many packages provide a test suite which, depending on the importance of the package, we may encourage you to run. Sometimes packages will generate false or expected failures. If you encounter these, you can check the LFS Wiki page at `http://wiki.linuxfromscratch.org/` to see whether we have already investigated and noted them. If we already know about them, then usually there is no need to be concerned.

Part II - Preparing for the build

Chapter 3
Preparing a new partition

Introduction

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

Creating a new partition

In order to build our new Linux system, we will need some space: an empty disk partition. If you don't have a free partition, and no room on any of your hard disks to make one, then you could build LFS on the same partition as the one on which your current distribution is installed. This procedure is not recommended for your first LFS install, but if you are short on disk space, and you feel brave, take a look at the hint at `http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing-_systems.txt`.

For a minimal system you will need a partition of around 1.2 GB. This is enough to store all the source tarballs and compile all the packages. But if you intend to use the LFS system as your primary Linux system, you will probably want to install additional software, and will need more space than this, probably around 2 or 3 GB.

As we almost never have enough RAM in our box, 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 more urgent stuff. The swap partition for your LFS system can be the same one as for your host system, so you won't have to create another if your host system already uses a swap partition.

Start a disk partitioning program such as `cfdisk` or `fdisk` with an argument naming the hard disk upon which the new partition must be created — for example `/dev/hda` for the primary 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 don't yet know how to use the programs.

Remember the designation of your new partition — something like `hda5`. This book will refer to it as the LFS partition. If you (now) also have a swap partition, remember its designation too. These names will later be needed for the `/etc/fstab` file.

Creating a file system on the new partition

Now that we have a blank partition, we can create a file system on it. Most widely used in the Linux world is the second extended file system (ext2), but with the high-capacity hard disks of today the so-called 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/stable/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 (something like `hda5`).

If you created a (new) swap partition you need to initialize it as a swap partition too (also known as formatting, like you did above with `mke2fs`) by running:

```
mkswap /dev/yyy
```

Replace yyy with the name of the swap partition.

Mounting the new partition

Now that we've created a file system, we want to be able to access the partition. For that, we need to mount it, and have to choose a mount point. In this book we assume that the file system is mounted under `/mnt/lfs`, but it doesn't matter what directory you choose.

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

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

Now 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 you have decided to use multiple partitions for LFS (say one for `/` and another for `/usr`), mount them like this:

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

Of course, replace xxx and yyy with the appropriate partition names.

You should also ensure that this new partition is not mounted with permissions that are too restrictive (such as the nosuid, nodev or noatime options). You can run the `mount` command without any parameters to see with what options the LFS partition is mounted. If you see nosuid, nodev or noatime, you will need to remount it.

Now that we've made ourselves a place to work in, we're ready to download the packages.

Chapter 4
The materials: packages and patches

Introduction

Below is a list of packages you need to download 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 upon them. Unless you are an experienced LFS builder, we highly recommend not to try out newer versions, as the build commands for one version may not work with a newer version. Also, there is often a good reason for not using the latest version due to known problems that haven't been worked around yet.

All the URLs, when possible, refer to the project's page at `http://www.freshmeat.net/`. The Freshmeat pages will give you easy access to the official download sites as well as project websites, mailing lists, FAQs, changelogs and more.

We can't guarantee that these download locations are always available. In case a download location has changed since this book was published, please try to google for the package. Should you remain unsuccessful with this, you can consult the book's errata page at `http://linuxfromscratch.org/lfs/print/` or, better yet, try one of the alternative means of downloading listed on `http://linuxfromscratch.org/lfs/packages.html`.

You'll need to store all the downloaded packages and patches somewhere that is conveniently available throughout the entire build. You'll also need a working directory in which to unpack the sources and build them. A scheme that works well is to use `$LFS/sources` as the place to store the tarballs and patches, *and* as a working directory. This way everything you need will be located on the LFS partition and available during all stages of the building process.

So you may want to execute, as *root*, the following command before starting your download session:

```
mkdir $LFS/sources
```

And make this directory writable (and sticky) for your normal user — as you won't do the downloading as *root*, we guess:

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

All the packages

Download or otherwise obtain the following packages:

Autoconf (2.57) - 792 KB:
`http://freshmeat.net/projects/autoconf/`

Automake (1.7.6) - 545 KB:
`http://freshmeat.net/projects/automake/`

Bash (2.05b) - 1,910 KB:
`http://freshmeat.net/projects/gnubash/`

Binutils (2.14) - 10,666 KB:
`http://freshmeat.net/projects/binutils/`

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

Bzip2 (1.0.2) - 650 KB:
`http://freshmeat.net/projects/bzip2/`

Coreutils (5.0) - 3,860 KB:
`http://freshmeat.net/projects/coreutils/`

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

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

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

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

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

File (4.04) - 338 KB: (*) See Note Below
`http://freshmeat.net/projects/file/`

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

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

Gawk (3.1.3) - 1,596 KB:
`http://freshmeat.net/projects/gnuawk/`

GCC (2.95.3) - 9,618 KB:
`http://freshmeat.net/projects/gcc/`

GCC-core (3.3.1) - 10,969 KB:
`http://freshmeat.net/projects/gcc/`

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

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

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

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

Glibc-linuxthreads (2.3.2) - 211 KB:
`http://freshmeat.net/projects/glibc/`

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

Groff (1.19) - 2,360 KB:
`http://freshmeat.net/projects/groff/`

Grub (0.93) - 870 KB:
`ftp://alpha.gnu.org/pub/gnu/grub/`

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

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

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

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

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

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

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

Linux (2.4.22) - 28,837 KB:
`http://freshmeat.net/projects/linux/`

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

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

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

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

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

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

Ncurses (5.3) - 2,019 KB:
`http://freshmeat.net/projects/ncurses/`

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

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

Perl (5.8.0) - 10,765 KB:
`http://freshmeat.net/projects/perl/`

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

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

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

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

Shadow (4.0.3) - 760 KB:
`http://freshmeat.net/projects/shadow/`

Sysklogd (1.4.1) - 80 KB:
`http://freshmeat.net/projects/sysklogd/`

Sysvinit (2.85) - 91 KB:
`http://freshmeat.net/projects/sysvinit/`

Tar (1.13.25) - 1,281 KB:
`ftp://alpha.gnu.org/gnu/tar/`

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

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

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

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

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

Total size of these packages: 134 MB

File (4.04) may not be available by the time you read this. The master download location is known to remove old versions when new ones are released. Please refer to the corresponding section in Appendix A for an alternate download location.

Needed patches

Besides all those packages, you'll also need several patches. These correct tiny mistakes in the packages that should be fixed by the maintainer, or just make some small modifications to bend things our way. You'll need the following:

Bash Patch - 7 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/bash-2.05b-2.patch`

Bison Attribute Patch - 2 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/bison-1.875-attribute.patch`

Coreutils Hostname Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils-5.0-hostname-2.patch`

Coreutils Uname Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils-5.0-uname.patch`

Ed Mkstemp Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/ed-0.2-mkstemp.patch`

Expect Spawn Patch - 6 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/expect-5.39.0-spawn.patch`

Gawk Libexecdir Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gawk-3.1.3-libexecdir.patch`

GCC No-Fixincludes Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-no_fixincludes-2.patch`

GCC Specs Patch - 10 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-specs-2.patch`

GCC Suppress-Libiberty Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-suppress-libiberty.patch`

GCC-2 Patch - 16 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-2.patch`

GCC-2 No-Fixincludes Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-no-fixinc.patch`

GCC-2 Return-Type Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-returntype-fix.patch`

Glibc Sscanf Patch - 2 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/glibc-2.3.2-sscanf-1.patch`

Grub Gcc33 Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/grub-0.93-gcc33-1.patch`

Kbd More-Programs Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/kbd-1.08-more-programs.patch`

Man 80-Columns Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-80cols.patch`

Man Manpath Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-manpath.patch`

Man Pager Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-pager.patch`

Ncurses Etip Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses-5.3-etip-2.patch`

Ncurses Vsscanf Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses-5.3-vsscanf.patch`

Net-tools Mii-Tool-Gcc33 Patch - 2 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/net-tools-1.60-miitool-gcc33-1.patch`

Perl Libc Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/perl-5.8.0-libc-3.patch`

Procps Locale Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/procps-3.1.11-locale-fix.patch`

Shadow Newgrp Patch - 1 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/shadow-4.0.3-newgrp-fix.patch`

Zlib Vsnprintf Patch - 10 KB:
`http://www.linuxfromscratch.org/patches/lfs/5.0/zlib-1.1.4-vsnprintf.patch`

In addition to the above required patches, there exist a number of optional ones created by the LFS community. Most of these solve slight problems, or enable some functionality that's not enabled by default. Feel free to examine the patches database, located at `http://www.linuxfromscratch.org/patches/`, and pick any additional patches you wish to use.

Chapter 5 Constructing a temporary system

Introduction

In this chapter we will compile and install a minimal Linux system. This system will contain just enough tools to be able to start constructing the final LFS system in the next chapter.

The building of this minimal system is done in two steps: first we build a brand-new and host-independent toolchain (compiler, assembler, linker and libraries), and then use this to build all 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. Since the packages compiled here are merely temporary, we don't want them to pollute the soon-to-be LFS system.

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 this purpose a short summary of the content of each package is given before the actual installation instructions. For a short description of each program in a package, please refer to the corresponding section in Appendix A.

The build instructions assume that you are using the bash shell. There is also a general expectation that you have already unpacked the sources for a package and have performed a `cd` into the unpacked source directory before issuing the build commands.

Several of the packages are patched before compilation, but only when the patch is needed to circumvent a problem. Often the patch is needed in both this and the next chapter, but sometimes in only one of them. Therefore, don't worry when instructions for a downloaded patch seem to be missing.

During the installation of most packages you will see all kinds of compiler warnings scroll by on your screen. These are normal and can be safely ignored. They are just what they say they are: warnings — mostly about deprecated, but not invalid, use of the C or C++ syntax. It's just that C standards have changed rather often and some packages still use the older standard, which is not really a problem.

Unless told not to, you should normally delete the source and build directories after installing each package — for cleanness sake and to save space.

Before continuing, make sure the LFS environment variable is set up properly by executing the following:

```
echo $LFS
```

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

Toolchain technical notes

This section attempts to explain some of the rationale and technical details behind the overall build method. It's not essential that you understand everything here immediately. Most of it will make sense once you have performed an actual build. Feel free to refer back here at any time.

The overall goal of Chapter 5 is to provide a sane, temporary environment that we can chroot into, and from which we can produce a clean, trouble-free build of the target LFS system in Chapter 6. Along the way, we attempt to divorce ourselves from the host system as much as possible, and in so doing build a self-contained and self-hosted toolchain. It should be noted that the build process has been designed in such a way so as 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.

Before continuing, you really should be aware of the name of your working platform, often also referred to as the *target triplet*. For many folks the target triplet will be, for example: *i686-pc-linux-gnu*. A simple way to determine your 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.

You'll also need to be aware of the name of your platform's *dynamic linker*, often also referred to as the *dynamic loader*, not to be confused with the standard linker *ld* that is part of Binutils. The dynamic linker is provided by Glibc and has the job of finding and loading the shared libraries needed by a program, preparing the program to run and then running it. For most folks, the name of the dynamic linker will be *ld-linux.so.2*. On platforms that are less prevalent, the name might be *ld.so.1* and newer 64 bit platforms might even have something completely different. You should be able to determine the name of your platform's dynamic linker by looking in the `/lib` directory on your host system. A surefire way is to inspect a random binary from your 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:

- Similar in principle to cross compiling whereby tools installed into the same prefix work in cooperation and thus utilize a little GNU "magic".

- Careful manipulation of the standard linker's library search path to ensure programs are linked only against libraries we choose.
- Careful manipulation of gcc's *specs* file to tell the compiler which target dynamic linker will be used.

Binutils is installed first because both GCC and Glibc perform various feature tests on the assembler and linker during their respective runs of `./configure` 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 a whole distribution. Thankfully, a test suite failure will usually alert us before too much time is wasted.

Binutils installs its assembler and linker into two locations, `/tools/bin` and `/tools/$TARGET_TRIPLET/bin`. In reality, 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: `'ld --verbose | grep SEARCH'` will show you the current search paths and their order. You can see what files are actually linked by `ld` by compiling a dummy program and passing the *--verbose* switch. For example: `'gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded'` will show you all the files successfully opened during the link.

The next package installed is GCC and during its run of `./configure` you'll see, for example:

```
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. You can find out which standard linker gcc will use by running: `'gcc -print-prog-name=ld'`. Detailed information can be obtained from gcc by passing it the *-v* flag while compiling a dummy program. For example: `'gcc -v dummy.c'` will show you detailed information about the preprocessor, compilation and assembly stages, including gcc's include 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 no problem as Glibc will always use the gcc found in a $PATH directory. The binary tools and kernel headers can be a little more troublesome. Therefore we take no risks and use the available configure switches to enforce the correct selections. After the run of `./configure` you can check the contents of the `config.make` file in the `glibc-build` directory for all the important details. You'll note some interesting items like the use of `CC="gcc -B/tools/bin/"` to control which binary tools are used, and also the use of the *-nostdinc* and *-isystem* flags to control the compiler's include search path. These items help to 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, we make some adjustments to ensure that searching and linking take place only within our `/tools` prefix. We install an adjusted `ld`, which has a hard-wired search path limited to `/tools/lib`. Then we amend `gcc`'s specs file to point to our 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 ELF shared executable. You can inspect this by running: `'readelf -l <name of binary> | grep interpreter'`. By amending `gcc`'s specs file, we are ensuring that every program compiled from here through the end of Chapter 5 will use our new dynamic linker in `/tools/lib`.

The need to use the new dynamic linker is also the reason why we apply the Specs patch 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 our 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` and all is well.

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

Notes on static linking

Most programs have to perform, beside their specific task, many rather common and sometimes trivial operations. These include allocating memory, searching directories, reading and writing files, string handling, pattern matching, arithmetic and many 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, what is included is 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 way 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, you need only one copy of the executable library code on your hard disk, instead of having many copies of the same code included into a whole bunch of 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, you only need to recompile this one library, instead of having to recompile all the 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. Historical, because earlier versions of LFS statically linked every program in this chapter. Educational, because knowing the difference is useful. Technical, because we gain an element of independence from the host in doing so, meaning that those programs can be used independently of the host system. However, it's worth noting that an overall successful LFS build can still be achieved when the first two packages are built dynamically.

Creating the $LFS/tools directory

All programs compiled in this chapter will be installed under `$LFS/tools` to keep them separate from the programs compiled in the next chapter. The programs compiled here are only temporary tools and won't be a part of the final LFS system and by keeping them in a separate directory, we can later easily throw them away.

If later you wish to search through the binaries of your system to see what files they make use of or link against, then to make this searching easier you may want to choose a unique name. Instead of the simple "tools" you could use something like "tools-for-lfs".

Create the required directory by running the following:

```
mkdir $LFS/tools
```

The next step is to create a `/tools` symlink on your host system. It will point to the directory we just created on the LFS partition:

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

This symlink enables us to compile our toolchain 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).

Study the above command closely. It can be confusing at first glance. The `ln` command has several syntax variations, so be sure to check the ln man page before reporting what you may think is an error.

Adding the user lfs

When logged in as *root*, making a single mistake can damage or even wreck your system. Therefore we recommend that you build the packages in this chapter as an unprivileged user. You could of course use your own user name, but to make it easier to set up a clean work environment we'll create a new user *lfs* and use this one during the installation process. As *root*, issue the following commands to add the new user:

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

Now grant this new user *lfs* full access to `$LFS/tools` by giving it ownership of the directory:

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

If you made a separate working directory as suggested, give user *lfs* ownership of this directory too:

```
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 new, clean shell.

Setting up the environment

While logged in as user *lfs*, issue the following commands to set up a good work environment:

```
cat > ~/.bash_profile << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
PATH=/tools/bin:$PATH
export LFS LC_ALL PATH
unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD
EOF

source ~/.bash_profile
```

The `set +h` command turns off bash's hash function. Normally hashing is a useful feature: bash uses a hash table to remember the full pathnames of executable files to avoid searching the PATH time and time again to find the same executable. However, we'd like the new tools to be used as soon as they are installed. By switching off the hash function, our "interactive" commands (`make`, `patch`, sed, cp and so forth) will always use the newest available version during the build process.

Setting the user file-creation mask to 022 ensures that newly created files and directories are only writable for their owner, but readable and executable for anyone.

The LFS variable should of course be set to the mount point you chose.

The LC_ALL variable controls the localization of certain programs, making their messages follow the conventions of a specified country. If your 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 trouble if you exit the chroot environment and wish to return later. By setting LC_ALL to "POSIX" (or "C", the two are equivalent) we ensure that everything will work as expected in the chroot environment.

We prepend `/tools/bin` to the standard PATH so that, as we move along through this chapter, the tools we build will get used during the rest of the building process.

The CC, CXX, CPP, LD_LIBRARY_PATH and LD_PRELOAD environment variables all have the potential to cause havoc with our Chapter 5 toolchain. We therefore unset them to prevent any chance of this happening.

Now, after sourcing the just-created profile, we're all set to begin building the temporary tools that will support us in later chapters.

Installing Binutils-2.14 - Pass 1

```
Estimated build time:           1.0 SBU
Estimated required disk space:  194 MB
```

Contents of Binutils

Binutils is a collection of software development tools containing a linker, assembler and other tools to work with object files and archives.

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]

Binutils Installation Dependencies

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

Installation of Binutils

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

Even though Binutils is an important toolchain package, we are not going to run the test suite at this early stage. First, the test suite framework is not yet in place and second, the programs from this first pass will soon be overwritten by those installed in the second pass.

This package is known to behave badly when you have changed its default optimization flags (including the -march and -mcpu options). Therefore, if you have defined any environment variables that override default optimizations, such as CFLAGS and CXXFLAGS, we recommend unsetting or modifying 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
```

If you want the SBU values listed in the rest of the book to be of any use, you will have to measure the time it takes to build this package. To achieve this easily, you could do something like:

`time { ./configure ... && ... && ... && make install; }`.

Now prepare Binutils for compilation:

```
../binutils-2.14/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 (a word often shortened to i18n). We don't need this for our static programs and *nls* often causes 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 the subdirectories to be configured immediately. A statically linked build will fail without it. We therefore use this option to work around the problem.
- `LDFLAGS="-all-static"`: This tells the linker that all the Binutils programs should be linked statically. However, strictly speaking, `"-all-static"` is first passed to the *libtool* program which then passes `"-static"` on to the linker.

And install the package:

```
make install
```

Now prepare the linker for the "locking in" of Glibc 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 the compiled files, but only 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 have it point to our temporary tools location. The value of this variable specifies the linker's default library search path. You'll see how this preparation is used later on in the chapter.

Do not yet remove the Binutils build and source directories. You will need them again in their current state a bit further on in this chapter.

Installing GCC-3.3.1 - Pass 1

```
Estimated build time:           4.4 SBU
Estimated required disk space:  300 MB
```

Contents of GCC

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

Installed programs: c++, cc (link to gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug, and gcov

Installed libraries: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] and libsupc++.a

GCC Installation Dependencies

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

Installation of GCC

Unpack only the GCC-core tarball, as we won't be needing a C++ compiler for the moment.

Even though GCC is an important toolchain package, we are not going to run the test suite at this early stage. First, the test suite framework is not yet in place and second, the programs from this first pass will soon be overwritten by those installed in the second pass.

This package is known to behave badly when you have changed its default optimization flags (including the -march and -mcpu options). Therefore, if you have defined any environment variables that override default optimizations, such as CFLAGS and CXXFLAGS, we recommend unsetting or modifying 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-3.3.1/configure --prefix=/tools \
    --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, we want to try to minimize the influence of the host system, thus making this a sensible thing to do.
- `--enable-shared`: This switch may seem counter-intuitive at first. But using it 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, as this is controlled by the `-static` value of BOOT_LDFLAGS further on.
- `--enable-languages=c`: This option ensures that only the C compiler is built. The option is only needed when you have downloaded and unpacked the full GCC 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 doesn't 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, which most probably means that it was compiled correctly.

And install the package:

```
make install
```

As a finishing touch we'll create the `/tools/bin/cc` symlink. Many programs and scripts run `cc` instead of `gcc`, a thing meant to keep programs generic and therefore usable on all kinds of Unix systems. Not everybody has the GNU C compiler installed. Simply running `cc` leaves the system administrator free to decide what C compiler to install, as long as there's a symlink pointing to it:

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

Installing Linux-2.4.22 headers

```
Estimated build time:           0.1 SBU
Estimated required disk space:  186 MB
```

Contents of Linux

The Linux kernel is at the core of every Linux system. It's what makes Linux tick. When a computer is turned on and boots a Linux system, the very first piece of Linux software that gets loaded is the kernel. The kernel initializes the system's hardware components: serial ports, parallel ports, sound cards, network cards, IDE controllers, SCSI controllers and a lot more. In a nutshell the kernel makes the hardware available so that the software can run.

