Cross-Compiled Linux From Scratch

Version 1.0.0rc6-x86_64-Pure64

Jim Gifford
Ryan Oliver

Cross-Compiled Linux From Scratch: Version 1.0.0rc6-x86_64-Pure64

by Jim Gifford and Ryan Oliver

Table of Contents

Preface

1. Foreword

The Linux From Scratch Project has seen many changes in the few years of its existence. I personally became involved with the project in 1999, around the time of the 2.x releases. At that time, the build process was to create static binaries with the host system, then chroot and build the final binaries on top of the static ones.

Later came the use of the /static directory to hold the initial static builds, keeping them separated from the final system, then the PureLFS process developed by Ryan Oliver and Greg Schafer, introducing a new toolchain build process that divorces even our initial builds from the host. Finally, LFS 6 bought Linux Kernel 2.6, the udev dynamic device structure, sanitized kernel headers, and other improvements to the Linux From Scratch system.

The one "flaw" in LFS is that it has always been based on an x86 class processor. With the advent of the Athlon 64 and Intel EM64T processors, the x86-only LFS is no longer ideal. Throughout this time, Ryan Oliver developed and documented a process by which you could build Linux for any system and from any system, by use of cross-compilation techniques. Thus, the Cross-Compiled LFS (CLFS) was born.

CLFS follows the same guiding principles the LFS project has always followed, e.g., knowing your system inside and out by virtue of having built the system yourself. Additionally, during a CLFS build, you will learn advanced techniques such as cross-build toolchains, multilib support (32 & 64-bit libraries side-by-side), alternative architectures such as Sparc, MIPS, and Alpha, and much more.

We hope you enjoy building your own CLFS system, and the benefits that come from a system tailored to your needs.

--
Jeremy Utley, CLFS 1.x Release Manager (Page Author)
Jim Gifford, CLFS Project Co-leader
Ryan Oliver, CLFS Project Co-leader
Joe Ciccone, Justin Knierim, Chris Staub, Matt Darcy, Ken Moffat,
Manuel Canales Esparcia, and Nathan Coulson - CLFS Developers

2. Audience

There are many reasons why somebody would want to read this book. The principal reason is to install a Linux system 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 the existence of CLFS is to help people understand how a Linux system works. Building an CLFS system helps demonstrate what makes Linux tick, and how things work together and depend on each other. One of the best things this learning experience provides is the ability to customize Linux to your own tastes and needs.

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

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

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

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

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

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

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

3. Prerequisites

Building a CLFS system is not a simple task. It requires a certain level of existing knowledge of Unix system administration in order to resolve problems, and correctly execute the commands listed. In particular, as an absolute minimum, the reader should already have the ability to use the command line (shell) to copy or move files and directories, list directory and file contents, and change the current directory. It is also expected that the reader has a reasonable knowledge of using and installing Linux software. A basic knowledge of the architectures being used in the Cross LFS process and the host operating systems in use is also required.

Because the CLFS book assumes *at least* this basic level of skill, the various CLFS support forums are unlikely to be able to provide you with much assistance. Your questions regarding such basic knowledge will likely go unanswered, or you will be referred to the CLFS essential pre-reading list.

Before building a CLFS system, we recommend reading the following HOWTOs:

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

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

- The Linux Users' Guide
  http://www.linuxhq.com/guides/LUG/guide.html

  This guide covers the usage of assorted Linux software.

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

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

4. Host System Requirements

You should be able to build a CLFS system from just about any Unix-type operating system. Your host system should have the following software with the minimum versions indicated. Also note that many distributions will place software headers into separate packages, often in the form of “[package-name]-devel” or “[package-name]-dev”. Be sure to install those if your distribution provides them.

- **Bash-2.05a**
- **Binutils-2.12** (Versions greater than 2.17 are not recommended as they have not been tested)
- **Bzip2-1.0.2**
- **Coreutils-5.0** (or Sh-Utils-2.0, Textutils-2.0, and Fileutils-4.1)
- **Diffutils-2.8**
- **Findutils-4.1.20**
- **Gawk-3.0**
- **Gcc-2.95.3** (Versions greater than 4.1.1 are not recommended as they have not been tested)
- **Glibc-2.2.5** (Versions greater than 2.4 are not recommended as they have not been tested)
- **Grep-2.5**
- **Gzip-1.2.4**
- **Make-3.79.1**
- **Patch-2.5.4**
- **Sed-3.0.2**
- **Tar-1.14**

To see whether your host system has all the appropriate versions, run the following:

```
cat > version-check.sh << "EOF"
#!/bin/bash

# Simple script to list version numbers of critical development tools

bash --version | head -n1 | cut -d" " -f2-4
echo -n "Binutils: "; ld --version | head -n1 | cut -d" " -f3-4
bzip2 --version 2>&1 < /dev/null | head -n1 | cut -d" " -f1,6-
echo -n "Coreutils: "; chown --version | head -n1 | cut -d")" -f2
diff --version | head -n1
find --version | head -n1
gawk --version | head -n1
gcc --version | head -n1
/lib/libc.so.6 | head -n1 | cut -d" " -f1-7
grep --version | head -n1
gzip --version | head -n1
make --version | head -n1
patch --version | head -n1
sed --version | head -n1
tar --version | head -n1

EOF

bash version-check.sh
```

5. Typography

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

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

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

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

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

Emphasis

This form of text is used for several purposes in the book. Its main purpose is to emphasize important points or items.

http://cross-lfs.org/

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

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

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

`[REPLACED TEXT]`

This format is used to encapsulate text that is not to be typed as seen or copied-and-pasted.