Installed files: the kernel and the kernel headers

Linux Installation Dependencies

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

Installation of the kernel headers

As some packages need to refer to the kernel header files, we're going to unpack the kernel archive now, set it up, and copy the required files to a place where gcc can later find them.

Prepare for the header installation with:

```
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. You shouldn't rely on the source tree being clean after untarring.

Create the include/linux/version.h file:

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

Create the platform-specific include/asm symlink:

```
make symlinks
```

Install the platform-specific header files:

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

Install the cross-platform kernel header files:

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

There are a few kernel header files which make use of the `autoconf.h` header file. Since we do not yet configure the kernel, we need to create this file ourselves in order to avoid compilation failures. Create an empty `autoconf.h` file:

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

Installing Glibc-2.3.2

```
Estimated build time:           11.8 SBU
Estimated required disk space:  800 MB
```

Contents of Glibc

Glibc is the C library that provides the system calls and basic functions such as open, malloc, printf, etc. The C library is used by all dynamically linked programs.

Installed programs: catchsegv, gencat, getconf, getent, glibcbug, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nscd_nischeck, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump and zic

Installed libraries: ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd-compat.a, libc.[a,so], libc_nonshared.a, libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so and libutil.[a,so]

Glibc Installation Dependencies

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

Glibc installation

Before starting to install Glibc, you must `cd` into the `glibc-2.3.2` directory and unpack Glibc-linuxthreads in that directory, not in the directory where you usually unpack all the sources.

We are going to run the test suite for Glibc in this chapter. However, it's worth pointing out that running the Glibc test suite here is considered not as important as running it in Chapter 6.

This package is known to behave badly when you have changed its default optimization flags (including the -march and -mcpu options). Therefore, if you have defined any environment variables that override default optimizations, such as CFLAGS and CXXFLAGS, we recommend unsetting them when building Glibc.

Basically, compiling Glibc in any other way than the book suggests is putting the stability of your system at risk.

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

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

Also, Glibc has a subtle problem when compiled with GCC 3.3.1. Apply the following patch to fix this:

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

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

The meaning of the configure options:

- `--disable-profile`: This disables the building of the libraries with profiling information. Omit this option if you plan to do profiling.
- `--enable-add-ons`: This enables any add-ons that were installed with Glibc, in our case Linuxthreads.
- `--with-binutils=/tools/bin` and `--with-headers=/tools/include`: Strictly speaking these switches are not required. But they ensure nothing can go wrong with regard to what kernel headers and Binutils programs get used during the Glibc build.
- `--without-gd`: This switch ensures that we don't build the `memusagestat` program, which strangely enough insists on linking against the host's libraries (libgd, libpng, libz, and so forth).

During this stage you might see the following warning:

```
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's believed it can sometimes cause problems when running the test suite.

Compile the package:

```
make
```

Run the test suite:

```
make check
```

The Glibc test suite is highly dependent on certain functions of your host system, in particular the kernel. Additionally, here in this chapter some tests can be adversely affected by existing tools or environmental issues on the host system. Of course, these won't be a problem when we run the Glibc test suite inside the chroot environment of Chapter 6. In general, the Glibc test suite is always expected to pass. However, as mentioned above, some failures are unavoidable in certain circumstances. Here is a list of the most common issues we are aware of:

- 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, or due to other file system quirks.
- The *shm* test might fail when the host system is running the devfs file system but doesn't have the tmpfs file system mounted at `/dev/shm` due to lack of support for tmpfs in the kernel.
- When running on older and slower hardware, some tests might fail due to test timeouts being exceeded.

In summary, don't worry too much if you see Glibc test suite failures here in this chapter. The Glibc in Chapter 6 is the one we'll ultimately end up using so that is the one we would really like to see pass. But please keep in mind, even in Chapter 6 some failures could still occur — the *math* tests for example. When experiencing a failure, make a note of it, then continue by reissuing the `make check`. The test suite should pick up where it left off and continue on. You can circumvent this stop-start sequence by issuing a `make -k check`. But if you do that, be sure to log the output so that you can later peruse the log file and examine the total number of failures.

Now install the package:

```
make install
```

Different countries and cultures have varying conventions for how to communicate. These conventions range from very simple ones, such as the format for representing dates and times, to very complex ones, such as the language spoken. The "internationalization" of GNU programs works by means of *locales*. We'll install the Glibc locales now:

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

An alternative to running the previous command is to install only those locales which you need or want. This can be achieved by using the `localedef` command. Information on this can be found in the `INSTALL` file in the `glibc-2.3.2` source. However, there are a number of locales that are essential for the tests of future packages to pass, in particular, the *libstdc++* tests from GCC. The following instructions, instead of the install-locales target 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 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
```

"Locking in" Glibc

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

First install the adjusted linker by running the following from within the `binutils-build` directory:

```
make -C ld install
```

The linker was adjusted a little while back, at the end of the first pass of Binutils. From this point onwards everything will link *only* against the libraries in `/tools/lib`.

If you somehow missed the earlier warning to retain the Binutils source and build directories from the first pass or otherwise accidentally deleted them or just don't have access to them, don't worry, all is not lost. Just ignore the above command. The result is a small chance of subsequent programs linking against libraries on the host. This is not ideal, however, it's not a major problem. The situation is corrected when we install the second pass of Binutils later on.

Now that the adjusted linker is installed, you have to remove the Binutils build and source directories.

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

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

We recommend that you cut-and-paste the above rather than try and type it all in. Or you can edit the specs file by hand if you want to: just replace any occurrence of "/lib/ld-linux.so.2" with "/tools/lib/ld-linux.so.2".

If you are working on a platform where the name of the dynamic linker is something other than `ld-linux.so.2`, you *must* substitute `ld-linux.so.2` with the name of your platform's dynamic linker in the above commands. Refer back to the Section called *Toolchain technical notes* if necessary.

Lastly, 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 because of GCC's "fixincludes" process which runs as part of the GCC build. We'll explain more about this further on in this chapter. For now, run the following commands to eliminate this possibility:

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

It is imperative at this point to stop and ensure that the basic functions (compiling and linking) of the new toolchain are working as expected. For this we are going to perform a simple sanity check:

```
echo 'main(){}' > dummy.c
gcc 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:

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

If you did not receive the output as shown above, or received no output at all, then something is seriously wrong. You will need to investigate and retrace your steps to find out where the problem is and correct it. There is no point in continuing until this is done. Most likely something went wrong with the specs file amendment above. Note especially that `/tools/lib` appears as the prefix of our dynamic linker. Of course, if you are working on a platform where the name of the dynamic linker is something other than `ld-linux.so.2`, then the output will be slightly different.

Once you are satisfied that all is well, clean up the test files:

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

This completes the installation of the self-contained toolchain, and it can now be used to build the rest of the temporary tools.

Installing Tcl-8.4.4

```
Estimated build time:           0.9 SBU
Estimated required disk space:  23 MB
```

Contents of Tcl

The Tcl package contains the Tool Command Language.

Installed programs: tclsh (link to tclsh8.4), tclsh8.4

Installed library: libtcl8.4.so

Tcl Installation Dependencies

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

Installation of Tcl

This package and the next two are only installed to be able to run the test suites for GCC and Binutils. Installing three packages just for testing purposes may seem like overkill, but it is very reassuring, if not essential, to know that our most important tools are working properly.

Prepare Tcl for compilation:

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

Build the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. However, the Tcl test suite in this chapter is known to experience failures under certain host conditions that are not fully understood. Therefore, test suite failures here are not surprising, but are not considered critical. Should you choose to run the test suite, the following command will do so:

```
TZ=UTC make test
```

The meaning of the make parameter:

- `TZ=UTC`: This 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 the clock tests are exercised correctly. More information on the TZ environment variable is available later on in Chapter 7.

Sometimes, package test suites will give false failures. You can consult the LFS Wiki at `http://wiki.linuxfromscratch.org/` to verify that these failures are normal. This applies to all tests throughout the book.

Install the package:

```
make install
```

Do not remove the `tcl8.4.4` source directory yet, as the next package will need its internal headers.

Make a necessary symbolic link:

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

Installing Expect-5.39.0

```
Estimated build time:           0.1 SBU
Estimated required disk space:  3.9 MB
```

Contents of Expect

The Expect package provides a program that performs programmed dialogue with other interactive programs.

Installed program: Expect 228

Installed library: libexpect5.39.a

Expect Installation Dependencies

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

Installation of Expect

First apply a patch:

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

This fixes a bug in Expect that can result in bogus failures during the GCC test suite run.

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 our temporary tools location. We don't want it to find an existing one that may possibly reside 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 possibly reside on the host system.

Build the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. However, the Expect test suite here in Chapter 5 is known to experience failures under certain host conditions that are not fully understood. Therefore, test suite failures here are not surprising, but are not considered critical. Should you choose to run the test suite, the following command will do so:

```
make test
```

And install:

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

The meaning of the make parameter:

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

You can now remove the source directories of both Tcl and Expect.

Installing DejaGnu-1.4.3

```
Estimated build time:            0.1 SBU
Estimated required disk space:   8.6 MB
```

Contents of DejaGnu

The DejaGnu package contains a framework for testing other programs.

Installed program: runtest

DejaGnu Installation Dependencies

Dejagnu depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation of DejaGnu

Prepare DejaGnu for compilation:

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

Build and install the package:

```
make install
```

Installing GCC-3.3.1 - Pass 2

```
Estimated build time:           11.0 SBU
Estimated required disk space:  274 MB
```

Re-installation of GCC

The tools required to test GCC and Binutils are installed now (Tcl, Expect and DejaGnu). We can continue on rebuilding GCC and Binutils, link them against the new Glibc, and test them properly. One thing to note, however, is that these test suites are highly dependent on properly functioning pseudo terminals (PTYs) which are provided by your host distribution. These days, PTYs are most commonly implemented via the *devpts* file system. You can quickly check if your host system is set up correctly in this regard by performing a simple test:

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

If you receive the message:

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

Your host distribution is not set up for proper PTY operation. In this case there is no point in running the test suites for GCC and Binutils until you are able to resolve the issue. You can consult the LFS Wiki at `http://wiki.linuxfromscratch.org/` for more information on how to get PTYs working.

Unpack all three GCC tarballs (-core, -g++, and -testsuite) in one and the same working directory. They will all unfold into a single `gcc-3.3.1/` subdirectory.

First correct one problem and make an essential adjustment:

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

The first patch disables the GCC "fixincludes" script. We mentioned this briefly earlier, but a slightly more in-depth explanation of the fixincludes process is warranted here. Under normal circumstances, the GCC fixincludes script scans your system for header files that need to be fixed. It might find that some Glibc header files on your host system need to be fixed, fix them and put them in the GCC private include directory. Then, later on in Chapter 6, after we've installed the newer Glibc, this private include directory would be searched before the system include directory, resulting in GCC finding the fixed headers from the host system, which would most likely not match the Glibc version actually used for the LFS system.

The last 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 our new dynamic linker gets used during the actual build of GCC. That is, all the final (and temporary) binaries created during the build will link against the new Glibc.

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

The meaning of the new configure options:

- `--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 and is essential for fully standards-compliant handling of destructors. It also affects the C++ ABI and therefore results in C++ shared libraries and C++ programs that are interoperable with other Linux distributions.
- `--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 model of *gnu*. However, people who don't install the *de_DE* locale, run the risk of building ABI incompatible C++ libraries due to the wrong locale model of *generic* being selected.
- `--enable-languages=c,c++`: This option is needed to ensure that both C and C++ compilers are built.

Compile the package:

```
make
```

There is no need to use the `bootstrap` target now, as the compiler we're using to compile this GCC was built from the exact same version of the GCC sources we used earlier.

It's worth pointing out that running the GCC test suite here is considered not as important as running it in Chapter 6.

Test the results:

```
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 get a summary of the test suite results, run this:

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

You can compare your results to those posted to the gcc-testresults mailing list for similar configurations to your own. For an example of how current GCC-3.3.1 should look on i686-pc-linux-gnu, see `http://gcc.gnu.org/ml/gcc-testresults/2003-08/msg01612.html`.

Note that the results contain:

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

The unexpected pass for g++ is due to the use of `--enable-__cxa_atexit`. Apparently not all platforms supported by GCC have support for "__cxa_atexit" in their C libraries, so this test is not always expected to pass.

The 26 unexpected passes for libstdc++ are due to the use of `--enable-clocale=gnu`, which is the correct choice on Glibc-based systems of versions 2.2.5 and above. The underlying locale support in the GNU C library is superior to that of the otherwise selected "generic" model (which may be applicable if for instance you were using Newlibc, Sun-libc or whatever libc). The libstdc++ test suite is apparently expecting the "generic" model, hence those tests are not always expected to pass.

Unexpected failures often cannot be avoided. The GCC developers are usually aware of them but haven't yet gotten around to fixing them. In short, unless your results are vastly different from those at the above URL, it is safe to continue on.

And finally install the package:

```
make install
```

At this point it is strongly recommended to repeat the sanity check we performed earlier in the chapter. Refer back to the Section called *"Locking in" Glibc* and repeat the check. If the results are wrong, then most likely you forgot to apply the above mentioned GCC Specs patch.

Installing Binutils-2.14 - Pass 2

```
Estimated build time:           1.5 SBU
Estimated required disk space:  108 MB
```

Re-installation of Binutils

Create a separate build directory again:

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

Now prepare Binutils for compilation:

```
../binutils-2.14/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 default library search path. We don't want the library search path to contain library directories from the host system.

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

Compile the package:

```
make
```

It's worth pointing out that running the Binutils test suite here is considered not as important as running it in Chapter 6.

Test the results (there should be no unexpected failures here, expected failures are fine):

```
make check
```

Unfortunately, there is no easy way to view the test results summary like there was for the previous GCC package. However, if a failure occurs here, it should be easy to spot. The output shown will contain something like:

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

And install the package:

```
make install
```

Now prepare Binutils for the re-adjusting of the toolchain in the next chapter:

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

Do not yet remove the Binutils source and build directories. We'll need these directories again in the next chapter in the state they are in now.

Installing Gawk-3.1.3

```
Estimated build time:           0.2 SBU
Estimated required disk space:  17 MB
```

Contents of Gawk

Gawk is an awk implementation that is used to manipulate text files.

Installed programs: awk (link to gawk), gawk, gawk-3.1.3, grcat, igawk, pgawk, pgawk-3.1.3 and pwcat

Gawk Installation Dependencies

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

Installation of Gawk

Prepare Gawk for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install it:

```
make install
```

Installing Coreutils-5.0

```
Estimated build time:           0.9 SBU
Estimated required disk space:  69 MB
```

Contents of Coreutils

The Coreutils package contains a whole series of basic shell utilities.

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, kill, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami and yes

Coreutils Installation Dependencies

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

Installation of Coreutils

Prepare Coreutils for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make RUN_EXPENSIVE_TESTS=yes check
```

The meaning of the make parameter:

- `RUN_EXPENSIVE_TESTS=yes`: This tells the test suite to run several additional tests that are considered relatively expensive on some platforms. However, they are generally not a problem on Linux.

And install the package:

```
make install
```

Installing Bzip2-1.0.2

```
Estimated build time:           0.1 SBU
Estimated required disk space:  2.5 MB
```

Contents of Bzip2

Bzip2 is a block-sorting file compressor which generally achieves a better compression than the traditional `gzip` does.

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

Bzip2 Installation Dependencies

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

Installation of Bzip2

The Bzip2 package doesn't contain a `configure` script. Compile and install it with a straightforward:

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

Installing Gzip-1.3.5

```
Estimated build time:           0.1 SBU
Estimated required disk space:  2.6 MB
```

Contents of Gzip

The Gzip package contains programs to compress and decompress files using the Lempel-Ziv coding (LZ77).

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

Gzip Installation Dependencies

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

Installation of Gzip

Prepare Gzip for compilation:

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

Compile the package:

```
make
```

And install it:

```
make install
```

Installing Diffutils-2.8.1

```
Estimated build time:           0.1 SBU
Estimated required disk space:  7.5 MB
```

Contents of Diffutils

The programs from this package show you the differences between two files or directories. It's most common use is to create software patches.

Installed programs: cmp, diff, diff3 and sdiff

Diffutils Installation Dependencies

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

Installation of Diffutils

Prepare Diffutils for compilation:

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

Compile the package:

```
make
```

And install it:

```
make install
```

Installing Findutils-4.1.20

```
Estimated build time:           0.2 SBU
Estimated required disk space:  7.6 MB
```

Contents of Findutils

The Findutils package contains programs to find files, either on-the-fly (by doing a live recursive search through directories and only showing files that match the specifications) or by searching through a database.

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

Findutils Installation Dependencies

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

Installing Findutils

Prepare Findutils for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Make-3.80

```
Estimated build time:           0.2 SBU
Estimated required disk space:  8.8 MB
```

Contents of Make

Make determines, automatically, which pieces of a large program need to be recompiled and issues the commands to recompile them.

Installed program: Make   249

Make Installation Dependencies

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

Installation of Make

Prepare Make for compilation:

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

Compile the program:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Then install it and its documentation:

```
make install
```

Installing Grep-2.5.1

```
Estimated build time:           0.1 SBU
Estimated required disk space:  5.8 MB
```

Contents of Grep

Grep is a program used to print lines from a file matching a specified pattern.

Installed programs: egrep (link to grep), fgrep (link to grep) and grep

Grep Installation Dependencies

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

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 `grep` does not get linked against a PCRE library that may be present on the host, but would not be available once we enter the chroot environment.
- `--with-included-regex`: This ensures that Grep uses its internal regular expression code. Without it, it will use the code from Glibc, which is known to be slightly buggy.

Compile the programs:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Then install them and their documentation:

```
make install
```

Installing Sed-4.0.7

```
Estimated build time:           0.2 SBU
Estimated required disk space:  5.2 MB
```

Contents of Sed

sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline).

Installed program: Sed 260

Sed Installation Dependencies

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

Installation of Sed

Prepare Sed for compilation:

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

Compile the program:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Then install it and its documentation:

```
make install
```

Installing Gettext-0.12.1

```
Estimated build time:            7.2 SBU
Estimated required disk space:  55 MB
```

Contents of Gettext

The Gettext package is used for internationalization and localization. Programs can be compiled with Native Language Support (NLS) which enable them to output messages in the user's native language.

Installed programs: autopoint, config.charset, config.rpath, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, project-id, team-address, trigger, urlget, user-email and xgettext

Installed libraries: libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] and libgettextsrc[a,so]

Gettext Installation Dependencies

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

Installation of Gettext

Prepare Gettext for compilation:

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

Compile the programs:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. However, the Gettext test suite here in Chapter 5 is known to experience failures under certain host conditions — for example, if it finds a Java compiler on the host. The Gettext test suite takes a very long time to run and is not considered critical. Therefore, we don't recommend running it here. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Ncurses-5.3

```
Estimated build time:           0.7 SBU
Estimated required disk space:  26 MB
```

Contents of Ncurses

The Ncurses package provides character and terminal handling libraries, including panels and menus.

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], libpanel.[a,so]

Ncurses Installation Dependencies

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

Installation of Ncurses

Fix two minor things:

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

The first patch corrects the `etip.h` header file, and the second patch prevents some compiler warnings being issued on the use of deprecated headers.

Now 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 must 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 programs and libraries:

```
make
```

Then install them and their documentation:

```
make install
```

Installing Patch-2.5.4

```
Estimated build time:           0.1 SBU
Estimated required disk space:  1.9 MB
```

Contents of Patch

The patch program modifies a file according to a patch file. A patch file usually is a list, created by the diff program, that contains instructions on how an original file needs to be modified.

Installed program: Patch   255

Patch Installation Dependencies

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

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. On other architectures you can leave it out.

Compile the program:

```
make
```

Then install it and its documentation:

```
make install
```

Installing Tar-1.13.25

```
Estimated build time:           0.2 SBU
Estimated required disk space:  10 MB
```

Contents of Tar

Tar is an archiving program designed to store and extract files from an archive file known as a tar file.

Installed programs: rmt and tar

Tar Installation Dependencies

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

Installation of Tar

Prepare Tar for compilation:

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

Compile the programs:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Then install them and their documentation:

```
make install
```

Installing Texinfo-4.6

```
Estimated build time:           0.2 SBU
Estimated required disk space:  16 MB
```

Contents of Texinfo

The Texinfo package contains programs used for reading, writing and converting Info documents, which provide system documentation.

Installed programs: info, infokey, install-info, makeinfo, texi2dvi and texindex

Texinfo Installation Dependencies

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

Installation of Texinfo

Prepare Texinfo for compilation:

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

Compile the programs:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Then install them and their documentation:

```
make install
```

Installing Bash-2.05b

```
Estimated build time:           1.2 SBU
Estimated required disk space:  27 MB
```

Contents of Bash

bash is the Bourne-Again SHell, which is a widely used command interpreter on Unix systems. The bash program reads from standard input (the keyboard). A user types something and the program will evaluate what he has typed and do something with it, like running a program.

Installed programs: bash, sh (link to bash) and bashbug

Bash Installation Dependencies

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

Installation of Bash

Bash contains several known bugs. Fix these with the following patch:

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

Now prepare Bash for compilation:

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

Compile the program:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make tests
```

Then install it and its documentation:

```
make install
```

And make a link for the programs that use sh for a shell:

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

Installing Util-linux-2.12

```
Estimated build time:           0.1 SBU
Estimated required disk space:  8 MB
```

Contents of Util-linux

The Util-linux package contains a number of miscellaneous utility programs. Some of the more prominent utilities are used to mount, unmount, format, partition and manage disk drives, open tty ports and fetch kernel messages.