`passwd(5)`

This format is used to refer to a specific manual page (hereinafter referred to simply as a “man” page). The number inside parentheses indicates a specific section inside of **man**. For example, **passwd** has two man pages. Per CLFS installation instructions, those two man pages will be located at `/usr/share/man/man1/passwd.1` and `/usr/share/man/man5/passwd.5`. Both man pages have different information in them. When the book uses `passwd(5)` it is specifically referring to `/usr/share/man/man5/passwd.5`. **man passwd** will print the first man page it finds that matches “passwd”, which will be `/usr/share/man/man1/passwd.1`. For this example, you will need to run **man 5 passwd** in order to read the specific page being referred to. It should be noted that most man pages do not have duplicate page names in different sections. Therefore, **man *`[program name]`*** is generally sufficient.

6. Structure

This book is divided into the following parts.

6.1. Part I - Introduction

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

6.2. Part II - Preparing for the Build

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

6.3. Part III - Make the Cross-Compile Tools

Part III shows you how to make a set of Cross-Compiler tools. These tools can run on your host system but allow you to build packages that will run on your target system.

6.4. Part IV - Building the Basic Tools

Part IV explains how to build a tool chain designed to operate on your target system. These are the tools that will allow you to build a working system on your target computer.

6.5. Part V - Building the CLFS System

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

6.6. Appendices

The appendices contain information that doesn't really fit anywhere else in the book. Appendix A contains definitions of acronyms and terms used in the book; Appendix B gives acknowledgments to people who have helped work on the CLFS project and website; Appendix C and D have information about package dependencies and the the build order. Some architectures may have additional appendices for arch-specific issues.

7. Errata

The software used to create a CLFS system is constantly being updated and enhanced. Security warnings and bug fixes may become available after the CLFS book has been released. Some host systems may also have problems building CLFS. To check whether the package versions or instructions in this release of CLFS need any modifications to accommodate security vulnerabilities, other bug fixes, or host-specific issues, please visit *http://trac.cross-lfs.org/clfs/errata/1.0.0rc6/* before proceeding with your build. You should note any changes shown and apply them to the relevant section of the book as you progress with building the CLFS system.

Part I. Introduction

Chapter 1. Introduction

1.1. Cross-LFS Acknowledgements

The CLFS team would like to acknowledge people who have assisted in making the book what it is today.

Our Leaders:

- Ryan Oliver - Build Process Developer.
- Jim Gifford - Lead Developer.
- Jeremy Utley - Release Manager 1.x Series.

Our CLFS Team:

- Joe Ciccone - Alpha, MIPS, Sparc builds.
- Nathan Coulson - Bootscripts.
- Matt Darcy - x86, X86_64, and Sparc builds.
- Manuel Canales Esparcia - Book XML.
- Karen McGuiness - Proofreader.
- Jeremy Huntwork - PowerPC, x86, Sparc builds.
- Justin Knierim - Website Architect.
- Ken Moffat - PowerPC and X86_64 builds. Developer of Pure 64 Hint.
- Alexander E. Patrakov - Udev/Hotplug Integration
- Chris Staub - x86 builds. Leader of Quality Control.

Outside the Development Team

- Jürg Billeter - Testing and assisting in the development of the Linux Headers Package
- Richard Downing - Testing, typo, and content fixes.
- Peter Ennis - Typo and content fixes.
- Tony Morgan - Typo and content fixes.

The CLFS team would also like to acknowledge contributions of people from *clfs-dev@lists.cross-lfs.org* and associated mailing lists who have provided valuable technical and editorial corrections while testing the Cross-LFS book.

- G. Moko - Text updates and Typos
- Maxim Osipov - MIPS Testing.

- Doug Ronne - Various x86_64 fixes.
- William Zhou - Text updates and Typos
- Theo Schneider - Testing of the Linux Headers Package

Thank you all for your support.

1.2. How to Build a CLFS System

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

As an alternative to installing an entire separate distribution onto your machine, you may wish to use the Linux From Scratch LiveCD. This CD works well as a host system, providing all the tools you need to successfully follow the instructions in this book. It does also contain source packages and patches for the LFS book, and a copy of the LFS book, but not the needed packages or book for CLFS. You can still use the CD for building CLFS, but you will need to download the packages, patches and book separately. You can also look at *http://www.linuxfromscratch.org/hints/downloads/files/lfscd-remastering-howto.txt* for infomation on building your own CD, replacing the LFS packages and book with those for CLFS. Once you have the CD, no network connection or additional downloads are necessary. For more information about the LFS LiveCD or to download a copy, visit *http://www.linuxfromscratch.org/livecd/*.

Preparing a New Partition of this book describes how to create a new Linux native partition and file system, the place where the new CLFS system will be compiled and installed. Packages and Patches explains which packages and patches need to be downloaded to build a CLFS system and how to store them on the new file system. Final Preparations discusses the setup for an appropriate working environment. Please read Final Preparations carefully as it explains several important issues the developer should be aware of before beginning to work through Constructing Cross-Compile Tools and beyond.

Constructing Cross-Compile Tools explains the installation of cross-compile tools which will be built on the host but be able to compile programs that run on the target machine. These cross-compile tools will be used to create a temporary, minimal system that will be the basis for building the final CLFS system. Some of these packages are needed to resolve circular dependencies—for example, to compile a compiler, you need a compiler.

The process of building cross-compile tools first involves building and installing all the necessary tools to create a build system for the target machine. With these cross-compiled tools, we eliminate any dependencies on the toolchain from our host distro.

After we build our “Cross-Tools”, we start building a very minimal working system in /tools. This minimal system will be built using the cross-toolchain in /cross-tools.

In Installing Basic System Software, the full CLFS system is built. Depending on the system you are cross-compiling for, you will either boot the minimal temp-system on the target machine, or chroot into it.

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 CLFS partition. This is very similar to rebooting and instructing the kernel to mount the CLFS partition as the root partition. The major advantage is that “chrooting” allows the builder to continue using the host while CLFS is being built. While waiting for package compilation to complete, a user can switch to a different virtual console (VC) or X desktop and continue using the computer as normal.

Some systems cannot be built by chrooting so they must be booted instead. Generally, if you building for a different arch than the host system, you must reboot because the kernel will likely not support the target machine. Booting involves installing a few additional packages that are needed for bootup, installing bootscripts, and building a miminal kernel. We also describe some alternative booting methods in Section 7.23, “What to do next”

To finish the installation, the CLFS-Bootscripts are set up in Setting Up System Bootscripts, and the kernel and boot loader are set up in Making the CLFS System Bootable. The End contains information on furthering the CLFS experience beyond this book. After the steps in this book have been implemented, the computer will be ready to reboot into the new CLFS system.

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

1.3. Master Changelog

This is version 1.0.0rc6 of the Cross-Compiled Linux From Scratch book, dated September 18, 2006. If this book is more than six months old, a newer and better version is probably already available. To find out, please check one of the mirrors via *http://trac.cross-lfs.org/*.

Below is a list of detailed changes made since the previous release of the book.

Changelog Entries:

- September 18, 2006
  - [Chris] - Removed TeX installation commands from Texinfo instructions.
- September 16, 2006
  - [jim] - Updated Udev-Cross-LFS rules to 1.0-3. Fixes bootup with /lib64/udev.
- September 10, 2006
  - [jim] - Added a patch to fix Inetutils. Telnet on 64 bit systems will fail, it will attempt to connect to 255.255.255.255 instead of the ip address entered. This was due to a change in glibc's handling of inet_addr, was long now is u_int32_t. Bug found by Vladimir Vainer.
  - [jim] - Add patch to fix gzexe. Used Matts change from LFS, that utilizes Robert's fix. More details at http://wiki.linuxfromscratch.org/lfs/ticket/1876.
  - [Chris] - Added Host System Requiremets page, and added more info to the "To Boot or Chroot?" page about needing a 2.6.x Linux kernel to chroot.
- September 9, 2006
  - [jim] - Updated Linux to 2.6.17.13 and Linux Headers to 2.6.17.13-09092006.
- September 8, 2006
  - [jim] - Fixed Perl Testsuite Network tests.
- September 7, 2006
  - [jim] - Updated Multilib Tree to follow build standards set in the book.
  - [jim] - Updated copy commands in Linux Headers, simpler and cleaner.
- September 6, 2006
  - [jim] - UDEV Rules - On multilib builds, change /lib/udev to /lib64/udev, to keep consistency through the system.

    - [jim] - Removed sed from tcl, not required.
    - [jim] - Replace Perl FPIC patch with a sed, to make sure fpic is set correctly.

- September 5, 2006

    - [jim] - UDEV - On multilib builds, change /lib/udev to /lib64/udev, to keep consistency through the system.
    - [jim] - IPRoute2 - On multilib builds, change /usr/lib/tc to /usr/lib64/tc, to keep consistency through the system.

- August 28, 2006

    - [jim] - Renamed Cross-LFS Bootscripts 0.4 to 1.0, for the release. No changes made to the scripts.
    - [jim] - Renamed Cross-LFS Udev Rules 0.1-07062006 to 1.0, for the release. No changes made to the rules.

- August 27, 2006

    - [jim] - Updated to Cross-LFS Bootscripts 0.4.

- August 23, 2006

    - [jim] - Updated Linux to 2.6.17.11 and Linux Headers to 2.6.17.11-08232006.
    - [jim] - Updated Linux to 2.6.17.10 and Linux Headers to 2.6.17.10-08232006.

- August 21, 2006

    - [jciccone] - Added a multilib perl setup that has a multilib wrapper. The multilib wrapper checks for the value of PERL_ARCH and executes the cooresponding perl binary.
    - [ken] - Remove include/net from instructions for headers.

- August 20, 2006

    - [jciccone] - Added the Perl fPIC patch which makes perl build a shared DynaLoader.a.
    - [Chris] - Removed mention of package users hint from "Package Management" page.
    - [jim] - Updated Linux to 2.6.17.9 and Linux Headers to 2.6.17.9-08202006.
    - [jim] - Added Linux Tulip Patch to all Linux 2.6.17.9 builds. Fixes an initialization error.