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, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, pg, pivot_root, ramsize (link to rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (link to rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (link to swapon), swapon, test.bash, test.tcsh, tunelp, ul, umount, vidmode (link to rdev), whereis and write

Util-linux Installation Dependencies

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

Installation of Util-linux

Util-linux doesn't use the freshly installed headers and libraries from the /tools directory. This is fixed by altering the configure script:

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

Prepare Util-linux for compilation:

```
./configure
```

Compile some support routines:

```
make -C lib
```

And, since you'll need only a couple of the utilities contained in this package, build just those:

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

Now copy these programs to the temporary tools directory:

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

Installing Perl-5.8.0

```
Estimated build time:          0.8 SBU
Estimated required disk space:  74 MB
```

Contents of Perl

The Perl package contains perl, the Practical Extraction and Report Language. Perl combines some of the best features of C, sed, awk and sh into one powerful language.

Installed programs: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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: (too many to name)

Perl Installation Dependencies

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

Installation of Perl

First adapt some hard-wired paths to the C library:

```
patch -Np1 -i ../perl-5.8.0-libc-3.patch
```

And make sure some static extensions get built:

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

Now prepare Perl for compilation:

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

Compile only the required tools:

```
make perl utilities
```

Then copy these tools and their libraries:

```
cp perl pod/pod2man /tools/bin
mkdir -p /tools/lib/perl5/5.8.0
cp -R lib/* /tools/lib/perl5/5.8.0
```

Stripping

The steps in this section are optional. If your LFS partition is rather small, you will be glad to learn that you can throw away some unnecessary things. The executables and libraries you have built so far contain about 130 MB of unneeded debugging symbols. Remove those symbols like this:

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

The first of the above commands will skip some twenty files, reporting that it doesn't recognize their file format. Most of them are scripts instead of binaries.

Take care *not* to use `--strip-unneeded` on the libraries — they would be destroyed and you would have to build Glibc all over again.

To save another couple of megabytes, you can throw away all the documentation:

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

You will now need to have at least 850 MB of free space on your LFS filesystem to be able 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

Introduction

In this chapter we enter the building site, and start constructing our LFS system in earnest. That is, we chroot into our temporary mini Linux system, create some auxiliary things, and then start installing all the packages, one by one.

The installation of all this software is pretty straightforward, and you will probably think it would be much shorter to give here the generic installation instructions and explain in full only the installation of those packages that require an alternate method. Although we agree with that, we nevertheless choose to give the full instructions for each and every package, simply to minimize the possibilities for mistakes.

If you plan to use compiler optimizations in this chapter, take a look at 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 even problems when running the program. If a package refuses to compile when using optimization, try to compile it without optimization and see if the problem goes away. Even if the package does compile when using optimization, there is the risk it may have been compiled incorrectly due to complex interactions between the code and build tools. In short, the small potential gains achieved in using compiler optimization are generally outweighed by the risk. First time builders of LFS are encouraged to build without custom optimizations. Your system will still be very fast and very stable at the same time.

The order in which packages are installed in this chapter has to be strictly followed, to ensure that no program gets a path referring to `/tools` hard-wired into it. For the same reason, *do not* compile packages in parallel. Compiling in parallel may save you some 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.

About debugging symbols

Most programs and libraries are, by default, compiled with debugging symbols included (with gcc option -g).

When debugging a program or library that was compiled with debugging information included, the debugger can give you not only memory addresses but also the names of the routines and variables.

But the inclusion of these debugging symbols enlarges a program or library significantly. To get an idea of the amount of space these symbols occupy, have a look at the following:

- 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 a little, depending on which compiler was used and which C library. But when comparing programs with and without debugging symbols, the difference will generally be a factor between 2 and 5.

As most people will probably never use a debugger on their system software, a lot of disk space can be regained by removing these symbols .

To remove debugging symbols from a binary (which must be an a.out or ELF binary), run `strip --strip-debug filename`. Wildcards can be used to treat multiple files (use something like `strip --strip-debug $LFS/tools/bin/*`).

For your convenience, Chapter 9 includes one simple command to strip all debugging symbols from all programs and libraries on your system. Additional information on optimization can be found in the hint at `http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt`.

Entering the chroot environment

It is time to enter the chroot environment in order to begin installing the packages we need. Before you can chroot, however, you need to become *root*, since only *root* can execute the `chroot` command.

Just like earlier, ensure the LFS environment variable is set up properly by running `echo $LFS` and ensuring it shows the path to your LFS partition's mount point, which is `/mnt/lfs` if you followed our example.

Become *root* and run the following command to enter the chroot environment:

```
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
```

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 you need other variables present, such as CFLAGS or CXXFLAGS, this is a good place to set them again.

From this point on there's no need to use the LFS variable anymore, because everything you do will be restricted to the LFS file system — since what the shell thinks is `/` is actually the value of `$LFS`, which was passed to the chroot command.

Notice that /tools/bin comes last in the PATH. This means that a temporary tool will not be used any more as soon as its final version is installed. Well, at least when the shell doesn't remember the locations of executed binaries — for this reason hashing is switched off a bit further on.

You have to make sure all the commands in the rest of this chapter and in the following chapters are run from within the chroot environment. If you ever leave this environment for any reason (rebooting for example), you must remember to again enter chroot and mount the proc and devpts filesystems (discussed later) before continuing with the installations.

Note that the bash prompt will say "I have no name!" This is normal, as the /etc/passwd file has not been created yet.

Changing ownership

Right now the /tools directory is owned by the user *lfs*, a user that exists only on your host system. Although you will probably want to delete the /tools directory once you have finished your LFS system, you may want to keep it around, for example to build more LFS systems. But if you keep the /tools directory as it is, you end up with files owned by a user ID without a corresponding account. This is dangerous because a user account created later on could get this same user ID and would suddenly own the /tools directory and all the files therein, thus exposing these files to possible malicious manipulation.

To avoid this issue, you could add the *lfs* user to your new LFS system later on when creating the /etc/passwd file, taking care to assign it the same user and group IDs as on your host system. Alternatively, you can (and the book assumes you do) 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.

Creating directories

Let's now create some structure in our LFS file system. Let's create a directory tree. Issuing the following commands will create a more or less standard tree:

```
mkdir -p /{bin,boot,dev/{pts,shm},etc/opt,home,lib,mnt,proc}
mkdir -p /{root,sbin,tmp,usr/local,var,opt}
for dirname in /usr /usr/local
    do
    mkdir $dirname/{bin,etc,include,lib,sbin,share,src}
    ln -s share/{man,doc,info} $dirname
    mkdir $dirname/share/{dict,doc,info,locale,man}
    mkdir $dirname/share/{nls,misc,terminfo,zoneinfo}
    mkdir $dirname/share/man/man{1,2,3,4,5,6,7,8}
```

```
done
mkdir /var/{lock,log,mail,run,spool}
mkdir -p /var/{tmp,opt,cache,lib/misc,local}
mkdir /opt/{bin,doc,include,info}
mkdir -p /opt/{lib,man/man{1,2,3,4,5,6,7,8}}
```

Directories are, by default, created with permission mode 755, but this isn't desirable for all directories. We will make two changes: one to the home directory of *root*, and another to the directories for temporary files.

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

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 in the 1777 bit mask.

FHS compliance note

We have based our directory tree on the FHS standard (available at `http://www.pathname.com/fhs/`). Besides the above created tree this standard stipulates the existence of `/usr/local/games` and `/usr/share/games`, but we don't much like these for a base system. However, feel free to make your system FHS-compliant. As to the structure of the `/usr/local/share` subdirectory, the FHS isn't precise, so we created here the directories that we think are needed.

Mounting the proc and devpts file systems

In order for certain programs to function properly, the *proc* and *devpts* file systems must be available within the chroot environment. A file system can be mounted as many times and in as many places as you like, thus it's not a problem that these file systems are already mounted on your host system — especially so because they are virtual file systems.

The *proc* file system is the process information pseudo-filesystem that the kernel uses to provide status information about the status of the system.

The proc file system is mounted on `/proc` by running the following command:

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

You might get warning messages from the mount command, such as these:

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

Ignore these, they're just due to the fact that the system isn't installed completely yet and some files are missing. The mount itself will be successful and that's all we care about at this point.

The *devpts* file system was mentioned earlier and is now the most common way for pseudo terminals (PTYs) to be implemented.

The devpts file system is mounted on `/dev/pts` by running:

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

Should this command fail with an error to the effect of:

```
filesystem devpts not supported by kernel
```

The most likely cause is that your host system's kernel was compiled without support for the devpts file system. You can check which file systems your kernel supports by peeking into its internals with a command such as `cat /proc/filesystems`. If a file system type named *devfs* is listed there, then we'll be able to work around the problem by mounting the host's devfs file system on top of the new `/dev` structure which we'll create later on in the "Creating devices (Makedev)" section. If devfs was not listed, do not worry because there is yet a third way to get PTYs working inside the chroot environment. We'll cover this shortly in the aforementioned Makedev section.

Remember, if for any reason you stop working on your LFS, and start again later, it's important to check that these filesystems are still mounted inside the chroot environment, otherwise problems are likely to occur.

Creating essential symlinks

Some programs hard-wire paths to programs which don't exist yet. In order to satisfy these programs, we create a number of symbolic links which will be replaced by real files throughout the course of this chapter when we're installing all the software.

```
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
```

Creating the passwd and group files

In order for *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" 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:
EOF
```

The created groups aren't part of any standard — they are the groups that the MAKEDEV script in the next section uses. Besides the group "root", the LSB (http://www.linuxbase.org) recommends only a group "bin", with a GID of 1, be present. All other group names and GIDs can be chosen freely by the user, as well-written packages don't depend on GID numbers but use the group's name.

Lastly, we re-login to the chroot environment. User name and group name resolution will start working immediately after the /etc/passwd and /etc/group files are created, because we installed a full Glibc in Chapter 5. This will get rid of the "I have no name!" prompt.

```
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. Since we want to use our newly compiled binaries as soon as they are installed, we turn off this function for the duration of this chapter.

Creating devices (Makedev-1.7)

```
Estimated build time:           0.1 SBU
Estimated required disk space:  50 KB
```

Contents of MAKEDEV

The MAKEDEV script creates the static device nodes which usually reside in the `/dev` directory. Detailed information about device nodes may be found in the `Documentation/devices.txt` file under the Linux kernel source tree.

Installed script: MAKEDEV   250

MAKEDEV Installation Dependencies

Make depends on: Bash, Coreutils.

Creating devices

Note that unpacking the `MAKEDEV-1.7.bz2` file doesn't create a directory for you to `cd` into, as the file contains only a shell script.

Install the `MAKEDEV` script:

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

Run the script to create the device files:

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

The meaning of the arguments:

- `-v`: This tells the script to run in verbose mode.
- `generic-nopty`: This instructs `MAKEDEV` to create a generic selection of commonly used device special files, except for the ptyXX and ttyXX range of files. We don't need those files because we are going to use Unix98 PTYs via the *devpts* file system.

If it turns out that some special device `zzz` that you need is missing, try running `./MAKEDEV -v zzz`. Alternatively, you may create devices via the `mknod` program. Please refer to its man and info pages if you need more information.

Additionally, if you were unable to mount the devpts filesystem earlier in the "Mounting the proc and devpts file systems" section, now is the time to try the alternatives. If your kernel supports the devfs file system, run the following command to mount devfs:

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

This will mount the devfs file system over the top of the new static `/dev` structure. This poses no problems, as the device nodes created are still present, they are just hidden by the new devfs filesystem.

If this still doesn't work, the only option left is to use the MAKEDEV script to create the ptyXX and ttyXX range of files that would otherwise not be needed. Ensure you are still in the `/dev` directory then run `./MAKEDEV -v pty`. The downside of this is, we are creating an extra 512 device special files which will not be needed when we finally boot into the finished LFS system.

Installing Linux-2.4.22 headers

```
Estimated build time:          0.1 SBU
Estimated required disk space:  186 MB
```

Contents of Linux

The Linux kernel is at the core of every Linux system. It's what makes Linux tick. When a computer is turned on and boots a Linux system, the very first piece of Linux software that gets loaded is the kernel. The kernel initializes the system's hardware components: serial ports, parallel ports, sound cards, network cards, IDE controllers, SCSI controllers and a lot more. In a nutshell the kernel makes the hardware available so that the software can run.

Installed files: the kernel and the kernel headers

Linux Installation Dependencies

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

Installation of the kernel headers

We won't be compiling a new kernel yet — we'll do that when we have finished the installation of all the packages. But as some packages need the kernel header files, we're going to unpack the kernel archive now, set it up and copy the header files so they can be found by these packages.

It is important to note that the files in the kernel source directory are not owned by *root*. Whenever you unpack a package as user *root* (like we do here inside chroot), the files end up having the user and group IDs of whatever they were on the packager's computer. This is usually not a problem for any other package you install because you remove the source tree after the installation. But the Linux kernel source tree is often kept around for a long time, so there's a chance that whatever user ID the packager used will be assigned to somebody on your machine and then that person would have write access to the kernel source.

In light of this, you might want to run `chown -R 0:0` on the `linux-2.4.22` directory to ensure all files are owned by user *root*.

Prepare for header installation:

```
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. You shouldn't rely on the source tree being clean after untarring.

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

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

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

```
make symlinks
```

Install the platform specific-header files:

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

Install the cross-platform kernel header files:

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

There are a few kernel header files which make use of the `autoconf.h` header file. Since we do not yet configure the kernel, we need to create this file ourselves in order to avoid compilation failures. Create an empty `autoconf.h` file:

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

Why we copy the kernel headers and don't symlink them

In the past it was common practice to symlink the `/usr/include/{linux,asm}` directories to `/usr/src/linux/include/{linux,asm}`. This was a *bad* practice, as the following extract from a post by Linus Torvalds to the Linux Kernel Mailing List points out:

```
I would suggest that people who compile new kernels should:

 - not have a single symbolic link in sight (except the one that the
   kernel build itself sets up, namely the "linux/include/asm" symlink
   that is only used for the internal kernel compile itself)

And yes, this is what I do. My /usr/src/linux still has the old 2.2.13
header files, even though I haven't run a 2.2.13 kernel in a _loong_
time. But those headers were what Glibc was compiled against, so those
headers are what matches the library object files.

And this is actually what has been the suggested environment for at
least the last five years. I don't know why the symlink business keeps
on living on, like a bad zombie. Pretty much every distribution still
has that broken symlink, and people still remember that the linux
sources should go into "/usr/src/linux" even though that hasn't been
true in a _loong_ time.
```

The essential part is where Linus states that the header files should be *the ones which Glibc was compiled against*. These are the headers that should be used when you later compile other packages, as they are the ones that match the object-code library files. By copying the headers, we ensure that they remain available if later you upgrade your kernel.

Note, by the way, that it is perfectly all right to have the kernel sources in `/usr/src/linux`, as long as you don't have the `/usr/include/{linux,asm}` symlinks.

Installing Man-pages-1.60

```
Estimated build time:           0.1 SBU
Estimated required disk space:  15 MB
```

Contents of Man-pages

The Man-pages package contains over 1200 manual pages. This documentation details the C and C++ functions, describes a few important device files and provides documents which would otherwise be missing from other packages.

Installed files: various manual pages

Man-pages Installation Dependencies

Man depends on: Bash, Coreutils, Make.

Installation of Man-pages

Install Man-pages by running:

```
make install
```

Installing Glibc-2.3.2

```
Estimated build time:           12.3 SBU
Estimated required disk space:  784 MB
```

Contents of Glibc

Glibc is the C library that provides the system calls and basic functions such as open, malloc, printf, etc. The C library is used by all dynamically linked programs.

Installed programs: catchsegv, gencat, getconf, getent, glibcbug, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nscd_nischeck, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump and zic

Installed libraries: ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd-compat.a, libc.[a,so], libc_nonshared.a, libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so and libutil.[a,so]

Glibc Installation Dependencies

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

Glibc installation

The Glibc build system is very well self-contained and will install perfectly, even though our compiler specs file and linker are still pointing at `/tools`. We cannot adjust the specs and linker before the Glibc install, because the Glibc autoconf tests would then give bogus results and thus defeat our goal of achieving a clean build.

The test suite for Glibc in this section is considered *critical*. Our advice is to not skip it under any circumstance.

Before starting to build Glibc, remember to unpack the Glibc-linuxthreads again inside the `glibc-2.3.2` directory, and to unset any environment variables that override the default optimization flags.

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

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

Then apply the same patch we used previously:

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

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

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

Now prepare Glibc for compilation:

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

The meaning of the new configure options:

- `--libexecdir=/usr/bin`: This will cause the `pt_chown` program to be installed in the `/usr/bin` directory.
- `--with-headers=/usr/include`: This ensures that the kernel headers in `/usr/include` are used for this build. If you don't pass this switch then the headers from `/tools/include` are used which of course is not ideal (although they should be identical). Using this switch has the advantage that you will be informed immediately should you have forgotten to install the kernel headers into `/usr/include`.

Compile the package:

```
make
```

Test the results:

```
make check
```

The test suite notes from the Section called *Installing Glibc-2.3.2* in Chapter 5 are still very much appropriate here. Be sure to refer back there should you have any doubts.

And install the package:

```
make install
```

The locales that can make your system respond in a different language weren't installed by the above command. Do it with this:

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

An alternative to running the previous command is to install only those locales which you need or want. This can be achieved using the `localedef` command. Information on this can be found in the `INSTALL` file in the `glibc-2.3.2` tree. However, there are a number of locales that are essential for the tests of future packages to pass correctly. The following instructions, in place of the install-locales command 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 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
```

Finally, build the linuxthreads man pages:

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

And install these pages:

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

Configuring Glibc

We need to create the `/etc/nsswitch.conf` file, because, although Glibc provides defaults when this file is missing or corrupt, the Glibc defaults don't work well with networking. Also, our time zone 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

publickey: files

hosts: files dns
networks: files

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

netgroup: db files

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

To find out what time zone you're in, run the following script:

```
tzselect
```

When you've answered a few questions about your location, the script will output the name of your time zone, something like *EST5EDT* or *Canada/Eastern*. Then create the `/etc/localtime` file by running:

```
cp --remove-destination /usr/share/zoneinfo/Canada/Eastern /etc/localtime
```

The meaning of the option:

- `--remove-destination`: This is needed to force removal of the already existing symbolic link. The reason why we copy instead of symlink is to cover the situation where `/usr` is on a separate partition. This could matter, for example, when booted into single user mode.

Of course, instead of *Canada/Eastern*, fill in the name of the time zone that the `tzselect` script gave you.

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 when you run them. However, if there are libraries in directories other than `/lib` and `/usr/lib`, you need to add them to the `/etc/ld.so.conf` file 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 we 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
```

Re-adjusting the toolchain

Now that the new C libraries have been installed, it's time to re-adjust our toolchain. We'll adjust it so that it will link any newly compiled program against the new C libraries. Basically, this is the reverse of what we did in the "locking in" stage in the beginning of the previous chapter.

The first thing to do is to adjust the linker. For this we retained the source and build directories from the second pass over Binutils. Install the adjusted linker by running the following from within the `binutils-build` directory:

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

If you somehow missed the earlier warning to retain the Binutils source and build directories from the second pass in Chapter 5 or otherwise accidentally deleted them or just don't have access to them, don't worry, all is not lost. Just ignore the above command. The result will be that the next package, Binutils, will link against the Glibc libraries in `/tools` rather than `/usr`. This is not ideal, however, our 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` is needed because the Makefile created during the second pass still contains the reference to `/usr/bin/install`, which we obviously haven't installed yet. Some host distributions contain a `ginstall` symbolic link which takes precedence in the Makefile and thus can cause a problem here. The above command takes care of this also.

You can now remove the Binutils source and build directories.

The next thing to do is to amend our GCC specs file so that it points to the new dynamic linker. Just like earlier on, we use a sed to accomplish this:

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

Again, cutting and pasting the above is recommended. And just like before, it is a good idea to check the specs file to ensure the intended changes were actually made.

If you are working on a platform where the name of the dynamic linker is something other than `ld-linux.so.2`, you *must* substitute `ld-linux.so.2` with the name of your platform's dynamic linker in the above commands. Refer back to the Section called *Toolchain technical notes* in Chapter 5 if necessary.

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. For this we are going to perform a simple sanity check:

```
echo 'main(){}' > dummy.c
gcc 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:

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

If you did not receive the output as shown above, or received no output at all, then something is seriously wrong. You will need to investigate and retrace your steps to find out where the problem is and correct it. There is no point in continuing until this is done. Most likely something went wrong with the specs file amendment above. Note especially that `/lib` now appears as the prefix of our dynamic linker. Of course, if you are working on a platform where the name of the dynamic linker is something other than `ld-linux.so.2`, then the output will be slightly different.