- August 8, 2006

    - [jim] - Moved man-pages to the beginning of the build. (fixes ticket #82).

    - [jim] - Updated the wording of Bzip2, changes were made to MIPS but not the other multilib architectures. Updated for all architectures now (fixes ticket #79).

- August 7, 2006

    - [Chris] - Updated udev explanatory text in the book and added "Custom Symlinks" page, taken from LFS. Thanks to Alexander Patrakov for the updated text (fixes ticket #75).

- August 3, 2006

    - [ken] - Fixed the temporary bash to correct a failure in the Glibc testsuite, (fixes ticket #78). Thanks to Go Moko.

- July 27, 2006

    - [jim] - Fixed symlink issue in bootscripts. New Package CLFS-bootscripts 0.3.

- July 25, 2006

    - [jim] - Symlinks /tools/bin/file to /usr/bin/file for the GCC testsuite.

- July 24, 2006

    - [Chris] - Updated commands for building the keymap into the kernel - changed the bootable/kernel page to account for the new keymap files location, and changed the instructions in the "boot" section to reflect that fact that we don't know where the keymap files may be on the host (fixes ticket #56).

- July 20, 2006

    - [jim] - Updated to linux-2.6.17.6 and linux-headers-2.6.17.6-07202006.
    - [jim] - Updated to Udev 096.

- July 17, 2006

    - [jeremy] - Minor foreword fixups courtesy of Karen.

- July 16, 2006

    - [Chris] - Removed unneeded chown and chmod commands from final-system linux-headers installation.
    - [Chris] - Added a sed command to fix updatedb due to the new location for the find program.

- July 14, 2006

    - [jeremy] - Updated to new foreword

- July 11, 2006
  - [jim] - Updated to linux-headers-2.6.17.4-07112006.
- July 9, 2006
  - [jim] - Updated to linux-headers-2.6.17.4-07092006.
- July 8, 2006
  - [jim] - Updated to linux-2.6.17.4 and linux-headers-2.6.17.4-07072006.
  - [Chris] - Removed many "${CLFS}" references from explanatory text in boot section.
- July 7, 2006
  - [Chris] - Modified the "creating directories" sections - changed multilib instructions to be consistent with the new format now used for every other arch, and changed mips instructions to use xincludes.
- July 6, 2006
  - [jim] - Updated Udev rules to 07062006, Fixed cdrom, permissions and groups.
- July 5, 2006
  - [jim] - Fixes a possible issue with bootscripts if the the /usr partition is not mounted during startup. Moved find to /bin and moved some kbd utilities to /bin.
- July 4, 2006
  - [jim] - Updated the Grub patch to fix various issues, see text of patch for more details.
- July 2, 2006
  - [Chris] - Updated passwd and group file creation to only create minimal users and groups and include information on other users/groups.
  - [jim] - Added file to Testsuite tools, required for gcc tests.
- July 1, 2006
  - [jim] - Updated the build to use CLFS in variables instead of LFS. Also change /mnt/lfs to /mnt/clfs.
- June 29, 2006
  - [jim] - Updated Temp-System and Boot builds to be more consistant. Added --build=${LFS_HOST}, where possible.

    - [jim] - Updated to Cross-LFS Bootscripts. Added check for for /etc/sysconfig/createfiles.

- June 27, 2006
    - [jim] - Updated to linux-headers-2.6.17.1-06272006.
- June 25, 2006
    - [jciccone] - Updated to linux-headers-2.6.17.1-06252006.
- June 24, 2006
    - [jeremy] - Release of 1.0.0-rc1
- June 23, 2006
    - [jim] - Updated Binutils 2.17.
    - [jim] - Updated Cross-LFS Specific Packages and Patches links for Release.
- June 22, 2006
    - [jim] - Added mktemp to Linux dependency list. New dependency as of 2.6.17.
    - [jim] - Updated Vim 7.0 patch.
- June 21, 2006
    - [Chris] - Text updates to the book, including replacing "LFS" with "CLFS".
    - [jim] - Updated to linux-headers-2.6.17.1-06212006-1.
    - [jim] - Updated Vim 7.0 patch.
- June 20, 2006
    - [jeremy] - Corrected e2fsprogs installation in the boot section, to take into account the new mke2fs.conf file.
    - [jciccone] - Updated to linux-2.6.17.1 and linux-headers-2.6.17.1-06202006.
    - [jim] - Updated to linux-headers-2.6.17.1-06202006-1.
- June 19, 2006
    - [jciccone] - Updated to man-1.6d.
    - [jciccone] - Added an iconv_fix patch to glibc which fixes an issue that has shown up primarily in samba.

    - [jciccone] - Updated to linux-2.6.17 and linux-headers-2.6.17-06192006.