Once you are satisfied that all is well, clean up the test files:

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

Installing Binutils-2.14

```
Estimated build time:           1.4 SBU
Estimated required disk space:  167 MB
```

Contents of Binutils

Binutils is a collection of software development tools containing a linker, assembler and other tools to work with object files and archives.

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]

Binutils Installation Dependencies

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

Installation of Binutils

Now is an appropriate time to verify that your pseudo terminals (PTYs) are working properly inside the chroot environment. We will again quickly check that everything is set up correctly by performing a simple test:

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

If you receive the message:

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

Your chroot environment is not set up for proper PTY operation. In this case there is no point in running the test suites for Binutils and GCC until you are able to resolve the issue. Please refer back to the Section called *Mounting the proc and devpts file systems* and the Section called *Creating devices (Makedev-1.7)* and perform the recommended steps to fix the problem.

The test suite for Binutils in this section is considered *critical*. Our advice is to not skip it under any circumstances.

This package is known to behave badly when you have changed its default optimization flags (including the -march and -mcpu options). Therefore, if you have defined any environment variables that override default optimizations, such as CFLAGS and CXXFLAGS, we recommend unsetting or modifying 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
```

Now prepare Binutils for compilation:

```
../binutils-2.14/configure \
    --prefix=/usr --enable-shared
```

Compile the package:

```
make tooldir=/usr
```

Normally, the *tooldir* (the directory where the executables end up) is set to $(exec_prefix)/$(target_alias), which expands into, for example, `/usr/i686-pc-linux-gnu`. Since we only build for our own system, we don't need this target specific directory in `/usr`. That 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).

Test the results:

```
make check
```

The test suite notes from the Section called *Installing Binutils-2.14 - Pass 2* in Chapter 5 are still very much appropriate here. Be sure to refer back there should you have any doubts.

Install the package:

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

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

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

Installing GCC-3.3.1

```
Estimated build time:           11.7 SBU
Estimated required disk space:  294 MB
```

Contents of GCC

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

Installed programs: c++, cc (link to gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug, and gcov

Installed libraries: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] and libsupc++.a

GCC Installation Dependencies

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

Installation of GCC

The test suite for GCC in this section is considered *critical*. Our advice is to not skip it under any circumstance.

This package is known to behave badly when you have changed its default optimization flags (including the -march and -mcpu options). Therefore, if you have defined any environment variables that override default optimizations, such as CFLAGS and CXXFLAGS, we recommend unsetting or modifying them when building GCC.

This time we will build both the C and the C++ compiler, so you'll have to unpack the GCC-core *and* the GCC-g++ tarball — they will unfold into the same directory. You should likewise extract the GCC-testsuite package. The full GCC package contains even more compilers. Instructions for building these can be found at `http://www.linuxfromscratch.org/blfs/view/stable/general/gcc.html`.

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

The second patch here suppresses the installation of libiberty from GCC, as we will use the one provided by binutils instead. Be careful *not* to apply the GCC specs patch from Chapter 5 here.

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

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

Now prepare GCC for compilation:

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

Compile the package:

```
make
```

Test the results, but don't stop at errors (you'll remember the few known ones):

```
make -k check
```

The test suite notes from the Section called *Installing GCC-3.3.1 - Pass 2* in Chapter 5 are still very much appropriate here. Be sure to refer back there should you have any doubts.

And install the package:

```
make install
```

Some packages expect the C PreProcessor to be installed in the `/lib` directory. To honor 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
```

At this point it is strongly recommended to repeat the sanity check we performed earlier in this chapter. Refer back to the Section called *Re-adjusting the toolchain* and repeat the check. If the results are wrong, then most likely you erroneously applied the GCC Specs patch from Chapter 5.

Installing Coreutils-5.0

```
Estimated build time:           0.9 SBU
Estimated required disk space:  69 MB
```

Contents of Coreutils

The Coreutils package contains a whole series of basic shell utilities.

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, kill, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami and yes

Coreutils Installation Dependencies

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

Installation of Coreutils

Normally the functionality of `uname` is somewhat broken, in that the `-p` switch always returns "unknown". The following patch fixes this behaviour for Intel architectures:

```
patch -Np1 -i ../coreutils-5.0-uname.patch
```

We do not want Coreutils to install its version of the `hostname` program, because it is inferior to the version provided by Net-tools. Prevent its installation by applying a patch:

```
patch -Np1 -i ../coreutils-5.0-hostname-2.patch
```

Now prepare Coreutils for compilation:

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

Compile the package:

```
make
```

The `su` program from Coreutils wasn't installed in Chapter 5 because it needed *root* privilege to do so. We're going to need it in a few moments for the test suite. Therefore we work around the problem by installing it now:

```
make install-root
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. However, this particular test suite makes some assumptions with regards to the presence of non-root users and groups that don't apply this early into the LFS build. We therefore create a dummy system user and two dummy groups to allow the tests to run properly. Should you choose not to run the test suite, skip down to "Install the package". The following commands will prepare us for 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
```

Some tests are meant to run as *root*:

```
make check-root
```

The remainder of the tests are run as the *dummy* user:

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

Remove the dummy groups and user name:

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

Install the package:

```
make install
```

And move some programs to their proper locations:

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

Finally, create a few necessary symlinks:

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

Installing Zlib-1.1.4

```
Estimated build time:           0.1 SBU
Estimated required disk space:  1.5 MB
```

Contents of Zlib

The Zlib package contains the libz library, which is used by some programs for its compression and uncompression functions.

Installed libraries: libz[a,so]

Zlib Installation Dependencies

Zlib depends on: Binutils, Coreutils, GCC, Glibc, Make, Sed.

Installation of Zlib

Zlib has a potential buffer overflow in its *gzprintf()* function, that, though difficult to take advantage of, should be taken care of by applying this patch:

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

Now prepare Zlib for compilation:

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

Note: Zlib is known to build its shared library incorrectly if a CFLAGS is specified in the environment. If you are using your own CFLAGS variables, ensure you add the **-fPIC** directive during this stage, and remove it afterwards.

Compile the package:

```
make
```

Install the shared libraries:

```
make install
```

Now also build the non-shared libraries:

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

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make test
```

And install the package:

```
make install
```

The shared Zlib library should be installed in the `/lib` directory. That way, in the event that you must boot without the `/usr` directory, vital system programs will still have access to the library:

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

The `/usr/lib/libz.so` symlink is linked to a file which no longer exists, because we moved it. Create a symbolic link to the new location of the library:

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

Zlib does not install its manual page. Issue the following command to install this documentation:

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

Installing Lfs-Utils-0.3

```
Estimated build time:           0.1 SBU
Estimated required disk space:  1.1 MB
```

Contents of Lfs-Utils

The Lfs-Utils package contains some miscellaneous programs used by various packages, but are not large enough to warrant their own individual package.

Installed programs: mktemp, tempfile, http-get and iana-net

Installed files: protocols, services

Lfs-Utils Installation Dependencies

(No dependencies checked yet.)

Installation of Lfs-Utils

Compile the package:

```
make
```

And install it:

```
make install
```

Now copy two supporting files included in the Lfs-Utils tarball to their destination:

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

The `/etc/services` file is used to resolve service numbers to human-readable names, and the `/etc/protocols` does the same for protocol numbers.

Installing Findutils-4.1.20

```
Estimated build time:           0.2 SBU
Estimated required disk space:  7.5 MB
```

Contents of Findutils

The Findutils package contains programs to find files, either on-the-fly (by doing a live recursive search through directories and only showing files that match the specifications) or by searching through a database.

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

Findutils Installation Dependencies

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

Installing Findutils

Prepare Findutils for compilation:

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

By default, the location of the updatedb database is in `/usr/var`. To make the location of `/var/lib/misc/locatedb` file FHS compliant, pass the *--localstatedir=/var/lib/misc* option to `configure`.

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Gawk-3.1.3

```
Estimated build time:           0.2 SBU
Estimated required disk space:  17 MB
```

Contents of Gawk

Gawk is an awk implementation that is used to manipulate text files.

Installed programs: awk (link to gawk), gawk, gawk-3.1.3, grcat, igawk, pgawk, pgawk-3.1.3 and pwcat

Gawk Installation Dependencies

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

Installation of Gawk

First apply a patch to fix the following issues:

- Gawk's default location for some of its executables is `$prefix/libexec/awk`. This location doesn't comply with the FHS, which never even mentions a directory called `libexec`. The patch makes it possible to pass a *--libexecdir* switch to the configure script, so that we can use a more appropriate location for the `grcat` and `pwcat` binaries: `/usr/bin`.
- Gawk's default data directory is `$prefix/share/awk`. But package-specific directories should be named using the package name and version number (for example: `gawk-7.7.2.`) and not simply the package name, as there may be different versions of a package installed on the system. The patch changes the name of the data directory to the correct `$prefix/share/gawk-3.1.3`.
- The patch also ensures that this data directory, including its contents, is removed on a *make uninstall*.

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

Now prepare Gawk for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Ncurses-5.3

```
Estimated build time:           0.6 SBU
Estimated required disk space:  27 MB
```

Contents of Ncurses

The Ncurses package provides character and terminal handling libraries, including panels and menus.

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], libpanel.[a,so]

Ncurses Installation Dependencies

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

Installation of Ncurses

First fix two tiny bugs:

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

The first patch corrects the `etip.h` header file, and the second patch prevents some compiler warnings on the use of deprecated headers.

Now prepare Ncurses for compilation:

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

Compile the package:

```
make
```

Install the package:

```
make install
```

Give the Ncurses libraries execute permissions:

```
chmod 755 /usr/lib/*.5.3
```

And fix a library that shouldn't be executable:

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

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

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

Since the libraries have been moved to `/lib`, a few symlinks are currently pointing towards non-existing files. Recreate those symlinks:

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

Installing Vim-6.2

```
Estimated build time:           0.4 SBU
Estimated required disk space:  34 MB
```

Alternatives to Vim

If you prefer another editor — like Emacs, Joe, or Nano — to Vim, have a look at `http://www.linuxfromscratch.org/blfs/view/stable/postlfs/editors.html` for suggested installation instructions.

Contents of Vim

The Vim package contains a configurable text editor built to enable efficient text editing.

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

Vim Installation Dependencies

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

Installation of Vim

Change the default locations of the `vimrc` and `gvimrc` files to `/etc`.

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
echo '#define SYS_GVIMRC_FILE "/etc/gvimrc"' >> src/feature.h
```

Now prepare Vim for compilation:

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

Compile the package:

```
make
```

And install the package:

```
make install
```

Vim can run in old-fashioned *vi* mode by creating a symlink, which may be created with the following command:

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

If you plan to install the X Window system on your LFS system, you might want to re-compile Vim after you have installed X. Vim comes with a nice GUI version of the editor which requires X and a few other libraries to be installed. For more information read the Vim documentation.

Configuring Vim

By default, vim runs in vi compatible mode. Some people might like this, but we have a high preference to run vim in vim mode (else we wouldn't have included vim in this book, but the original vi). Create the `/root/.vimrc` by running the following:

```
cat > /root/.vimrc << "EOF"
" Begin /root/.vimrc

set nocompatible
set bs=2

" End /root/.vimrc
EOF
```

Installing M4-1.4

```
Estimated build time:           0.1 SBU
Estimated required disk space:  3.0 MB
```

Contents of M4

M4 is a macro processor. It copies input to output, expanding macros as it goes. Macros are either built-in or user-defined and can take any number of arguments. Besides just doing macro expansion, m4 has built-in functions for including named files, running Unix commands, doing integer arithmetic, manipulating text in various ways, 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.

Installed program: M4 248

M4 Installation Dependencies

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

Installation of M4

Prepare M4 for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Bison-1.875

```
Estimated build time:           0.6 SBU
Estimated required disk space:  10.6 MB
```

Contents of Bison

Bison is a parser generator, a replacement for yacc. Bison generates a program that analyzes the structure of a text file.

Installed programs: bison and yacc

Installed library: liby.a

Bison Installation Dependencies

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

Installation of Bison

First we use a patch to bison, backported from CVS, which fixes a minor compilation problem with some packages:

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

Prepare Bison for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so (and takes a long time):

```
make check
```

And install the package:

```
make install
```

Installing Less-381

```
Estimated build time:           0.1 SBU
Estimated required disk space:  3.4 MB
```

Contents of Less

Less is a file pager, or text viewer. It displays the contents of a file, or stream, and has the ability to scroll. Less has a few features not included in the `more` pager, such as the ability to scroll backwards.

Installed programs: less, lessecho and lesskey

Less Installation Dependencies

Less depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

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 their configuration files.

Compile the package:

```
make
```

And install it:

```
make install
```

Installing Groff-1.19

```
Estimated build time:           0.5 SBU
Estimated required disk space:  43 MB
```

Contents of Groff

The Groff package includes several text processing programs for text formatting. Groff translates standard text and special commands into formatted output, such as what you see in a manual page.

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)

Groff Installation Dependencies

Groff depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation of Groff

Groff expects the environment variable PAGE to contain the default paper size. For those in the United States, the command below is appropriate. If you live elsewhere, you may want to change *PAGE=letter* to *PAGE=A4*.

Prepare Groff for compilation:

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

Compile the package:

```
make
```

And install it:

```
make install
```

Some documentation programs, such as `xman`, will not work work properly without the following symlinks:

```
ln -s soelim /usr/bin/zsoelim
ln -s eqn /usr/bin/geqn
ln -s tbl /usr/bin/gtbl
```

Installing Sed-4.0.7

```
Estimated build time:           0.2 SBU
Estimated required disk space:  5.2 MB
```

Contents of Sed

sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline).

Installed program: Sed 260

Sed Installation Dependencies

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

Installation of Sed

Prepare Sed for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Flex-2.5.4a

```
Estimated build time:           0.1 SBU
Estimated required disk space:  3.4 MB
```

Contents of Flex

The Flex package is used to generate programs which recognize patterns in text.

Installed programs: flex, flex++ (link to flex) and lex

Installed library: libfl.a

Flex Installation Dependencies

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

Installation of Flex

Prepare Flex for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make bigcheck
```

And 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 don't know about `flex` yet and try to run its predecessor `lex`. To support those programs, create a shell 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
```

Installing Gettext-0.12.1

```
Estimated build time:           6.9 SBU
Estimated required disk space:  55 MB
```

Contents of Gettext

The Gettext package is used for internationalization and localization. Programs can be compiled with Native Language Support (NLS) which enable them to output messages in the user's native language.

Installed programs: autopoint, config.charset, config.rpath, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, project-id, team-address, trigger, urlget, user-email and xgettext

Installed libraries: libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] and libgettextsrc[a,so]

Gettext Installation Dependencies

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

Installation of Gettext

Prepare Gettext for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so (and takes a very long time):

```
make check
```

And install the package:

```
make install
```

Installing Net-tools-1.60

```
Estimated build time:           0.1 SBU
Estimated required disk space:  9.4 MB
```

Contents of Net-tools

The Net-tools package contains a collection of programs which form the base of Linux networking.

Installed programs: arp, dnsdomainname (link to hostname), domainname (link to hostname), hostname, ifconfig, nameif, netstat, nisdomainname (link to hostname), plipconfig, rarp, route, slattach and ypdomainname (link to hostname)

Net-tools Installation Dependencies

Net-tools depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make.

Installation of Net-tools

If you don't know what to answer to all the questions asked during the `make config` phase below, then just accept the defaults. This will be just fine in the majority of cases. What you're asked here is a bunch of questions about which network protocols you've enabled in your kernel. The default answers will enable the tools from this package to work with the most common protocols: TCP, PPP, and several others. You still need to actually enable these protocols in the kernel — what you do here is merely telling the package to include support for those protocols in its programs, but it's up to the kernel to make the protocols available.

First fix a small syntax problem in the sources of the mii-tool program:

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

Now prepare Net-tools for compilation with:

```
make config
```

If you intend to accept the default settings, you may skip the questions generated by *make config* by running `yes "" | make config` instead.

Compile the package:

```
make
```

And install it:

```
make update
```

Installing Inetutils-1.4.2

```
Estimated build time:           0.2 SBU
Estimated required disk space:  11 MB
```

Contents of Inetutils

The Inetutils package contains network clients and servers.

Installed programs: ftp, ping, rcp, rlogin, rsh, talk, telnet and tftp

Inetutils Installation Dependencies

Inetutils depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Installation of Inetutils

Prepare Inetutils for compilation:

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

The meaning of the configure options:

- `--disable-syslogd`: This option prevents inetutils from installing the System Log Daemon, which is installed with the Sysklogd package.
- `--disable-logger`: This option prevents inetutils from installing the logger program, which is used by scripts to pass messages to the System Log Daemon. We do not install it because Util-linux installs a better version later.
- `--disable-whois`: This option disables the building of the inetutils whois client, which is woefully 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/stable/basicnet/inetutils.html`. Note that better replacements are available for many of these servers.

Compile the package:

```
make
```

Install it:

```
make install
```

And move the `ping` program to its proper place:

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

Installing Perl-5.8.0

```
Estimated build time:           2.9 SBU
Estimated required disk space:  143 MB
```

Contents of Perl

The Perl package contains perl, the Practical Extraction and Report Language. Perl combines some of the best features of C, sed, awk and sh into one powerful language.

Installed programs: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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: (too many to name)

Perl Installation Dependencies

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

Installation of Perl

Prepare Perl for compilation:

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

If you want more control over the way Perl sets itself up to be built, you can run the interactive `Configure` script instead and modify the way Perl is built. If you think you can live with the (sensible) defaults Perl auto-detects, then just use the command listed above.

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, you first have to create a basic `/etc/hosts` file, needed by a couple of tests to resolve the name *localhost*:

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

Now run the tests, if you wish:

```
make test
```

And install the package:

```
make install
```

Installing Texinfo-4.6

```
Estimated build time:           0.2 SBU
Estimated required disk space:  17 MB
```

Contents of Texinfo

The Texinfo package contains programs used for reading, writing and converting Info documents, which provide system documentation.

Installed programs: info, infokey, install-info, makeinfo, texi2dvi and texindex

Texinfo Installation Dependencies

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

Installation of Texinfo

Prepare Texinfo for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Install the package:

```
make install
```

And 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 your TeX tree if, for example, you plan to install a TeX package later on.

Installing Autoconf-2.57

```
Estimated build time:           2.9 SBU
Estimated required disk space:  7.7 MB
```

Contents of Autoconf

Autoconf produces shell scripts which automatically configure source code.

Installed programs: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate and ifnames

Autoconf Installation Dependencies

Autoconf depends on: Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Installation of Autoconf

Prepare Autoconf for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Automake-1.7.6

```
Estimated build time:           5.3 SBU
Estimated required disk space:  6.8 MB
```

Contents of Automake

Automake generates Makefile.in files, intended for use with Autoconf.

Installed programs: acinstall, aclocal, aclocal-1.7, automake, automake-1.7, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, ylwrap

Automake Installation Dependencies

Automake depends on: Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Installation of Automake

Prepare Automake for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Install the package:

```
make install
```

And create a necessary symbolic link:

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

Installing Bash-2.05b

```
Estimated build time:           1.2 SBU
Estimated required disk space:  27 MB
```

Contents of Bash

bash is the Bourne-Again SHell, which is a widely used command interpreter on Unix systems. The bash program reads from standard input (the keyboard). A user types something and the program will evaluate what he has typed and do something with it, like running a program.

Installed programs: bash, sh (link to bash) and bashbug

Bash Installation Dependencies

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

Installation of Bash

Bash has a number of bugs in it that cause it to not behave the way it is expected at times. Fix this behaviour with the following patch:

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

Prepare Bash for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make tests
```

Install the package:

```
make install
```

And reload the newly compiled `bash` program:

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

Installing File-4.04

```
Estimated build time:           0.1 SBU
Estimated required disk space:  6.3 MB
```

Contents of File

File is a utility used to determine file types.

Installed program: File 228

Installed library: libmagic.[a,so]

File Installation Dependencies

File depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Zlib.

Installation of File

Prepare File for compilation:

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

Compile the package:

```
make
```

And install it:

```
make install
```

Installing Libtool-1.5

```
Estimated build time:           1.5 SBU
Estimated required disk space:  20 MB
```

Contents of Libtool

GNU libtool is a generic library support script. Libtool hides the complexity of using shared libraries behind a consistent, portable interface.

Installed programs: libtool and libtoolize

Installed libraries: libltdl.[a,so].

Libtool Installation Dependencies

Libtool depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation of Libtool

Prepare Libtool for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Bzip2-1.0.2

```
Estimated build time:           0.1 SBU
Estimated required disk space:  3.0 MB
```

Contents of Bzip2

Bzip2 is a block-sorting file compressor which generally achieves a better compression than the traditional `gzip` does.

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

Bzip2 Installation Dependencies

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

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
```

Install it:

```
make install
```

And install the shared `bzip2` binary into the `/bin` directory, then 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}
mv /usr/bin/{bzip2recover,bzless,bzmore} /bin
ln -s bzip2 /bin/bunzip2
ln -s bzip2 /bin/bzcat
```

Installing Diffutils-2.8.1

```
Estimated build time:           0.1 SBU
Estimated required disk space:  7.5 MB
```

Contents of Diffutils

The programs from this package show you the differences between two files or directories. It's most common use is to create software patches.

Installed programs: cmp, diff, diff3 and sdiff

Diffutils Installation Dependencies

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

Installation of Diffutils

Prepare Diffutils for compilation:

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

Compile the package:

```
make
```

And install it:

```
make install
```

Installing Ed-0.2

```
Estimated build time:           0.1 SBU
Estimated required disk space:  3.1 MB
```

Contents of Ed

GNU ed is an 8-bit clean, POSIX-compliant line editor.

Installed programs: ed and red (link to ed)

Ed Installation Dependencies

Ed depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation of Ed

Ed isn't something which many people use. It's installed here because it can be used by the patch program if you encounter an ed-based patch file. This happens rarely because diff-based patches are preferred these days.

Ed normally uses the mktemp function to create temporary files in `/tmp`, but this function contains a vulnerability (see the section on Temporary Files in `http://en.tldp.org/HOWTO/Secure-Programs-HOWTO/avoid-race.html`). The following patch makes Ed use mkstemp instead, which is the recommended way to create temporary files.

Apply the patch:

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

Now prepare Ed for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Install the package:

```
make install
```

And move the programs to the `/bin` directory, so they can be used in the event that the `/usr` partition is unavailable.

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

Installing Kbd-1.08

```
Estimated build time:          0.1 SBU
Estimated required disk space:  12 MB
```

Contents of Kbd

Kbd contains keytable files and keyboard utilities.

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

Kbd Installation Dependencies

Kbd depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, Gzip, M4, Make, Sed.

Installation of Kbd

By default some of Kbd's utilities (`setlogcons`, `setvesablank` and `getunimap`) are not installed. First enable the compilation of these utilities:

```
patch -Np1 -i ../kbd-1.08-more-programs.patch
```

Now prepare Kbd for compilation:

```
./configure
```

Compile the package:

```
make
```

And install it:

```
make install
```

Installing E2fsprogs-1.34

```
Estimated build time:           0.6 SBU
Estimated required disk space:  48.4 MB
```

Contents of E2fsprogs

E2fsprogs provides the filesystem utilities for use with the ext2 filesystem. It also supports the ext3 filesystem with journaling support.

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]

E2fsprogs Installation Dependencies

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

Installation of E2fsprogs

It is recommended to build E2fsprogs outside of the source tree:

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

Prepare E2fsprogs for compilation:

```
../e2fsprogs-1.34/configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs
```

The meaning of the configure options:

- `--with-root-prefix=""`: Certain programs (such as the e2fsck program) are considered essential programs. When, for example, `/usr` isn't mounted, these essential program have to be available. They belong in directories like `/lib` and `/sbin`. If this option isn't passed to E2fsprogs's configure, the programs are placed in the `/usr` directory, which is not what we want.
- `--enable-elf-shlibs`: This creates the shared libraries which some programs in this package make use of.

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

Install most of the package:

```
make install
```

And install also the shared libraries:

```
make install-libs
```

Installing Grep-2.5.1

```
Estimated build time:           0.1 SBU
Estimated required disk space:  5.8 MB
```

Contents of Grep

Grep is a program used to print lines from a file matching a specified pattern.

Installed programs: egrep (link to grep), fgrep (link to grep) and grep

Grep Installation Dependencies

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

Installation of Grep

Prepare Grep for compilation:

```
./configure --prefix=/usr --bindir=/bin \
    --with-included-regex
```

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Grub-0.93

```
Estimated build time:           0.2 SBU
Estimated required disk space:  10 MB
```

Contents of Grub

The Grub package contains a bootloader.

Installed programs: grub, grub-install, grub-md5-crypt, grub-terminfo and mbchk

Grub Installation Dependencies

Grub depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Installation of Grub

This package is known to behave badly when you have changed its default optimization flags (including the -march and -mcpu options). Therefore, if you have defined any environment variables that override default optimizations, such as CFLAGS and CXXFLAGS, we recommend unsetting them when building Grub.

First fix a compilation problem with GCC-3.3.1:

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

Now prepare Grub for compilation:

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

Compile the package:

```
make
```

And install it:

```
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 your hardware.

The `i386-pc` directory also contains a number of `*stage1_5` files, different ones for different filesystems. Have a look at the ones available and copy the appropriate ones to the `/boot/grub` directory. Most people will copy the `e2fs_stage1_5` and/or `reiserfs_stage1_5` files.

Installing Gzip-1.3.5

```
Estimated build time:           0.1 SBU
Estimated required disk space:  2.6 MB
```

Contents of Gzip

The Gzip package contains programs to compress and decompress files using the Lempel-Ziv coding (LZ77).

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

Gzip Installation Dependencies

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

Installation of Gzip

Prepare Gzip for compilation:

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

The gzexe program has the location of the `gzip` binary hard-wired into it. Because we later change the location of this latter binary, the following command assures that the new location gets placed into the binary:

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

Compile the package:

```
make
```

Install the package:

```
make install
```

And 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
```

Installing Man-1.5m2

```
Estimated build time:          0.1 SBU
Estimated required disk space:  1.9MB
```

Contents of Man

Man is a man pager.

Installed programs: apropos, makewhatis, man, man2dvi, man2html and whatis

Man Installation Dependencies

Man depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation of Man

We'll make three adjustments to the sources of Man.

The first patch comments out the "MANPATH /usr/man" line in the `man.conf` file to prevent redundant results when using programs such as `whatis`:

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

The second patch adds the *-R* option to the *PAGER* variable so that escape sequences are handled properly:

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

The third and last patch prevents a problem when man pages not formatted with more than 80 columns are used in conjunction with recent releases of `groff`:

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

Now prepare Man for compilation:

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

The meaning of the configure options:

- `-default`: This tells the configure script to select a sensible set of default options. For example: only English man pages, no message catalogs, man not suid, handle compressed man pages, compress cat pages, create cat pages whenever the appropriate directory exists, follow FHS by putting cat pages under /var/cache/man provided that that directory exists.
- `-confdir=/etc`: This tells the `man` program to look for the `man.conf` configuration file in the `/etc` directory.

Compile the package:

```
make
```

And install it:

```
make install
```

If you wish to disable SGR escape sequences, you should edit the man.conf file and add the `-c` argument to nroff.

You may want to also take a look at the BLFS page at `http://www.linuxfromscratch.org/blfs/view/cvs/postlfs/compressdoc.html` which deals with formatting and compression issues for man pages.

Installing Make-3.80

```
Estimated build time:           0.2 SBU
Estimated required disk space:  8.8 MB
```

Contents of Make

Make determines, automatically, which pieces of a large program need to be recompiled and issues the commands to recompile them.

Installed program: Make   249

Make Installation Dependencies

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

Installation of Make

Prepare Make for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Modutils-2.4.25

```
Estimated build time:           0.1 SBU
Estimated required disk space:  2.9 MB
```

Contents of Modutils

The Modutils package contains programs that you can use to work with kernel modules.

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)

Modutils Installation Dependencies

Modutils depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Glibc, Grep, M4, Make, Sed.

Installation of Modutils

Prepare Modutils for compilation:

```
./configure
```

Compile the package:

```
make
```

And install it:

```
make install
```

Installing Patch-2.5.4

```
Estimated build time:           0.1 SBU
Estimated required disk space:  1.9 MB
```

Contents of Patch

The patch program modifies a file according to a patch file. A patch file usually is a list, created by the diff program, that contains instructions on how an original file needs to be modified.

Installed program: Patch   255

Patch Installation Dependencies

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

Installation of Patch

Prepare Patch for compilation:

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

Again, the preprocessor flag `-D_GNU_SOURCE` is only needed on the PowerPC platform. On other architectures you can leave it out.

Compile the package:

```
make
```

And install it:

```
make install
```

Installing Procinfo-18

```
Estimated build time:           0.1 SBU
Estimated required disk space:  0.2 MB
```

Contents of Procinfo

The procinfo program gathers system data, such as memory usage and IRQ numbers, from the `/proc` directory and formats this data in a meaningful way.

Installed programs: lsdev, procinfo and socklist

Procinfo Installation Dependencies

Procinfo depends on: Binutils, GCC, Glibc, Make, Ncurses.

Installation of Procinfo

Compile Procinfo:

```
make LDLIBS=-lncurses
```

The meaning of the make parameter:

- `LDLIBS=-lncurses`: This tells Procinfo to use the `libncurses` library instead of the long-obsolete `libtermcap`.

And install the package:

```
make install
```

Installing Procps-3.1.11

```
Estimated build time:           0.1 SBU
Estimated required disk space:  6.2 MB
```

Contents of Procps

The Procps package provides programs to monitor and halt system processes. Procps gathers information about processes via the `/proc` directory.

Installed programs: free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w and watch

Installed library: libproc.so

Procps Installation Dependencies

Procps depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses.

Installation of Procps

First fix a problem that can crash `w` under certain locale settings:

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

Now compile Procps:

```
make
```

Install it:

```
make install
```

And remove a spurious library link:

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

Installing Psmisc-21.3

```
Estimated build time:           0.1 SBU
Estimated required disk space:  2.2 MB
```

Contents of Psmisc

The Psmisc package contains three programs which help manage the `/proc` directory.

Installed programs: fuser, killall and pstree

Psmisc Installation Dependencies

Psmisc depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed.

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` and not in `/usr/bin`. As the Psmisc programs are often used in bootscripts, they should be available also when the `/usr` filesystem isn't mounted.

Compile the package:

```
make
```

And install it:

```
make install
```

By default Psmisc's `pidof` program isn't installed. Generally, this isn't a problem because we later install the Sysvinit package, which provides a better `pidof` program. But if you're not going to use Sysvinit, you should complete the installation of Psmisc by creating the following symlink:

```
ln -s killall /bin/pidof
```

Installing Shadow-4.0.3

```
Estimated build time:           0.4 SBU
Estimated required disk space:  11 MB
```

Contents of Shadow

The Shadow package was created to strengthen the security of system passwords.

Installed programs: chage, chfn, chpasswd, chsh, dpasswd, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, groups, grpck, grpconv, grpunconv, lastlog, login, logoutd, mkpasswd, newgrp, newusers, passwd, pwck, pwconv, pwunconv, sg (link to newgrp), useradd, userdel, usermod, vigr (link to vipw) and vipw

Shadow Installation Dependencies

Shadow depends on: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation of Shadow

The `login`, `getty` and `init` programs (and some others) maintain a number of logfiles to record who are and who were logged in to the system. These programs, however, don't create these logfiles when they don't exist, so if you want this logging to occur you will have to create the files yourself. The Shadow package needs to detect these files in their proper place, so we create them now, with their proper permissions:

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

The `/var/run/utmp` file lists the users that are currently logged in, the `/var/log/wtmp` file who *were* logged in and when. The `/var/log/lastlog` file shows for each user when he or she last logged in, and the `/var/log/btmp` lists the bad login attempts.

Shadow hard-wires the path to the `passwd` binary within the binary itself, but does this the wrong way. If a `passwd` binary is not present before installing Shadow, the package incorrectly assumes it is going to be located at `/bin/passwd`, but then installs it in `/usr/bin/passwd`. This will lead to errors about not finding `/bin/passwd`. To work around this bug, create a dummy `passwd` file, so that it gets hard-wired properly:

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

The current Shadow suite has a problem that causes the `newgrp` command to fail. The following patch (also appearing in Shadow's CVS code) fixes this problem:

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

Now prepare Shadow for compilation:

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

Compile the package:

```
make
```

And install it:

```
make install
```

Shadow uses two files to configure authentication settings for the system. Install these two config files:

```
cp etc/{limits,login.access} /etc
```

We want to change the password method to enable MD5 passwords which are theoretically more secure than the default "crypt" method and also allow password lengths greater than 8 characters. We also need to change the old `/var/spool/mail` location for user mailboxes to the current location at `/var/mail`. We do this by changing the relevant configuration file while copying it to its destination:

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

Be extra careful when typing all of the above. It is probably safer to cut-and-paste it rather than try and type it all in.

According to the man page of `vipw`, a `vigr` program should exist too. Since the installation procedure doesn't create this program, create a symlink manually:

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

As the `/bin/vipw` symlink is redundant (and even pointing to a non-existent file), remove it:

```
rm /bin/vipw
```

Now move the sg program to its proper place:

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

And move Shadow's dynamic libraries to a more appropriate location:

```
mv /usr/lib/lib{shadow,misc}.so.0* /lib
```

As 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
```

Coreutils has already installed a `groups` program in `/usr/bin`. If you wish, you can remove the one installed by Shadow:

```
rm /bin/groups
```

Configuring Shadow

This package contains utilities to modify users' passwords, add or delete users and groups, and the like. We're not going to explain what 'password shadowing' means. A full explanation can be found in the `doc/HOWTO` file within the unpacked Shadow source tree. There's one thing to keep in mind if you decide to use Shadow support: programs that need to verify passwords (for example xdm, ftp daemons, pop3 daemons) need to be 'shadow-compliant', that is they need to be able to work with shadowed passwords.

To enable shadowed passwords, run the following command:

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

And to enable shadowed group passwords, run the following command:

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

Under normal circumstances, you won't have created any passwords yet. However, if returning to this section to enable shadowing, you should reset any current user passwords with the `passwd` command or any group passwords with the `gpasswd` command.

Installing Sysklogd-1.4.1

```
Estimated build time:           0.1 SBU
Estimated required disk space:  0.5 MB
```

Contents of Sysklogd

The Sysklogd package contains programs for recording system log messages, such as those reported by the kernel.

Installed programs: klogd and syslogd

Sysklogd Installation Dependencies

Sysklogd depends on: Binutils, Coreutils, GCC, Glibc, Make.

Installation of Sysklogd

Compile Sysklogd:

```
make
```

And install it:

```
make install
```

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
```

Installing Sysvinit-2.85

```
Estimated build time:           0.1 SBU
Estimated required disk space:  0.9 MB
```

Contents of Sysvinit

The Sysvinit package contains programs to control the startup, running and shutdown of all other programs.

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

Sysvinit Installation Dependencies

Sysvinit depends on: Binutils, Coreutils, GCC, Glibc, Make.

Installation of Sysvinit

When run levels are changed (for example, when halting the system), init sends the TERM and KILL signals to the processes which it started. Init prints "Sending processes the TERM signal" to the screen. This seems to imply that init is sending these signals to all the currently running processes. To avoid this confusion, the init.c file can be modified, so that the sentence reads "Sending processes started by init the TERM signal".

Edit the halt message:

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

Compile Sysvinit:

```
make -C src
```

And install it:

```
make -C src install
```

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

# End /etc/inittab
EOF
```

Installing Tar-1.13.25

```
Estimated build time:           0.2 SBU
Estimated required disk space:  10 MB
```

Contents of Tar

Tar is an archiving program designed to store and extract files from an archive file known as a tar file.

Installed programs: rmt and tar

Tar Installation Dependencies

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

Installation of Tar

Prepare Tar for compilation:

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

Compile the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. Should you choose to run it, the following command will do so:

```
make check
```

And install the package:

```
make install
```

Installing Util-linux-2.12

```
Estimated build time:           0.2 SBU
Estimated required disk space:  16 MB
```

Contents of Util-linux

The Util-linux package contains a number of miscellaneous utility programs. Some of the more prominent utilities are used to mount, unmount, format, partition and manage disk drives, open tty ports and fetch kernel messages.

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, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, pg, pivot_root, ramsize (link to rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (link to rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (link to swapon), swapon, test.bash, test.tcsh, tunelp, ul, umount, vidmode (link to rdev), whereis and write

Util-linux Installation Dependencies

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

FHS compliance notes

The FHS recommends that we use `/var/lib/hwclock`, instead of the usual `/etc`, as the location for the `adjtime` file. To make the `hwclock` program FHS-compliant, run the following:

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

Installation of Util-linux

Prepare Util-linux for compilation:

```
./configure
```

Compile the package:

```
make HAVE_SLN=yes
```

The meaning of the make parameter:

- `HAVE_SLN=yes`: This prevents the `sln` program (a statically linked `ln`, already installed by Glibc) from being built again.

And install the package:

```
make HAVE_SLN=yes install
```

Installing GCC-2.95.3

```
Estimated build time:           1.5 SBU
Estimated required disk space:  130 MB
```

Installation of GCC

This package is known to behave badly when you have changed its default optimization flags (including the -march and -mcpu options). Therefore, if you have defined any environment variables that override default optimizations, such as CFLAGS and CXXFLAGS, we recommend unsetting or modifying them when building GCC.

This is an older release of GCC which we are going to install for the purpose of compiling the Linux kernel in Chapter 8. This version is recommended by the kernel developers when you need absolute stability. Later versions of GCC have not received as much testing for Linux kernel compilation. Using a later version is likely to work, however, we recommend adhering to the kernel developer's advice and using the version here to compile your kernel.

We don't install the C++ compiler or libraries here. However, there may be reasons why you would want to install them. More information can be found at `http://www.linuxfromscratch.org/blfs/view/stable/general/gcc2.html`.

We'll install this older release of GCC into the non-standard prefix of `/opt` so as to avoid interfering with the system GCC already installed in `/usr` .

Apply the patches and make a small adjustment:

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

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

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

Compile and install the compiler:

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

Revised chroot command

From now on when you exit the chroot environment and wish to re-enter it, you should run 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 being there is no longer any need to use programs from the `/tools` directory. However, we don't want to remove the `/tools` directory just yet. There is still some use for it towards the end of the book.

Installing LFS-Bootscripts-1.12

```
Estimated build time:           0.1 SBU
Estimated required disk space:  0.3 MB
```

Contents of LFS-bootscripts

The LFS-Bootscripts package contains SysV init style shell scripts. These scripts do various tasks such as check filesystem integrity during boot, load keymaps, set up networks and halt processes at shutdown.

Installed scripts: checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountproc, network, rc, reboot, sendsignals, setclock, swap, sysklogd and template

LFS-Bootscripts Installation Dependencies

Bzip2 depends on: Bash, Coreutils.

Installation of LFS-Bootscripts

We will be using SysV style init scripts. We have chosen this style because it is widely used and we feel comfortable with it. If you would prefer to try something else, Marc Heerdink has written a hint about BSD style init scripts, to be found at `http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt`. And if you'd like something more radical, search the LFS mailing lists for depinit.

If you decide to use BSD style, or some other style scripts, you can skip the next chapter when you arrive at it and move on to Chapter 8.

Install the boot scripts:

```
cp -a rc.d sysconfig /etc
```

Give *root* ownership of the scripts:

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

Configuring system components

Now that all software is installed, all that we need to do is perform a few configuration tasks.

Configuring your keyboard

Few things are more annoying than using Linux while a wrong keymap for your keyboard is loaded. If you have a standard US keyboard, however, you can skip this section, as the US keymap is the default as long as you don't change it.

To change the default keymap, create the `/usr/share/kbd/keymaps/defkeymap.map.gz` symlink by running the following command:

```
ln -s path/to/keymap /usr/share/kbd/keymaps/defkeymap.map.gz
```

Of course, replace `path/to/keymap` with the path and name of your keyboard's map file. For example, if you have a Dutch keyboard, you would use `i386/qwerty/nl.map.gz`.