- June 15, 2006
    - [jim] - Let Perl use thread support.
- June 14, 2006
    - [jim] - Updated to Binutils 2.16.94.
- June 10, 2006
    - [Chris] - Updates to lists of installed programs for several packages.
    - [Ken] - Fixed the module-init-tools install in the presence of existing files. Thanks to Manuel for pointing me to the fix in LFS, and to Dan Nicholson for the fix.
- June 9, 2006
    - [jim] - Updates to Shadow 4.0.16 build instructions.
    - [jim] - Updated to Linux Headers 2.6.16.20 to 06092006.
- June 7, 2006
    - [jim] - Updated to Shadow 4.0.16.
- June 6, 2006
    - [Chris] - Added many more -v switches for verbosity.
- June 5, 2006
    - [jim] - Updated to linux-2.6.16.20, and linux-headers-2.6.16.20.
    - [jim] - Updated to Bison 2.3.
- June 4, 2006
    - [Chris] - Removed obsolete paragraph about "resetting" passwords when using pwconv from Shadow instructions.
    - [jim] - Updated to Linux Headers 2.6.16.19 to 06042006.
- June 1, 2006
    - [Chris] - Updated list of installed programs for several packages.

- May 31, 2006
  - [ken] - Removed redundant chown of /usr/share/libtool/libltdl.
  - [jim] - Updated to linux-2.6.16.19, and linux-headers-2.6.16.19.
- May 30, 2006
  - [jim] - Added a patch to fix a missing declaration of R_OK in util-linux.
  - [Chris] - Removed the long-gone swapdev program from the list of programs installed by util-linux, and several programs not installed by default from the kbd program list.
  - [jim] - Updated to E2fsprogs 1.39.
  - [jim] - Updated to Man-Pages 2.33.
- May 29, 2006
  - [jim] - Updated to Binutils 2.16.93.
  - [jim] - Updated to UDEV 093.
  - [jim] - Updated Udev Rules to 0.1-05292006.
  - [jim] - Updated Linux Headers 2.6.16.18 to 05292006.
- May 28, 2006
  - [jim] - Added a patch to Grub to prevent. Error 24: Attempt to access block outside partition.
- May 25, 2006
  - [jim] - Updated to GCC 4.1.1.
  - [jim] - Add Vim 7.0 Upstream Patches.
- May 22, 2006
  - [jim] - Updated to Coreutils 5.96.
  - [jim] - Updated linux-2.6.16.18, and linux-headers-2.6.16.18.
- May 21, 2006
  - [jciccone] - Updated to Bison-2.2, linux-2.6.16.17, and linux-headers-2.6.16.17.
- May 17, 2006
  - [Chris] - Made a number of text updates and grammar fixes and added more dependency info (bootscripts, udev-rules, tree) to Appendix C.

- May 16, 2006
  - [Chris] - Changed the temp-system and final-system package build order and a few package build instructions to account for the changed order (the rest of ticket #26).
  - [ken] - Change name of coreutils patch to match the patch.
- May 15, 2006
  - [jim] - Updated to Man-Pages 2.32.
- May 14, 2006
  - [jim] - Updated Udev Rules to 0.1-05142006.
  - [jim] - Updated Linux Headers 2.6.16.16 to 05142006.
- May 13, 2006
  - [jim] - Updated to Coreutils 5.95.
- May 12, 2006
  - [jim] - Add Vim 7.0 Upstream Patches.
- May 11, 2006
  - [jim] - Updated Linux and Linux Headers to 2.6.16.16.
  - [jim] - Add Vim 7.0 Upstream Patches.
- May 10,2006
  - [Chris] - Added more detailed dependency info and moved it to Appendices C and D. Fixes half of ticket #26.
- May 9, 2006
  - [jim] - Updated to Binutils 2.16.92.
  - [jim] - Updated to Vim 7.0.
  - [jciccone] - Updated to Linux 2.6.16.15.
- May 7, 2006
  - [jim] - Updated Linux Headers 2.6.16.14 to 05072006.
- May 6, 2006

    - [jciccone] - Updated Linux Headers 2.6.16.14 to 05062006.
- May 4, 2006
    - [jim] - Updated to Linux-2.6.16.14.
    - [jim] - Updated Linux Headers 2.6.16.13 to 05042006.
- May 3, 2006
    - [Chris] - Updated to Man-Pages 2.31.
    - [jim] - Updated to Linux-2.6.16.13.
    - [jim] - Updated to Iana-Etc 2.10.
- May 2, 2006
    - [jim] - Updated Linux Headers 2.6.16.12 to 05022006.
    - [jim] - Reverted change to bash. Needed on some architectures.
    - [jim] - Fixed a coreutils cross-compile issues on some architectures.
- May 1, 2006
    - [Chris] - Updated to Man-Pages 2.30.
    - [jim] - Updated Linux Headers 2.6.16.11 to 05012006.
    - [Chris] - Updated to Linux-2.6.16.12.
- April 30, 2006
    - [jim] - Updated Linux Headers 2.6.16.11 to 04302006.
- April 29, 2006
    - [ken] - Add asm-generic to headers which are chowned in non-multilib books.
- April 28, 2006
    - [jim] - Updated Linux Headers 2.6.16.11 to 04282006.
    - [jim] - Updated Udev Rules to 0.1-04282006.
- April 27, 2006
    - [jim] - Updated to Linux Headers 2.6.16.11 to 04272006.

- April 25, 2006
  - [jim] - Updated to Udev 091.
  - [jim] - Updated to Linux Headers 2.6.16.11 to 04262006.
- April 24, 2006
  - [ken] - Updated to iproute2-2.6.16-060323.
  - [jim] - Updated to Linux and Linux Headers 2.6.16.11.
- April 22, 2006
  - [ken] - Add example commands to test if chroot is possible, thanks to William Zhou.
- April 21, 2006
  - [jim] - Updated Linux-Headers to fix x86_64-biarch problem.
- April 20, 2006
  - [jim] - Added missing asm-generic copy to Linux-Headers.
  - [jim] - Updated Linux-Headers to include nvram.h.
- April 19, 2006
  - [jim] - Updated to Bash Patch -8.
  - [jim] - Removed Linux-Libc-Headers and replace it with our Linux-Headers package.
  - [jim] - Updated to Linux 2.6.16.9.
  - [jim] - Updated to Linux 2.6.16.8.
- April 18, 2006
  - [jim] - Renamed gcc fold_const patch to the PR #.
  - [jim] - Added GCC 4.1.0 patch PR20425. This allows searching of multilib dirs for the specs file.
- April 17, 2006
  - [Chris] - Removed sed command from temp-system bash - no longer needed for Bash 3.1.
  - [jim] - Updated to Udev 090.
  - [jim] - Updated to udev-rules 0.1-04172006.
  - [jciccone] - Added a gcc patch that fixes an optimaztion error which can result in incorrect code.

    - [jciccone] - Removed the util-linux and kernel patch that fixes the checksum calculation for sun disklabels.

- April 14, 2006
    - [jciccone] - Added a security patch to tar.
    - [jciccone] - Added a util-linux and kernel patch to fix a checksum calculation issue for sun disklabels.
- April 13, 2006
    - [jim] - Updated to GCC 4.1.0.
    - [jim] - Fixed make install-minimal in udev-rules.
    - [jim] - Updated to Linux 2.6.16.5.
- April 12, 2006
    - [jim] - Updated to Glibc 2.4.
    - [jim] - Updated to latest bash patch -7.
- April 10, 2006
    - [jim] - Changed numbering of udev-rules to use 0.1-SVNDATE.
- April 6, 2006
    - [jim] - Added install-minimal to udev rules.
    - [jim] - Updated to Udev 089.
- April 1, 2006
    - [jim] - Updated to Make 3.81.
    - [jim] - Updated to Man-Pages 2.28.
- March 28, 2006
    - [jim] - Updated to Linux 2.6.16.1.
    - [jim] - Updated to Man-Pages 2.27.
- March 27, 2006
    - [jciccone] - Updated coreutils suppress_uptime_kill_su patch to -2.
    - [jciccone] - Updated readline fixes patch to -3.patch.

    - [jciccone] - Updated bash fixes patch to -6.

- March 22, 2006

    - [jim] - Updated Udev to build floppy helper.

- March 21, 2006

    - [jim] - Updated to Udev 088.
    - [jim] - Updated to Man-Pages 2.26.
    - [jim] - Updated to IPRoute2 2.6.15-060110.

- March 20, 2006

    - [jim] - Fixed build issue with Texinfo in temp-system. Added --build.
    - [jim] - Updated to Linux 2.6.16.
    - [jim] - Updated to Shadow 4.0.15.
    - [Chris] - Removed note about GCC 2.95.3 from kernel section, as documentation has been updated in linux-2.6.16 and it now recommends GCC >= 3.2.

- March 15, 2006

    - [jim] - Changed bootscripts to CLFS-bootscripts 0.1.
    - [jim] - Updated to Psmisc 22.2.
    - [jim] - Updated to Flex 2.5.33.

- March 14, 2006

    - [jim] - Updated to Linux 2.6.16-rc6.
    - [jim] - Removal of Hotplug.
    - [jim] - Updated to Udev 087.
    - [jim] - Removal of old udev rules.
    - [jim] - Added Udev-Cross-LFS 0.1.
    - [jim] - Updated to GCC 4.0.3.
    - [jim] - Updated to File 4.17.

- March 13, 2006

    - [jciccone] - Removed pure64 bootloader warning from the top page.

- March 9, 2006
  - [Chris] - Simplified IPRoute2 instruction by removing redundant "configure" command.
- March 2, 2006
  - [Chris] - Updated to Man-Pages 2.25.
  - [Chris] - Reverted to Expect 5.43.0. 5.44.1 depends on Tk.
- February 27, 2006
  - [jim] - Updated Bash fixes patch to -5.
- February 26, 2006
  - [jim] - Updated to Expect 5.44.1.
- February 23, 2006
  - [Chris] - Simplified the "Changing Ownership" page for the boot section, and rewrote some of the text in the final "Reboot" page.
- February 21, 2006
  - [jim] - Removed Dependency of Tempfile from Bzip2.
  - [jim] - Updated Bash fixes patch to -4.
  - [jciccone] - Updated to Man-Pages 2.24
- February 14, 2006
  - [jim] - Updated to Coreutils 5.94.
- February 11, 2006
  - [Chris] - Added -v switches to commands that accept it.
- February 9, 2006
  - [jim] - Updated to Man-Pages 2.23.
- February 8, 2006
  - [jim] - Changed x86_64 to use unknown in the target triplet.
- February 7, 2006