Another way to set your keyboard's layout is to compile the keymap into the kernel. This ensures that your keyboard will always work as expected, even when you boot into maintenance mode (by passing `init=/bin/sh' to the kernel), as then the bootscript that normally sets up your keymap isn't run.

Run the following command to patch the current default keymap into the kernel source. You will have to repeat this command whenever you unpack a new kernel:

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

Setting the root password

Choose a password for user root and set it by running the following command:

```
passwd root
```

Chapter 7 Setting up system boot scripts

Introduction

This chapter will set up the bootscripts that you installed in chapter 6. Most of these scripts will work without needing to modify them, but a few do require additional configuration files set up as they deal with hardware dependent information.

How does the booting process with these scripts work?

Linux uses a special booting facility named SysVinit. It's based on a concept of *runlevels*. It can be widely different from one system to another, so it can't be assumed that because things worked in <insert distro name> they should work like that in LFS too. LFS has its own way of doing things, but it respects generally accepted standards.

SysVinit (which we'll call *init* from now on) works using a runlevels scheme. There are 7 (from 0 to 6) runlevels (actually, there are more runlevels but they are for special cases and generally not used. The init man page describes those details), and each one of those corresponds to the things the computer is supposed to do when it starts up. The default runlevel is 3. Here are the descriptions of the different runlevels as they are often 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 runlevels is `init <runlevel>` where <runlevel> is the target runlevel. For example, to reboot the computer, a user would issue the init 6 command. The reboot command is just an alias, as is the halt command an alias to init 0.

There are a number of directories under /etc/rc.d that look like like rc?.d where ? is the number of the runlevel and rcsysinit.d which contain 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 sooner it gets executed. When init switches to another runlevel, the appropriate services get killed and others get started.

The real scripts are in /etc/rc.d/init.d. They do all the work, and the symlinks all point to them. Killing links and starting links point to the same script in /etc/rc.d/init.d. That's because the scripts can be called with different parameters like start, stop, restart, reload, status. When a K link is encountered, the appropriate script is run with the stop argument. When a S link is encountered, the appropriate script is run with the start argument.

There is one exception. 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 it is that when you are going to reboot or halt the system, you don't want to start anything, only stop the system.

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 doesn't 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's your own LFS system). The files given here are just an example of how it can be done in a nice way (well, what we consider nice — you may hate it).

Configuring the setclock script

This setclock script reads the time from your hardware clock (also known as BIOS or CMOS clock) and either converts that time to localtime using the `/etc/localtime` file (if the hardware clock is set to GMT) or not (if the hardware clock is already set to localtime). There is no way to auto-detect whether the hardware clock is set to GMT or not, so we need to configure that here ourselves.

Change the value of the *UTC* variable below to a *0* (zero) if your hardware clock is not set to GMT 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
```

Now, you may want to take a look at a very good hint explaining how we deal with time on LFS at `http://www.linuxfromscratch.org/hints/downloads/files/time.txt`. It explains issues such as time zones, UTC, and the TZ environment variable.

Do I need the loadkeys script?

If you decided to compile your keymap file directly into the kernel back at the end of Chapter 6, then you strictly speaking don't need to run this loadkeys script, since the kernel has already set up the keymap for you. You can still run it if you want, it isn't going to hurt you. It could even be beneficial to keep it in case you run a lot of different kernels and don't remember or want to compile the keymap into every kernel you lay your hands on.

If you decided you don't need to, or don't want to use the loadkeys script, remove the `/etc/rc.d/rcsysinit.d/S70loadkeys` symlink.

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. If you want to turn on this periodic timestamp mark, edit the `sysklogd` script and make the changes accordingly. See `man syslogd` for more information.

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. You should not enter the FQDN (Fully Qualified Domain Name) here. That information will be put in the `/etc/hosts` file later on.

Creating the /etc/hosts file

If a network card is to be configured, you have to decide on the IP-address, FQDN and possible aliases for use in the /etc/hosts file. The syntax is:

```
<IP address> myhost.mydomain.org aliases
```

You should 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.

If you aren't going to use a network card, you still need to come up with a FQDN. This is necessary for certain programs to operate correctly.

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 <value of HOSTNAME>.mydomain.com <value of HOSTNAME> localhost

# End /etc/hosts (no network card version)
EOF
```

If a network card is to be configured, create the `/etc/hosts` file by running:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (network card version)

127.0.0.1 localhost.localdomain localhost
192.168.1.1 <value of HOSTNAME>.mydomain.org <value of HOSTNAME>

# End /etc/hosts (network card version)
EOF
```

Of course, the 192.168.1.1 and <value of HOSTNAME>.mydomain.org have to be changed to your liking (or requirements if assigned an IP-address by a network/system administrator and this machine is planned to be connected to an existing network).

Configuring the network script

This section only applies if you're going to configure a network card.

If you don't have any network cards, you are most likely not going to create any configuration files relating to network cards. If that is the case, you must remove the `network` symlinks from all the runlevel directories (`/etc/rc.d/rc*.d`)

Configuring default gateway

If you're on a network you may need to set up the default gateway for this machine. This is done by adding the proper values to the /etc/sysconfig/network file by running the following:

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

The values for GATEWAY and GATEWAY_IF need to be changed to match your network setup. GATEWAY contains the IP address of the default gateway, and GATEWAY_IF contains the network interface through which the default gateway can be reached.

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 you decide to rename or move this /etc/sysconfig/network-devices directory, make sure you update the /etc/sysconfig/rc file as well and update the network_devices by providing it with the new path.

Now, new files are created in that directory containing the following. The following command creates a sample ifconfig.eth0 file:

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

Of course, the values of those variables have to be changed in every file to match the proper setup. If the ONBOOT variable is set to yes, the network script will bring it up during the booting of the system. If set to anything else but yes, it will be ignored by the network script and thus not brought up.

Chapter 8
Making the LFS system bootable

Introduction

This chapter will make LFS bootable. This chapter deals with creating a new fstab file, building a new kernel for the new LFS system and installing the Grub bootloader so that the LFS system can be selected for booting at startup.

Creating the /etc/fstab file

The `/etc/fstab` file is used by some programs to determine where partitions are to be mounted by default, which file systems must be checked and in which order. Create a new file systems table like this:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab

# filesystem  mount-point  fs-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
devpts        /dev/pts     devpts   gid=4,mode=620  0     0
shm           /dev/shm     tmpfs    defaults        0     0

# End /etc/fstab
EOF
```

Of course, replace xxx, yyy and `fff` with the values appropriate for your system — for example hda2, hda5 and `reiserfs`. For all the details on the six fields in this table, see `man 5 fstab`.

When using a reiserfs partition, the *1 1* at the end of the line should be replaced with *0 0*, as 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. Your kernel must have the required support built into it for this to work — more about this in the next section. Please note that currently very little software actually uses POSIX shared memory. Therefore you can 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 you may consider adding to your `fstab` file. One example is a line to use if you intend to use USB devices:

```
usbfs        /proc/bus/usb  usbfs    defaults    0     0
```

This option will of course only work if you have the relevant support compiled into your kernel.

Installing Linux-2.4.22

```
Estimated build time:           All default options: 4.20 SBU
Estimated required disk space:  All default options: 181 MB
```

Contents of Linux

The Linux kernel is at the core of every Linux system. It's what makes Linux tick. When a computer is turned on and boots a Linux system, the very first piece of Linux software that gets loaded is the kernel. The kernel initializes the system's hardware components: serial ports, parallel ports, sound cards, network cards, IDE controllers, SCSI controllers and a lot more. In a nutshell the kernel makes the hardware available so that the software can run.

Installed files: the kernel and the kernel headers

Linux Installation Dependencies

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

Installation of the kernel

Building the kernel involves a few steps: configuration, compilation, and installation. If you don't like the way this book configures the kernel, view the `README` file in the kernel source tree for alternative methods.

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. You shouldn't rely on the source tree being clean after untarring.

Configure the kernel via a menu-driven interface:

```
make menuconfig
```

`make oldconfig` may be more appropriate in some situations. See the `README` file for more information.

If you wish, you may skip kernel configuration by simply copying the kernel config file, `.config`, from your host system (assuming it is available) to the unpacked `linux-2.4.22` directory. However, we don't recommend this option. You're much better off exploring all the configuration menus and creating your own 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.

Verify dependencies and create dependency information files:

```
make CC=/opt/gcc-2.95.3/bin/gcc dep
```

Compile the kernel image:

```
make CC=/opt/gcc-2.95.3/bin/gcc bzImage
```

Compile the drivers which have been configured as modules:

```
make CC=/opt/gcc-2.95.3/bin/gcc modules
```

If you intend to use kernel modules, you will need an `/etc/modules.conf` file. Information pertaining to modules and to kernel configuration in general may be found in the kernel documentation, which is found in the `linux-2.4.22/Documentation` directory. The modules.conf man page and the kernel HOWTO at `http://www.tldp.org/HOWTO/Kernel-HOWTO.html` may also be of interest to you.

Install the modules:

```
make CC=/opt/gcc-2.95.3/bin/gcc modules_install
```

As nothing is complete without documentation, build the manual pages that come with the kernel:

```
make mandocs
```

And install these pages:

```
cp -a Documentation/man /usr/share/man/man9
```

Kernel compilation has finished, but some of the files created still reside in the source tree. To complete the installation, two files need to be copied to the `/boot` directory.

The path to the kernel file may vary depending on the platform you're using. Issue the following command to install the kernel:

```
cp arch/i386/boot/bzImage /boot/lfskernel
```

`System.map` is a symbol file for the kernel. It maps the function entrypoints 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
```

Making the LFS system bootable

Your shiny new LFS system is almost complete. One of the last things to do is ensure you can boot it. The instructions below apply only to computers of IA-32 architecture, i.e. mainstream PC's. 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. First, a few cautionary words. You really should be familiar with your current boot loader and any other operating systems present on your hard drive(s) that you might wish to keep bootable. Please make sure that you have an emergency boot disk ready, so that you can rescue your computer if, by any chance, your computer becomes unusable (unbootable).

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. Before we get to that, we highly recommend that you create a Grub boot floppy diskette just in case. 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 we'll 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* the partition number, both starting from zero. This means, for instance, that partition `hda1` is (hd0,0) to Grub, and `hdb2` is (hd1,1). In contrast to Linux, Grub doesn't consider CD-ROM drives to be hard drives, so if you have a CD on `hdb`, for example, and a second hard drive on `hdc`, that second hard drive would still be (hd1).

Using the above information, determine the appropriate designator for your root partition. For the following example, we'll assume your root partition is `hda4`.

First, tell Grub where to search for its `stage{1,2}` files — you can use Tab everywhere to make Grub show the alternatives:

```
root (hd0,3)
```

The following command will overwrite your current boot loader. Don't run the command if this is not what you want. For example, you may be using a third party boot manager to manage your MBR (Master Boot Record). In this scenario, it would probably make more sense to install Grub into the "boot sector" of the LFS partition, in which case the command would become: `setup (hd0,3)`.

Then tell it to install itself into the MBR (Master Boot Record) of `hda`:

```
setup (hd0)
```

If all is well, Grub will have reported finding its files in `/boot/grub`. That's all there is to it:

```
quit
```

Now we need to 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 5.0
root (hd0,3)
kernel /boot/lfskernel root=/dev/hda4 ro
EOF
```

You may want to add an entry for your host distribution. 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 ro
initrd /boot/initrd-2.4.20
EOF
```

Also, if you happen to dual-boot Windows, the following entry should allow booting it:

```
cat >> /boot/grub/menu.lst << "EOF"
title Windows
rootnoverify (hd0,0)
chainloader +1
EOF
```

If `info grub` doesn't tell you all you want to know, you can find more information regarding Grub on its website, located at: `http://www.gnu.org/software/grub`.

Chapter 9
The End

The End

Well done! You have finished installing your LFS system. It may have been a long process, but we hope it was worth it. We wish you a lot of fun with your new shiny custom built Linux system.

Now would be a good time to strip all debug symbols from the binaries on your LFS system. If you are not a programmer and don't plan on debugging your software, then you will be happy to know that you can reclaim a few tens of megs by removing debug symbols. This process causes no inconvenience other than not being able to debug the software fully anymore, which is not an issue if you don't know how to debug.

Disclaimer: 98% of the people who use the command mentioned below don't experience any problems. But do make a backup of your LFS system before you run this command. There's a slight chance it may backfire on you and render your system unusable (mostly by destroying your kernel modules and dynamic & shared libraries). This is caused more often by typos than by a problem with the command used.

Having said that, the --strip-debug option we use to strip is quite harmless under normal circumstances. It doesn't strip anything vital from the files. It also is quite safe to use --strip-all on regular programs (don't use that on libraries - they will be destroyed), but it's not as safe, and the space you gain is not all that much. But if you're tight on disk space every little bit helps, so decide for yourself. Please refer to the strip man page for other strip options you can use. The general idea is to not run strip on libraries (other than --strip-debug), just to be on the safe side.

If you are planning to go ahead and perform the strip, special care is needed to ensure you're not running any binaries that are about to be stripped — including the active bash shell. Therefore you'll need to exit the chroot environment and reenter it using a modified chroot command:

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

Now run the following command:

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

Quite a number of files will be reported as having their file format not recognized. Most of these are scripts instead of binaries. These warnings can be safely ignored.

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 are going to ask for help with something at some point) to find out which LFS version you have installed on your system. Create this file by running:

```
echo 5.0 > /etc/lfs-release
```

Get Counted

Want to be counted as an LFS user now that you have finished the book? Head over to `http://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...

Rebooting the system

Now that all of the software has been installed, it's time to exit the chroot environment and reboot the computer. Before we exit the chroot environment, let's unmount any mounted virtual file systems by running:

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

Exit the chroot environment:

```
logout
```

Additionally, now that all software has been installed, there is no longer a need for the `/tools` directory. You may delete it. As this will also remove the temporary copies of Tcl, Expect and DejaGnu, which were used for running the toolchain tests, you will need to recompile and re-install them on your LFS system if you want to use these programs later.

Also you may now want to move the contents of `/sources` to `/usr/src/packages` or something similar (or simply delete them if you've burned them on a CD) and delete the directory.

Before we reboot, let's unmount the LFS partition itself:

```
umount $LFS
```

If you earlier decided to create multiple partitions, you'll need to unmount the other partitions before you unmount $LFS, like this:

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

And now you can reboot your system by running something like:

```
/sbin/shutdown -r now
```

Assuming the Grub boot loader was set up as outlined earlier, the default menu should be set to boot *LFS 5.0* automatically.

After you have rebooted, your LFS system is ready for use and you can start adding your own software.

What now?

We thank you for reading the LFS Book and hope that you've found this book useful and worth your time.

Now that you have finished installing your LFS system, you may be wondering "What now?". In order to answer that question, we have composed 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 can be found at `http://www.linuxfromscratch.org/blfs/`.

- LFS Hints

  The LFS Hints are a collection of small, 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. See Chapter 1 - Mailing lists for more information.

- The Linux Documentation Project

  The goal of the Linux Documentation Project is to collaborate in all of the issues of Linux documentation. The LDP features a large collection of HOWTOs, Guides and man pages; it may be found at `http://www.tldp.org/`.

Part IV - Appendices

Appendix A
Package descriptions and dependencies

Introduction

In this appendix the following aspects of every package installed in this book are described:

- the official download location for the package,
- what the package contains,
- what each program from the package does,
- what the package needs to be compiled.

Most information about these packages (especially the descriptions of them) come from the man pages of those packages. We do not include the entire man page, but just some key elements to make it possible to understand what a program does. To get information on all details of a program, please refer to its man page or info page.

Certain packages are documented in more depth than others, because we just happen to know more about certain packages than about others. If you think anything should be added to the following descriptions, please don't hesitate to email the mailing lists. We intend that the list should contain an in-depth description of every package installed, but we can't do it without help.

Please note that currently only what a package does is described and not why it needs to be installed. This may be added later.

Also listed are all of the installation dependencies for all the packages that are installed in this book. The listings will include which programs from which packages are needed to successfully compile the package to be installed.

These are not running dependencies, meaning they don't tell you what programs are needed to use that package's programs, just the ones needed to compile it.

The dependency list can be, from time to time, outdated in regards to the currently used package version. Checking dependencies takes quite a bit of work, so they may lag behind a bit on the package update. But often with minor package updates, the installation dependencies hardly change, so they'll be current in most cases. When we upgrade to a major new release, we'll make sure the dependencies are checked too.

Autoconf

For installation instructions see the Section called *Installing Autoconf-2.57* in Chapter 6.

Official Download Location

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

Contents of Autoconf

Autoconf produces shell scripts which automatically configure source code.

Installed programs: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate and ifnames

Short descriptions

autoconf is a tool for producing 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 is a tool for creating template files of C #define statements for configure to use.

autom4te is a wrapper for the M4 macro processor.

autoreconf comes in handy when there are a lot of autoconf-generated configure scripts around. The program runs autoconf and autoheader repeatedly (where appropriate) to remake the autoconf configure scripts and configuration header templates in a given directory tree.

autoscan can help to create a `configure.in` file for a software package. It examines the source files in a directory tree, searching them for common portability problems and creates a `configure.scan` file that serves as as a preliminary `configure.in` 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 can be helpful when writing a `configure.in` 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 to determine what `configure` needs to check. It can fill in some gaps in a `configure.in` file generated by autoscan.

Autoconf Installation Dependencies

Autoconf depends on: Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Automake

For installation instructions see the Section called *Installing Automake-1.7.6* in Chapter 6.

Official Download Location

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

Contents of Automake

Automake generates Makefile.in files, intended for use with Autoconf.

Installed programs: acinstall, aclocal, aclocal-1.7, automake, automake-1.7, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, ylwrap

Short descriptions

acinstall is a script that installs aclocal-style M4 files.

aclocal generates `aclocal.m4` files based on the contents of `configure.in` files.

automake is a tool for automatically generating `Makefile.in`'s from files called `Makefile.am`. To create all the `Makefile.in` files for a package, run this program in the top level directory. By scanning the `configure.in`s it automatically finds each appropriate `Makefile.am` and generate the corresponding `Makefile.in`.

compile is a wrapper for compilers.

config.guess is a script that attempts to guess the canonical triplet for the given build, host, or target architecture.

config.sub is a configuration validation subroutine script.

depcomp is a script for compiling a program so that not only the desired output is generated but also dependency information.

elisp-comp byte-compiles Emacs Lisp code.

install-sh is a script that installs a program, a script, or a datafile.

mdate-sh is a script that prints the modification time of a file or directory.

missing is a script acting as a common stub for missing GNU programs during an installation.

mkinstalldirs is a script that creates a directory tree.

py-compile compiles a Python program.

ylwrap is a wrapper for lex and yacc.

Automake Installation Dependencies

Automake depends on: Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Bash

For installation instructions see the Section called *Installing Bash-2.05b* in Chapter 6.

Official Download Location

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

Bash Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/bash-2.05b-2.patch`

Contents of Bash

bash is the Bourne-Again SHell, which is a widely used command interpreter on Unix systems. The bash program reads from standard input (the keyboard). A user types something and the program will evaluate what he has typed and do something with it, like running a program.

Installed programs: bash, sh (link to bash) and bashbug

Short descriptions

bash is a widely-used command interpreter. It performs all kinds of expansions and substitutions on a given command line before executing it, thus making this interpreter a powerful tool.

bashbug is a shell script to help the user compose and mail bug reports concerning bash in a standard format.

sh is 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.

Bash Installation Dependencies

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

Binutils

For installation instructions see the Section called *Installing Binutils-2.14* in Chapter 6.

Official Download Location

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

Contents of Binutils

Binutils is a collection of software development tools containing a linker, assembler and other tools to work with object files and archives.

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 figure out which source file and line number are associated with the address.

ar creates, modifies, and extracts from archives. An archive is a single file holding a collection of other files in a structure that makes it possible to retrieve the original individual files (called members of the archive).

as is an assembler. It assembles the output of gcc into object files.

c++filt is used by the linker to demangle C++ and Java symbols, to keep overloaded functions from clashing.

gprof displays call graph profile data.

ld is a linker. It 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 is used to translate one type of object file into another.

objdump displays information about the given object file, with options controlling what particular information to display. The information shown is mostly only 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 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 grand total — for the given object files.

strings outputs for each file given the sequences of printable characters that are of at least the specified length (defaulting to 4) For object files it prints by default only the strings from the initializing and loading sections. For other types of files it scans the whole file.

strip discards symbols from object files.

libiberty contains routines used by various GNU programs, including getopt, obstack, strerror, strtol and strtoul.

libbfd is the Binary File Descriptor library.

libopcodes is a library for dealing with opcodes. It is used for building utilities like objdump. Opcodes are the "readable text" versions of instructions for the processor.

Binutils Installation Dependencies

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

Bison

For installation instructions see the Section called *Installing Bison-1.875* in Chapter 6.

Official Download Location

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

Bison Attribute Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/bison-1.875-attribute.patch`

Contents of Bison

Bison is a parser generator, a replacement for yacc. Bison generates a program that analyzes the structure of a text file.

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 is a wrapper for bison, meant for programs that still call yacc instead of bison. It calls bison with the -y option.

liby.a is the Yacc library containing implementations of Yacc-compatible yyerror and main functions. This library is normally not very useful, but POSIX requires it.

Bison Installation Dependencies

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

Bzip2

For installation instructions see the Section called *Installing Bzip2-1.0.2* in Chapter 6.

Official Download Location

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

Contents of Bzip2

Bzip2 is a block-sorting file compressor which generally achieves a better compression than the traditional `gzip` does.

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 and friends run grep on bzipped files.

bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm with Huffman coding. The compression rate is generally considerably better than that achieved by more conventional compressors using LZ77/LZ78, like `gzip`.

bzip2recover tries to recover data from damaged bzip2 files.

bzless runs less on bzipped files.

bzmore runs more on bzipped files.

libbz2* is the library implementing lossless, block-sorting data compression, using the Burrows-Wheeler algorithm.

Bzip2 Installation Dependencies

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

Coreutils

For installation instructions see the Section called *Installing Coreutils-5.0* in Chapter 6.

Official Download Location

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

Coreutils Hostname Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils-5.0-hostname-2.patch`

Coreutils Uname Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils-5.0-uname.patch`

Contents of Coreutils

The Coreutils package contains a whole series of basic shell utilities.

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, kill, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami and yes

Short descriptions

basename strips any path and a given suffix from the given file name.

cat concatenates files to standard output.

chgrp changes the group ownership of each given file to the given group. The group can be either given a a name or a numeric ID.

chmod changes the permissions of each given 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 given command with the specified directory as the / directory. The given command can be an interactive shell. On most systems only *root* can do this.

cksum prints the 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 parts 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 blocksize and count, while optionally performing conversions on it.

df reports the amount of disk space available (and used) on all mounted filesystems, or only on the filesystems holding the given files.

dir is the same as ls.

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 given file name.

du reports the amount of disk space used by the current directory, or by each of the given directories including all their 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 of a given user.

install copies files while setting their permission modes and, if possible, their owner and group.

join joins from two files the lines that have identical join fields.

kill terminates the given process.

link creates a hard link with the given name to the given file.

ln makes hard links or soft links between files.

logname reports the current user's login name.

ls lists the contents of each given directory. By default it orders the files and subdirectories alphabetically.

md5sum reports or checks MD5 checksums.

mkdir creates directories with the given names.

mkfifo creates FIFOs with the given names.

mknod creates device nodes with the given names. A device node is a character special file, or 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 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 TABs.

pathchk checks whether file names are valid or portable.

pinky is a lightweight finger. 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 from the contents of the given files a permuted index, with each keyword in its context.

pwd reports the name of the current 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 SHA1 checksums.

shred overwrites the given files repeatedly with strange patterns, to make it real hard 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.

stty sets or reports terminal line settings.

su runs a shell with substitute user and group IDs.

sum prints checksum and block counts for each given file.

sync flushes filesystem 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 totally 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.

uptime reports how long the system has been running, how many users are logged on, and the system load averages.

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, and 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 outputs 'y' or a given string repeatedly, until killed.

Coreutils Installation Dependencies

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

DejaGnu

For installation instructions see the Section called *Installing DejaGnu-1.4.3* in Chapter 5.

Official Download Location

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

Contents of DejaGnu

The DejaGnu package contains a framework for testing other programs.

Installed program: runtest

Short description

runtest is the wrapper script that finds the proper expect shell and then runs DejaGnu.

DejaGnu Installation Dependencies

Dejagnu depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Diffutils

For installation instructions see the Section called *Installing Diffutils-2.8.1* in Chapter 6.

Official Download Location

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

Contents of Diffutils

The programs from this package show you the differences between two files or directories. It's most common use is to create software patches.

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.

Diffutils Installation Dependencies

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

E2fsprogs

For installation instructions see the Section called *Installing E2fsprogs-1.34* in Chapter 6.

Official Download Location

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

Contents of E2fsprogs

E2fsprogs provides the filesystem utilities for use with the ext2 filesystem. It also supports the ext3 filesystem with journaling support.

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 is a command line utility to locate and print block device attributes.

chattr changes the attributes of files on a second extended (ext2) filesystem.

compile_et is 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 is a filesystem debugger. It can be used to examine and change the state of an ext2 filesystem.

dumpe2fs prints the super block and blocks group information for the filesystem present on a given device.

e2fsck is used to check, and optionally repair, second extended (ext2) filesystems, and also ext3 filesystems.

e2image is used to save critical ext2 filesystem data to a file.

e2label will display or change the filesystem label on the ext2 filesystem present on a given device.

findfs finds a file system by label or UUID.

fsck is used to check, and optionally repair, filesystems. By default it checks the filesystems listed in `/etc/fstab`

logsave saves the output of a command in a logfile.

lsattr lists the attributes of files on a second extended filesystem.

mk_cmds converts a table of command names and help messages into a C source file suitable for use with the `libss` subsystem library.

mke2fs is used to create a second extended filesystem on the given device.

mklost+found is used to create a `lost+found` directory on a second extended filesystem. It pre-allocates disk blocks to this directory to lighten the task of e2fsck.

resize2fs can be used to enlarge or shrink an ext2 filesystem.

tune2fs is used adjust tunable filesystem parameters on a second extended filesystem.

uuidgen creates new universally unique identifiers (UUID). 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 is the common error display routine.

libe2p is used by dumpe2fs, chattr, and lsattr.

libext2fs contains routines to enable user-level programs to manipulate an ext2 filesystem.

libss is used by debugfs.

libuuid contains routines for generating unique identifiers for objects that may be accessible beyond the local system.

E2fsprogs Installation Dependencies

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

Ed

For installation instructions see the Section called *Installing Ed-0.2* in Chapter 6.

Official Download Location

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

Ed Mkstemp Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/ed-0.2-mkstemp.patch`

Contents of Ed

GNU ed is an 8-bit clean, POSIX-compliant line editor.

Installed programs: ed and red (link to ed)

Short descriptions

ed is a line-oriented text editor. It can be used to create, display, modify and otherwise manipulate text files.

red is a restricted ed — it can only edit files in the current directory and cannot execute shell commands.

Ed Installation Dependencies

Ed depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Expect

For installation instructions see the Section called *Installing Expect-5.39.0* in Chapter 5.

Official Download Location

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

Expect Spawn Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/expect-5.39.0-spawn.patch`

Contents of Expect

The Expect package provides a program that performs programmed dialogue with other interactive programs.

Installed program: Expect   228

Installed library: libexpect5.39.a

Short description

expect "talks" to other interactive programs according to a script.

Expect Installation Dependencies

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

File

For installation instructions see the Section called *Installing File-4.04* in Chapter 6.

Official Download Location

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

Alternate Download Location:
`ftp://gaosu.rave.org/pub/linux/lfs/`

Contents of File

File is a utility used to determine file types.

Installed program: File 228

Installed library: libmagic.[a,so]

Short description

file tries to classify each given file. It does this by performing several tests: filesystem tests, magic number tests, and language tests. The first test that succeeds determines the result.

libmagic contains routines for magic number recognition, used by the file program.

File Installation Dependencies

File depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Zlib.

Findutils

For installation instructions see the Section called *Installing Findutils-4.1.20* in Chapter 6.

Official Download Location

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

Contents of Findutils

The Findutils package contains programs to find files, either on-the-fly (by doing a live recursive search through directories and only showing files that match the specifications) or by searching through a database.

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 4 to 5.

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 filesystem (including other filesystems that are currently mounted, unless told not to) and puts every file name it finds in the database.

xargs can be used to apply a given command to a list of files.

Findutils Installation Dependencies

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

Flex

For installation instructions see the Section called *Installing Flex-2.5.4a* in Chapter 6.

Official Download Location

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

Contents of Flex

The Flex package is used to generate programs which recognize patterns in text.

Installed programs: flex, flex++ (link to flex) and lex

Installed library: libfl.a

Short descriptions

flex is a tool for generating programs that recognize patterns in text. Pattern recognition is useful in many applications. From a set of rules on what to look for flex makes a program that looks for those patterns. The reason to use flex is that it is much easier to specify the rules for than to write the actual pattern-finding program.

flex++ invokes a version of flex that is used exclusively for C++ scanners.

libfl.a is the flex library.

Flex Installation Dependencies

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

Gawk

For installation instructions see the Section called *Installing Gawk-3.1.3* in Chapter 6.

Official Download Location

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

Gawk Libexecdir Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gawk-3.1.3-libexecdir.patch`

Contents of Gawk

Gawk is an awk implementation that is used to manipulate text files.

Installed programs: awk (link to gawk), gawk, gawk-3.1.3, grcat, igawk, pgawk, pgawk-3.1.3 and pwcat

Short descriptions

gawk is a program for manipulating text files. It is the GNU implementation of awk.

grcat dumps the group database `/etc/group`.

igawk gives gawk the ability to include files.

pgawk is the profiling version of gawk.

pwcat dumps the password database `/etc/passwd`.

Gawk Installation Dependencies

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

GCC

For installation instructions see the Section called *Installing GCC-3.3.1* in Chapter 6.

Official Download Location

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

GCC No-Fixincludes Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-no_fixincludes-2.patch`

GCC Specs Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-specs-2.patch`

GCC Suppress-Libiberty Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-3.3.1-suppress-libiberty.patch`

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

GCC-2 Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-2.patch`

GCC-2 No-Fixincludes Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-no-fixinc.patch`

GCC-2 Return-Type Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/gcc-2.95.3-returntype-fix.patch`

Contents of GCC

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

Installed programs: c++, cc (link to gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug, and gcov

Installed libraries: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] and libsupc++.a

Short descriptions

cpp is the C preprocessor. It is used by the compiler to have the #include and #define and such statements expanded in the source files.

g++ is the C++ compiler.

gcc is the C compiler. It is used to translate the source code of a program into assembly code.

gccbug is a shell script used to help create good bug reports.

gcov is a coverage testing tool. It is used to analyze programs to find out where optimizations will have the most effect.

libgcc* contains run-time support for gcc.

libstdc++ is the standard C++ library. It contains many frequently-used functions.

libsupc++ provides supporting routines for the c++ programming language.

GCC Installation Dependencies

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

Gettext

For installation instructions see the Section called *Installing Gettext-0.12.1* in Chapter 6.

Official Download Location

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

Contents of Gettext

The Gettext package is used for internationalization and localization. Programs can be compiled with Native Language Support (NLS) which enable them to output messages in the user's native language.

Installed programs: autopoint, config.charset, config.rpath, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, project-id, team-address, trigger, urlget, user-email 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.

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 is a private library containing common routines used by the various gettext programs. They're not meant for general use.

libgettextpo is used to write specialized programs that process PO files. This library is used when the standard applications shipped with gettext won't suffice (such as msgcomm, msgcmp, msgattrib and msgen).

libgettextsrc is a private library containing common routines used by the various gettext programs. They're not meant for general use.

Gettext Installation Dependencies

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

Glibc

For installation instructions see the Section called *Installing Glibc-2.3.2* in Chapter 6.

Official Download Location

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

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

Glibc Sscanf Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/glibc-2.3.2-sscanf-1.patch`

Contents of Glibc

Glibc is the C library that provides the system calls and basic functions such as open, malloc, printf, etc. The C library is used by all dynamically linked programs.

Installed programs: catchsegv, gencat, getconf, getent, glibcbug, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nscd_nischeck, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump and zic

Installed libraries: ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd-compat.a, libc.[a,so], libc_nonshared.a, libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so 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 filesystem specific variables.

getent gets entries from an administrative database.

glibcbug creates a bug report and mails it to the bug email address.

iconv performs character set conversion.

iconvconfig creates fastloading iconv module configuration file.

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 is a Perl program that tells the compiler to enable or disable the use of POSIX locales for built-in operations.

localedef compiles locale specifications.

mtrace...

nscd is a name service cache daemon providing 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 is 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 RPC protocol.

rpcinfo makes an RPC call to an RPC server.

sln is used to make symbolic links. The program is statically linked, so it is useful for making symbolic links to dynamic libraries if the dynamic linking system for some reason is nonfunctional.

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 is the time zone dumper.

zic is the time zone compiler.

ld.so is the helper program for shared library executables.

libBrokenLocale is used by programs, such as Mozilla, to solve broken locales.

libSegFault is a segmentation fault signal handler. It tries to catch segfaults.

libanl is an asynchronous name lookup library.

libbsd-compat provides the portability needed in order to run certain BSD programs under Linux.

libc is the main C library — a collection of commonly used functions.

libcrypt is the cryptography library.

libdl is the dynamic linking interface library.

libg is a runtime library for g++.

libieee is the IEEE floating point library.

libm is the mathematical library.

libmcheck contains code run at boot.

libmemusage is used by memusage to help collect information about the memory usage of a program.

libnsl is the network services library.

libnss* are the Name Service Switch libraries, containing functions for resolving host names, user names, group names, aliases, services, protocols,and the like.

libpcprofile contains profiling functions used to track the amount of CPU time spent in which source code lines.

libpthread is the POSIX threads library.

libresolv contains functions for creating, sending, and interpreting packets to the Internet domain name servers.

librpcsvccontains 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.

Glibc Installation Dependencies

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

Grep

For installation instructions see the Section called *Installing Grep-2.5.1* in Chapter 6.

Official Download Location

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

Contents of Grep

Grep is a program used to print lines from a file matching a specified pattern.

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.

Grep Installation Dependencies

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

Groff

For installation instructions see the Section called *Installing Groff-1.19* in Chapter 6.

Official Download Location

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

Contents of Groff

The Groff package includes several text processing programs for text formatting. Groff translates standard text and special commands into formatted output, such as what you see in a manual page.

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 an EQN equation into a cropped image.

grn is a groff preprocessor for gremlin files.

grodvi is a driver for groff that produces TeX dvi format.

groff is 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.

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 makes an inverted index for the bibliographic databases 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 from the standard input a line containing a set of keywords, 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 is a simple preprocessor for groff.

neqn formats equations for ascii output.

nroff is 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.

pre-grohtml translates the output of GNU troff to html.

post-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 about 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. Usually it should 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.

Groff Installation Dependencies

Groff depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Grub

For installation instructions see the Section called *Installing Grub-0.93* in Chapter 6.

Official Download Location

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

Grub Gcc33 Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/grub-0.93-gcc33-1.patch`

Contents of Grub

The Grub package contains a bootloader.

Installed programs: grub, grub-install, grub-md5-crypt, grub-terminfo and mbchk

Short descriptions

grub is 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 used if you have an uncommon terminal.

mbchk checks the format of a multiboot kernel.

Grub Installation Dependencies

Grub depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Gzip

For installation instructions see the Section called *Installing Gzip-1.3.5* in Chapter 6.

Official Download Location

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

Contents of Gzip

The Gzip package contains programs to compress and decompress files using the Lempel-Ziv coding (LZ77).

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 is used to create self-uncompressing executable files.

gzip compresses the given files, using Lempel-Ziv (LZ77) coding.

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 recompresses files from compress format to gzip format — .Z to .gz.

Gzip Installation Dependencies

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

Inetutils

For installation instructions see the Section called *Installing Inetutils-1.4.2* in Chapter 6.

Official Download Location

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

Contents of Inetutils

The Inetutils package contains network clients and servers.

Installed programs: ftp, ping, rcp, rlogin, rsh, talk, telnet and tftp

Short descriptions

ftp is the ARPANET file transfer program.

ping sends echo-request packets and reports how long the replies take.

rcp does remote file copy.

rlogin does remote login.

rsh runs a remote shell.