    - [Chris] - Removed inetutils from boot section - it will be covered in the "netboot" hint.
    - [Chris] - Added section on Package Management - moved out of BLFS.
- February 6, 2006
    - [Chris] - Moved the text explaining why binutils should be the first package compiled to a more appopriate location at the first installation of binutils, and changed the wording in temp-system gettext.
    - [jim] - Updated Readline and Bash patches from Upstream.
- February 5, 2006
    - [jim] - Updated Procps 32bit build in Multilib builds to use lib64=lib. Fixed via Trac Ticket #2.
    - [jim] - Updated to Man-Pages 2.22.
- February 3, 2006
    - [jim] - Updated to Sed 4.1.5.
- February 2, 2006
    - [jim] - Updated to Perl 5.8.8.
    - [jhuntwork] - Minor textual fix to GCC.
- February 1, 2006
    - [Chris] - Added a sed substitution to man instructions to fix an error in the **makewhatis** script.
- January 31, 2006
    - [Chris] - Added -v to commands that accept them, for consistency with LFS.
- January 30, 2006
    - [jim] - Change final-system GCC, to use make bootstrap. Update provided by Ryan Oliver.
- January 29, 2006
    - [jim] - Updated Toolchain adjustment in final-system. Thank you Dan Nicholson.
- January 24, 2006
    - [jim] - Updated to Man 1.6c.
- January 23, 2006

    - [Chris] - Removed unnecessary --with-ncurses switches from temp-system bash and inetutils.
    - [Chris] - Moved zlib from temp-system to boot as it's not needed if you chroot.

- January 19, 2006

    - [Chris] - Removed patch from shadow instructions.
    - [Chris] - Added perl sprintf vulnerability patch.
    - [jim] - Updated to Man-Pages 2.21.

- January 13, 2006

    - [ken] - Moved grep ahead of libtool to avoid /tools being hardcoded into the libtool script for EGREP.
    - [ken] - Alter bison to build repeatably, from LFS.
    - [ken] - Alter gccbug to use mktemp, from LFS.
    - [ken] - Move creation of /etc/hosts within Perl instructions, for repeatability, from LFS.
    - [Chris] - Moved bootloader setup to right after "Changing Ownership" in Chapter 7, and changed several package installation instructions in that chapter.

- January 12, 2006

    - [Chris] - Moved directory and symlink creation pages to the beginning of Chapter 7.
    - [jim] - Updated to Psmisc 22.1.
    - [jim] - Updated to Man-Pages 2.20.
    - [jim] - Updated Coreutils build instructions to copy more files to /bin. These are changes for bootscripts and the new udev rules.
    - [jim] - Updated to Linux 2.6.14.6.
    - [jim] - Numerous complaints about temp-system Perl failing to build. So I moved temp perl to first package in final-system build. The failures we noted on Pure 64 builds.

- January 11, 2006

    - [jim] - Rewrote wrote zlib final instructions to include a patch the will allow build of static and shared libraries at the same time. Thanx for the recommendation Tushar Teredesai.

- January 8, 2006

    - [Chris] - Rewrote much of the description of the build process and test suite information.

- January 3, 2006

    - [jim] - Updated to Shadow 4.0.14.

- December 30, 2005

    - [Chris] - Updated package dependencies, removed explanation of "target alias" and "cross-compiling" from final-system binutils, and removed note about GRUB's testsuite failure.
    - [ken] - Fix where tree gets installed.

- December 29, 2005

    - [jim] - Added CC="gcc" to temp-system perl build.

- December 28, 2005

    - [ken] - Remove --with-x=no from expect - either we booted, or we are in chroot.
    - [ken] - Fix a failure to build glibc-headers.
    - [jim] - Updated to Linux 2.6.14.5.

- December 23, 2005

    - [jim] - Updated Shadow patch to fix linking to outside libraries.

- December 22, 2005

    - [jim] - Bash Maintainer Released Bash-001 patch. This patch fixes the following issue: There are parsing problems with compound assignments in several contexts, including as arguments to builtins like `local', `eval', and `let', and as multiple assignments in a single command.
    - [jim] - Readline Maintainer Released Readline-001 patch. This patch fixes the following issue: A problem with the readline callback interface can result in segmentation faults when using the delete-char function via a multiple-key sequence. Two consecutive calls to delete-char will crash the application calling readline.
    - [jim] - Removed halt, sync, and shutdown users. These users work, but offer a potential security threat. Thus the reason for removal.
    - [jim] - Fixed Vim symlink. Thank you LFS and Jeremy Huntwork.

- December 20, 2005

    - [jim] - Updated to M4 1.4.4.

- December 19, 2005

    - [ken] - Fix for accessing vim's documentation, from LFS.

    - [jim] - Fixed the Gettext testsuite from failing, by fixing an issue in Gawk. Thank you Greg Schaefer.

- December 18, 2005

    - [jim] - Re-arranged temp-tools. Renamed it to testsuite-tools. Moved perl and texinfo to temp-system. Moved flags from testsuite-tools to chroot and boot sections.
    - [jim] - Removed 32bit and N32 builds from temp-system, upon testing it was found that these were not needed for building the final system on a multilib capable build.
    - [jim] - Updated to Libtool 1.5.22.
    - [jim] - Updated to Man-Pages 2.18.

- December 17, 2005

    - [jim] - Updated to psmisc 21.9.

- December 16, 2005

    - [jim] - Moved Tree to temp-tools, since it's not needed for the final-system.
    - [jim] - Moved Procps before perl in the final-system build. Fixes test suite issue in perl.