talk is used to chat up another user.

telnet is an interface to the TELNET protocol.

tftp is a trivial file transfer program.

Inetutils Installation Dependencies

Inetutils depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Kbd

For installation instructions see the Section called *Installing Kbd-1.08* in Chapter 6.

Official Download Location

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

Kbd More-Programs Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/kbd-1.08-more-programs.patch`

Contents of Kbd

Kbd contains keytable files and keyboard utilities.

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 is 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).

psf* are a set of tools for handling Unicode character tables for console fonts.

resizecons changes the kernel idea of the console size.

setfont lets you change the EGA/VGA fonts on the console.

setkeycodes loads kernel scancode-to-keycode mapping table entries, useful if you have some unusual keys on your keyboard.

setleds sets the keyboard flags and LEDs. Many people find it useful to have NumLock on by default, setleds +num achieves this.

setlogcons sends kernel messages to the console.

setmetamode defines the keyboard meta key handling.

setvesablank lets you fiddle with the built-in hardware screensaver (no toasters, just a blank screen).

showconsolefont shows the current EGA/VGA console screen font.

showkey reports the scancodes and keycodes and ASCII codes of the keys pressed on the keyboard.

unicode_start puts the keyboard and console in unicode mode.

unicode_stop reverts keyboard and console from unicode mode.

Kbd Installation Dependencies

Kbd depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, Gzip, M4, Make, Sed.

Less

For installation instructions see the Section called *Installing Less-381* in Chapter 6.

Official Download Location

Less (381):
`ftp://ftp.gnu.org/gnu/less/`

Contents of Less

Less is a file pager, or text viewer. It displays the contents of a file, or stream, and has the ability to scroll. Less has a few features not included in the `more` pager, such as the ability to scroll backwards.

Installed programs: less, lessecho and lesskey

Short descriptions

less is a file viewer or pager. It displays the contents of the given file, letting you scroll around, find strings, and jump to marks.

lessecho is needed to expand metacharacters, such as * and ?, in filenames on Unix systems.

lesskey is used to specify the key bindings for less.

Less Installation Dependencies

Less depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

LFS-Bootscripts

For installation instructions see the Section called *Installing LFS-Bootscripts-1.12* in Chapter 6.

Official Download Location

LFS-Bootscripts (1.12):
`http://downloads.linuxfromscratch.org/`

Contents of LFS-bootscripts

The LFS-Bootscripts package contains SysV init style shell scripts. These scripts do various tasks such as check filesystem integrity during boot, load keymaps, set up networks and halt processes at shutdown.

Installed scripts: checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountproc, network, rc, reboot, sendsignals, setclock, swap, sysklogd and template

Short descriptions

The **checkfs** script checks the file systems just before they are mounted (with the exception of journal and network based file systems).

The **cleanfs** script removes files that shouldn't 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.

The **functions** script contains functions shared among different scripts, such as error and status checking.

The **halt** script halts the system.

The **ifdown** and **ifup** scripts assist the network script with network devices.

The **loadkeys** script loads the keymap table you specified as proper for your keyboard layout.

The **localnet** script sets up the system's hostname and local loopback device.

The **mountfs** script mounts all file systems that aren't marked noauto or aren't network based.

The **mountproc** script is used to mount the proc filesystem.

The **network** script sets up network interfaces, such as network cards, and sets up the default gateway where applicable.

The **rc** script is the master runlevel control script. It is responsible for running all the other scripts one-by-one in a specific sequence.

The **reboot** script reboots the system.

The **sendsignals** script makes sure every process is terminated before the system reboots or halts.

The **setclock** script resets the kernel clock to localtime in case the hardware clock isn't set to GMT time.

The **swap** script enables and disables swap files and partitions.

The **sysklogd** script starts and stops the system and kernel log daemons.

The **template** script is a template you can use to create your own bootscripts for your other daemons.

LFS-Bootscripts Installation Dependencies

Bzip2 depends on: Bash, Coreutils.

Lfs-Utils

For installation instructions see the Section called *Installing Lfs-Utils-0.3* in Chapter 6.

Official Download Location

Lfs-utils (0.3):
`http://www.linuxfromscratch.org/~winkie/downloads/lfs-utils/`

Contents of Lfs-Utils

The Lfs-Utils package contains some miscellaneous programs used by various packages, but are not large enough to warrant their own individual package.

Installed programs: mktemp, tempfile, http-get and iana-net

Installed files: protocols, services

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.

The **http-get** script takes advantage of a little known feature of `bash` called "net redirection". It is used to download from websites without using any other programs.

iana-net uses the `http-get` script to simplify the process of procuring IANA's services and protocols configuration files.

Lfs-Utils Installation Dependencies

(No dependencies checked yet.)

Libtool

For installation instructions see the Section called *Installing Libtool-1.5* in Chapter 6.

Official Download Location

Libtool (1.5):
`ftp://ftp.gnu.org/gnu/libtool/`

Contents of Libtool

GNU libtool is a generic library support script. Libtool hides the complexity of using shared libraries behind a consistent, portable interface.

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.

Libtool Installation Dependencies

Libtool depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Linux (the kernel)

For installation instructions see the Section called *Installing Linux-2.4.22* in Chapter 8.

Official Download Location

Linux (2.4.22):
`ftp://ftp.kernel.org/pub/linux/kernel/`

Contents of Linux

The Linux kernel is at the core of every Linux system. It's what makes Linux tick. When a computer is turned on and boots a Linux system, the very first piece of Linux software that gets loaded is the kernel. The kernel initializes the system's hardware components: serial ports, parallel ports, sound cards, network cards, IDE controllers, SCSI controllers and a lot more. In a nutshell the kernel makes the hardware available so that the software can run.

Installed files: the kernel and the kernel headers

Short descriptions

The *kernel* is the engine of your GNU/Linux system. When switching on your box, the kernel is the first part of your operating system that gets loaded. It detects and initializes all the components of your computer's hardware, then makes these components available as a tree of files to the software, and turns a single CPU into a multi-tasking machine capable of running scores of programs seemingly at the same time.

The *kernel headers* define the interface to the services that the kernel provides. The headers in your system's `include` directory should *always* be the ones against which Glibc was compiled and should therefore *not* be replaced when upgrading the kernel.

Linux Installation Dependencies

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

M4

For installation instructions see the Section called *Installing M4-1.4* in Chapter 6.

Official Download Location

M4 (1.4):
`ftp://ftp.gnu.org/gnu/m4/`

Contents of M4

M4 is a macro processor. It copies input to output, expanding macros as it goes. Macros are either built-in or user-defined and can take any number of arguments. Besides just doing macro expansion, m4 has built-in functions for including named files, running Unix commands, doing integer arithmetic, manipulating text in various ways, 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.

Installed program: M4 248

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 just doing macro expansion, m4 has built-in functions for including named files, running Unix commands, doing integer arithmetic, manipulating text in various ways, recursion, and so on. The m4 program can be used either as a front-end to a compiler or as a macro processor in its own right.

M4 Installation Dependencies

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

Make

For installation instructions see the Section called *Installing Make-3.80* in Chapter 6.

Official Download Location

Make (3.80):
`ftp://ftp.gnu.org/gnu/make/`

Contents of Make

Make determines, automatically, which pieces of a large program need to be recompiled and issues the commands to recompile them.

Installed program: Make 249

Short description

make automatically determines which pieces of a large package need to be recompiled, and then issues the relevant commands.

Make Installation Dependencies

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

MAKEDEV

For installation instructions see the Section called *Creating devices (Makedev-1.7)* in Chapter 6.

Official Download Location

MAKEDEV (1.7):
`http://downloads.linuxfromscratch.org/`

Contents of MAKEDEV

The MAKEDEV script creates the static device nodes which usually reside in the `/dev` directory. Detailed information about device nodes may be found in the `Documentation/devices.txt` file under the Linux kernel source tree.

Installed script: MAKEDEV   250

Short description

MAKEDEV is a script for creating the necessary static device nodes, usually residing in the `/dev` directory.

MAKEDEV Installation Dependencies

Make depends on: Bash, Coreutils.

Man

For installation instructions see the Section called *Installing Man-1.5m2* in Chapter 6.

Official Download Location

Man (1.5m2):
`ftp://ftp.win.tue.nl/pub/linux-local/utils/man/`

Man 80-Columns Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-80cols.patch`

Man Manpath Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-manpath.patch`

Man Pager Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/man-1.5m2-pager.patch`

Contents of Man

Man is a man pager.

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 for each page writes the name and a short description in the whatis database.

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.

Man Installation Dependencies

Man depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Man-pages

For installation instructions see the Section called *Installing Man-pages-1.60* in Chapter 6.

Official Download Location

Man-pages (1.60):
`ftp://ftp.kernel.org/pub/linux/docs/manpages/`

Contents of Man-pages

The Man-pages package contains over 1200 manual pages. This documentation details the C and C++ functions, describes a few important device files and provides documents which would otherwise be missing from other packages.

Installed files: various manual pages

Short description

Examples of provided *manual pages* are the pages describing all the C and C++ functions, important device files, and important configuration files.

Man-pages Installation Dependencies

Man depends on: Bash, Coreutils, Make.

Modutils

For installation instructions see the Section called *Installing Modutils-2.4.25* in Chapter 6.

Official Download Location

Modutils (2.4.25):
`ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils/`

Contents of Modutils

The Modutils package contains programs that you can use to work with kernel modules.

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 shows which modules are loaded.

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 the relevant modules.

rmmod unloads modules from the running kernel.

Modutils Installation Dependencies

Modutils depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Glibc, Grep, M4, Make, Sed.

Ncurses

For installation instructions see the Section called *Installing Ncurses-5.3* in Chapter 6.

Official Download Location

Ncurses (5.3):
`ftp://ftp.gnu.org/gnu/ncurses/`

Ncurses Etip Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses-5.3-etip-2.patch`

Ncurses Vsscanf Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses-5.3-vsscanf.patch`

Contents of Ncurses

The Ncurses package provides character and terminal handling libraries, including panels and menus.

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], libpanel.[a,so]

Short descriptions

captoinfo converts a termcap description into a terminfo description.

clear clears the screen, if this is 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 is the terminfo action checker. It is mainly used to test the correctness of an entry in the terminfo database.

tic is the terminfo entry-description compiler. It 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, for each giving its primary name and its description.

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.

libncurses* contain functions to display text in many complicated 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* contain functions to implement forms.

libmenu* contain functions to implement menus.

libpanel* contain functions to implement panels.

Ncurses Installation Dependencies

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

Net-tools

For installation instructions see the Section called *Installing Net-tools-1.60* in Chapter 6.

Official Download Location

Net-tools (1.60):
`http://www.tazenda.demon.co.uk/phil/net-tools/`

Net-tools Mii-Tool-Gcc33 Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/net-tools-1.60-miitool-gcc33-1.patch`

Contents of Net-tools

The Net-tools package contains a collection of programs which form the base of Linux networking.

Installed programs: arp, dnsdomainname (link to hostname), domainname (link to hostname), hostname, ifconfig, nameif, netstat, nisdomainname (link to hostname), plipconfig, rarp, route, slattach and ypdomainname (link to hostname)

Short descriptions

arp is used to manipulate the kernel's ARP cache, usually to add or delete an entry, or to dump the entire cache.

dnsdomainname reports the system's DNS domain name.

domainname reports or sets the system's NIS/YP domain name.

hostname reports or sets the name of the current host system.

ifconfig is the main utility for configuring network interfaces.

nameif names network interfaces based on MAC addresses.

netstat is used to report network connections, routing tables, and interface statistics..

nisdomainname does the same as domainname.

plipconfig is used to fine tune the PLIP device parameters, to improve its performance.

rarp is used to manipulate the kernel's RARP table.

route is used to manipulate the IP routing table.

slattach attaches a network interface to a serial line. This allows you to use normal terminal lines for point-to-point links to other computers.

ypdomainname does the same as domainname.

Net-tools Installation Dependencies

Net-tools depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make.

Patch

For installation instructions see the Section called *Installing Patch-2.5.4* in Chapter 6.

Official Download Location

Patch (2.5.4):
`ftp://ftp.gnu.org/gnu/patch/`

Contents of Patch

The patch program modifies a file according to a patch file. A patch file usually is a list, created by the diff program, that contains instructions on how an original file needs to be modified.

Installed program: Patch 255

Short description

patch modifies files according to a patch file. A patch file normally is a difference listing created with the diff program. By applying these differences to the original files, patch creates the patched versions. Using patches instead a entire new tarballs to keep your sources up-to-date can save you a lot of download time.

Patch Installation Dependencies

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

Perl

For installation instructions see the Section called *Installing Perl-5.8.0* in Chapter 6.

Official Download Location

Perl (5.8.0):
`http://www.perl.com/`

Perl Libc Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/perl-5.8.0-libc-3.patch`

Contents of Perl

The Perl package contains perl, the Practical Extraction and Report Language. Perl combines some of the best features of C, sed, awk and sh into one powerful language.

Installed programs: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (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: (too many to name)

Short descriptions

a2p translates awk to perl.

c2ph dumps C structures as generated from "cc -g -S" stabs.

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.

perlbug is 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 is the Perl Installation Verification Procedure. It can be used to verify that Perl and its libraries have been installed correctly.

piconv is a Perl version of the character encoding converter `iconv`.

pl2pm is 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 is 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.

Perl Installation Dependencies

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

Procinfo

For installation instructions see the Section called *Installing Procinfo-18* in Chapter 6.

Official Download Location

Procinfo (18):
`ftp://ftp.cistron.nl/pub/people/svm/`

Contents of Procinfo

The procinfo program gathers system data, such as memory usage and IRQ numbers, from the `/proc` directory and formats this data in a meaningful way.

Installed programs: lsdev, procinfo and socklist

Short descriptions

lsdev lists the devices present in your system, and which IRQs and IO ports they use.

procinfo displays an overview of some of the info present in the virtual proc filesystem.

socklist lists the open sockets, reporting their type, portnumber, and other specifics.

Procinfo Installation Dependencies

Procinfo depends on: Binutils, GCC, Glibc, Make, Ncurses.

Procps

For installation instructions see the Section called *Installing Procps-3.1.11* in Chapter 6.

Official Download Location

Procps (3.1.11):
`http://procps.sourceforge.net/`

Procps Locale Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/procps-3.1.11-locale-fix.patch`

Contents of Procps

The Procps package provides programs to monitor and halt system processes. Procps gathers information about processes via the `/proc` directory.

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 in the system, both physical and swap memory.

kill is used to send 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 gives a snapshot of the current 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 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 screenful of its output. This allows you to watch the output change over time.

libproc contains the functions used by most programs in this package.

Procps Installation Dependencies

Procps depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses.

Psmisc

For installation instructions see the Section called *Installing Psmisc-21.3* in Chapter 6.

Official Download Location

Psmisc (21.3):
`http://download.sourceforge.net/psmisc/`
`ftp://download.sourceforge.net/pub/sourceforge/psmisc/`

Contents of Psmisc

The Psmisc package contains three programs which help manage the `/proc` directory.

Installed programs: fuser, killall and pstree

Short descriptions

fuser reports the PIDs of processes that use the given files or filesystems.

killall kills processes by name. It sends a signal to all processes running any of the given commands.

pidof reports the PIDs of the given programs. (Not this pidof program is used, however, but the one from Sysvinit.)

pstree displays running processes as a tree.

Psmisc Installation Dependencies

Psmisc depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed.

Sed

For installation instructions see the Section called *Installing Sed-4.0.7* in Chapter 6.

Official Download Location

Sed (4.0.7):
`ftp://ftp.gnu.org/gnu/sed/`

Contents of Sed

sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline).

Installed program: Sed 260

Short description

sed is used to filter and transform text files in a single pass.

Sed Installation Dependencies

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

Shadow

For installation instructions see the Section called *Installing Shadow-4.0.3* in Chapter 6.

Official Download Location

Shadow (4.0.3):
`ftp://ftp.pld.org.pl/software/shadow/`

Shadow Newgrp Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/shadow-4.0.3-newgrp-fix.patch`

Contents of Shadow

The Shadow package was created to strengthen the security of system passwords.

Installed programs: chage, chfn, chpasswd, chsh, dpasswd, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, groups, grpck, grpconv, grpunconv, lastlog, login, logoutd, mkpasswd, newgrp, newusers, passwd, pwck, pwconv, pwunconv, sg (link to newgrp), useradd, userdel, usermod, vigr (link to vipw) and vipw

Short descriptions

chage is used to change the maximum number of days between obligatory password changes.

chfn is used to change a user's full name and some other info.

chpasswd is used to update the passwords of a whole series of user accounts in one go.

chsh is used to change a user's default login shell.

dpasswd is used to change dial-up passwords for user login shells.

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 let users sign on.

logoutd is a daemon used to enforce restrictions on log-on time and ports.

mkpasswd encrypts the given password using the also given perturbation.

newgrp is used to change the current GID during a login session.

newusers is used to create or update a whole series of user accounts in one go.

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.

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, UID, shell, initial group, home directory, and the like.

vigr can be used to edit the `/etc/group` or `/etc/gshadow` files.

vipw can be used to edit the `/etc/passwd` or `/etc/shadow` files.

libmisc...

libshadow contains functions used by most programs in this package.

Shadow Installation Dependencies

Shadow depends on: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Sysklogd

For installation instructions see the Section called *Installing Sysklogd-1.4.1* in Chapter 6.

Official Download Location

Sysklogd (1.4.1):
`http://www.infodrom.org/projects/sysklogd/`

Contents of Sysklogd

The Sysklogd package contains programs for recording system log messages, such as those reported by the kernel.

Installed programs: klogd and syslogd

Short descriptions

klogd is a system daemon for intercepting and logging kernel messages.

syslogd logs the messages that system programs offer for logging. Every logged message contains at least a date stamp and a hostname, and normally the program's name too, but that depends on how trusting the logging daemon is told to be.

Sysklogd Installation Dependencies

Sysklogd depends on: Binutils, Coreutils, GCC, Glibc, Make.

Sysvinit

For installation instructions see the Section called *Installing Sysvinit-2.85* in Chapter 6.

Official Download Location

Sysvinit (2.85):
`ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/`

Contents of Sysvinit

The Sysvinit package contains programs to control the startup, running and shutdown of all other programs.

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 flag, except when already in runlevel 0, then it tells the kernel to halt the system. But first it notes in the file `/var/log/wtmp` that the system is being brought down.

init is the mother of all processes. It reads its commands from `/etc/inittab`, which normally tell it which scripts to run for which runlevel, and how many gettys to spawn.

killall5 sends a signal to all processes, except the processes in its own session — so it won't kill the shell running the script that called it.

last shows which users last logged in (and out), searching back through the file `/var/log/wtmp`. It can also show system boots and shutdowns, and runlevel 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. But see halt.

reboot tells the kernel to reboot the system. But see halt.

runlevel reports the previous and the current runlevel, as noted in the last runlevel 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 the superuser to log in. It is normally invoked by init when the system goes into single user mode.

telinit tells init which runlevel to enter.

utmpdump displays the content of the given login file in a friendlier format.

wall writes a message to all logged-in users.

Sysvinit Installation Dependencies

Sysvinit depends on: Binutils, Coreutils, GCC, Glibc, Make.

Tar

For installation instructions see the Section called *Installing Tar-1.13.25* in Chapter 6.

Official Download Location

Tar (1.13.25):
`ftp://alpha.gnu.org/gnu/tar/`

Contents of Tar

Tar is an archiving program designed to store and extract files from an archive file known as a tar file.

Installed programs: rmt and tar

Short descriptions

rmt is used to remotely manipulate a magnetic tape drive, through an interprocess communication connection.

tar is used to create and extract files from archives, also known as tarballs.

Tar Installation Dependencies

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

Tcl

For installation instructions see the Section called *Installing Tcl-8.4.4* in Chapter 5.

Official Download Location

Tcl (8.4.4):
`http://download.sourceforge.net/tcl/`
`ftp://download.sourceforge.net/pub/sourceforge/tcl/`

Contents of Tcl

The Tcl package contains the Tool Command Language.

Installed programs: tclsh (link to tclsh8.4), tclsh8.4

Installed library: libtcl8.4.so

Short description

tclsh8.4 is the Tcl command shell.

libtcl8.4.so is the Tcl library.

Tcl Installation Dependencies

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

Texinfo

For installation instructions see the Section called *Installing Texinfo-4.6* in Chapter 6.

Official Download Location

Texinfo (4.6):
`ftp://ftp.gnu.org/gnu/texinfo/`

Contents of Texinfo

The Texinfo package contains programs used for reading, writing and converting Info documents, which provide system documentation.

Installed programs: info, infokey, install-info, makeinfo, texi2dvi and texindex

Short descriptions

info is used to read Info documents. Info documents are a bit like man pages, but often go much deeper than just explaining all the flags. Compare for example man tar and info tar.

infokey compiles a source file containing Info customizations into a binary format.

install-info is used to install Info files. It updates entries in the Info index file.

makeinfo translates the given Texinfo source documents into various other formats: Info files, plain text, or HTML.

texi2dvi is used to format the given Texinfo document into a device-independent file that can be printed.

texindex is used to sort Texinfo index files.

Texinfo Installation Dependencies

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

Util-linux

For installation instructions see the Section called *Installing Util-linux-2.12* in Chapter 6.

Official Download Location

Util-linux (2.12):
`http://ftp.cwi.nl/aeb/util-linux/`

Contents of Util-linux

The Util-linux package contains a number of miscellaneous utility programs. Some of the more prominent utilities are used to mount, unmount, format, partition and manage disk drives, open tty ports and fetch kernel messages.

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, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, pg, pivot_root, ramsize (link to rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (link to rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (link to swapon), swapon, test.bash, test.tcsh, 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 you to call block device ioctls from the command line.

cal displays a simple calender.

cfdisk is used to manipulate the partition table of the given device.

chkdupexe finds duplicate executables.

col filters out reverse line feeds.

colcrt is used to filter 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 was used to tune 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 can be used to tune the performance and interactiveness of a block device.

fdformat low-level formats a floppy disk.

fdisk could be used to manipulate the partition table of the given device.

fsck.cramfs performs a consistency check on the Cramfs filesystem on the given device.

fsck.minix performs a consistency check on the Minix filesystem 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 is used to read or set the system's hardware clock (also called the RTC or BIOS clock).

ipcrm removes the given IPC resource.

ipcs provides IPC status information.

isosize reports the size of an iso9660 filesystem.

kill terminates the specified process.

line copies a single line.

logger enters the given message into the system log.

look displays lines that begin with the given string.

losetup is used to set up and control loop devices.

mcookie generates magic cookies, 128-bit random hexadecimal numbers, for xauth.

mkfs is used to build a filesystem on a device (usually a harddisk partition).

mkfs.bfs creates an SCO bfs filesystem.

mkfs.cramfs creates a cramfs filesystem.

mkfs.minix creates a Minix filesystem.

mkswap initializes the given device or file to be used as a swap area.

more is a filter for paging through text one screenful at a time. But less is much better.

mount attaches the filesystem on the given device to the given directory in the system's file tree.

namei shows the symbolic links in the given pathnames.

pg displays a text file one screenful at a time.

pivot_root makes the given filesystem the new root filesystem of the current process.

ramsize could be used to set the size of the RAM disk in a bootable image.

rdev could be used to query and set the root device and other things in a bootable image.

readprofile reads kernel profiling information.

rename renames the given files, replacing a given string with another.

renice is used to alter the priority of running processes.

rev reverses the lines of a given file.

rootflags could be used to set the rootflags in a bootable image.

script makes a typescript of a terminal session, of everything printed to the terminal.

setfdprm sets user-provided floppy disk parameters.

setsid runs the given program in a new session.

setterm is used to set terminal attributes.

sfdisk is a disk partition table manipulator.

swapdev could be used to set 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 is used to tune the parameters of the line printer.

ul is a filter for translating underscores into escape sequences indicating underlining for the terminal in use.

umount disconnects a filesystem from the system's file tree.

vidmode could be used to set 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. That is, if that user has not disabled such messages.

Util-linux Installation Dependencies

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

Vim

For installation instructions see the Section called *Installing Vim-6.2* in Chapter 6.

Official Download Location

Vim (6.2):
`ftp://ftp.vim.org/pub/editors/vim/unix/`

Contents of Vim

The Vim package contains a configurable text editor built to enable efficient text editing.

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 is 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 is 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 can't be suspended.

rvim is a restricted version of vim: no shell commands can be started and vim can't be suspended.

shtags.pl generates a tag file for perl scripts.

tcltags generates a tag file for TCL code.

vi starts vim in vi-compatible mode.

view starts vim in read-only mode.

vim is the editor.

vim132 starts vim with the terminal in 132-column mode.

vim2html.pl converts vim documentation to 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 is a script which spells a file and generates the syntax statements necessary to highlight in vim.

vimtutor teaches you the basic keys and commands of vim.

xxd makes a hexdump of the given file. It can also do the reverse, so it can be used for binary patching.

Vim Installation Dependencies

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

Zlib

For installation instructions see the Section called *Installing Zlib-1.1.4* in Chapter 6.

Official Download Location

Zlib (1.1.4):
`http://www.gzip.org/zlib/`

Zlib Vsnprintf Patch:
`http://www.linuxfromscratch.org/patches/lfs/5.0/zlib-1.1.4-vsnprintf.patch`

Contents of Zlib

The Zlib package contains the libz library, which is used by some programs for its compression and uncompression functions.

Installed libraries: libz[a,so]

Short description

libz* contains compression and uncompression functions used by some programs.

Zlib Installation Dependencies

Zlib depends on: Binutils, Coreutils, GCC, Glibc, Make, Sed.

Appendix B
Index of programs and library files

This is a list of all the programs and library files that are installed in this book, each with a link to the package in Appendix A to which it belongs.