- December 15, 2005

    - [ken] - Minor fix for rendering in temp-tools/texinfo.
    - [jim] - Added not to temp-system about the WARNING message.

- December 14, 2005

    - [jim] - Change the CLFS_HOST and CLFS_TARGET pages to prevent further build issues.
    - [jim] - Updated to Man-Pages 2.17.
    - [jim] - Added Tree 1.5.0. Added for the udev testsuite.

- December 13, 2005

    - [jim] - Removed setvbuf test from Temp-System Bash compile. Thanks to David Fix.

- December 12, 2005

    - [jim] - Removed testsuites from Temp-Tools section.

- December 11, 2005

    - [ken] - Fix Tcl configure, thanks to Greg Schafer.

- December 9, 2005
  - [jim] - Upgraded to Module Init Tools 3.2.2.
  - [ken] - Bzip2 - clarify what is being tested.
  - [jim] - Removed installation of sound sanitized headers. They will be removed in the next release of linux-libc-headers.
- December 8, 2005
  - [jim] - Upgraded to Less 394.
  - [jim] - Upgraded to Bash 3.1.
  - [jim] - Upgraded to Readline 5.1.
- December 7, 2005
  - [jim] - Upgraded to TCL 8.4.12.
  - [jim] - Upgraded to IPRoute2 2.6.14-051107.
  - [jim] - Upgraded to Readline 5.0.5.
  - [jim] - Upgraded to Bash 3.0.16.
- December 6, 2005
  - [jim] - Upgraded to Glibc 20051107.
  - [jim] - Sparcv8 now uses the same Glibc as the other architectures.
  - [jim] - Ported Gettext update from Temp-system LFS to Cross-LFS.
  - [ken] - Tidying in Gettext to reflect our differences from LFS, and correct the date in changelog.
  - [jim] - Moved Cross-tools from $HOME to LFS. This provide a easy way for livecd builders to not loose valueable build time, if something goes wrong.
  - [jim] - Upgraded to Findutils 4.2.27.
- December 5, 2005
  - [Matt Darcy] - Updated the resources page to include reference and base instructions on using the LFS pastebin
  - [Matt Darcy] - Updated some of the missing text poritions of the book
  - [Matt Darcy] - Updated creation of $CLFSHOME parameter to a less user error system. This change will will need to be validated by by other devs
  - [ken] - Moved Mktemp ahead of Module-init-tools so that the latter's testsuite doesn't bail out when testing generate-modprobe.conf.

- December 3, 2005
  - [ken] - Reference a home directory for dummy user when testing coreutils, and correct sed 'man page' to 'HTML documentation', both from LFS.
- December 2, 2005
  - [jim] - Updated to Man-Pages 2.16.
- December 1, 2005
  - [jim] - Updated to Linux 2.6.14.3.
  - [jim] - Updated to Man-Pages 2.15.
  - [jim] - Updated to Udev 076.
  - [jim] - Added support for Sparc v8 and below.
- November 30, 2005
  - [jim] - Text updates to final-system.
  - [jim] - Text updates to temp-tools.
- November 29, 2005
  - [jim] - Text updates to chroot.
  - [jim] - Text updates to boot.
- November 28, 2005
  - [jim] - Text updates to glibc in all sections.
  - [jim] - Text updates to temp-system.
  - [jim] - Fixed duplicated gcc configure line in gcc. Thank you Dennis Perkins.
- November 27, 2005
  - [jim] - Text updates to gcc in all sections.
  - [jim] - Updates to Ncurses, added curses file and link. Thank you G. Moko.
- November 25, 2005
  - [jim] - Text updates to binutils in all sections.
- November 24, 2005

    - [Matt Darcy] - Text updates to resources page
    - [Matt Darcy] - Updated reference to LFS news server to reflect that it no longer exists
- November 23, 2005
    - [jim] - For inetutils to utilize the Ncurses headers in /tools/include.
- November 21, 2005
    - [jim] - Updated to Udev 075.
    - [jim] - Updated to Module Init Tools 3.2.1.
- November 20, 2005
    - [jim] - Updated to Findutils 4.2.26.
- November 18, 2005
    - [manuel] - Fixed the unpack of the module-init-tools-testsuite package.
- November 17, 2005
    - [jim] - Update to Man-Pages 2.14.
    - [jim] - Update to Linux 2.6.14.2. This includes changes for the MIPS Architecture. New Patch added.
    - [jim] - Added change to move /usr/bin/less to /bin/less.
- November 13, 2005
    - [manuel] - Improve the heuristic for determining a locale that is supported by both Glibc and packages outside LFS. Ported from LFS-SVN.
    - [manuel] - Omit running Bzip2's testsuite as a separate step, as make runs it automatically. Ported from LFS-SVN.
    - [jim] - Updated TCL build to install headers. Thank you Greg Schafer.
    - [jim] - Updated Expect to use the newly relocated TCL Headers. Thank you Greg Schafer.
- November 10, 2005
    - [ken] - Added Data/Dumper to temp perl modules, for coreutils testsuite. Ported from LFS-SVN.
    - [ken] - Removed POSIX VERSION information from coreutils page Thank you G Moko.
    - [jim] - Added text for multilib kernel build requirements.

- November 9, 2005
  - [manuel] - Stop Udev from killing udevd processes on the system and removed udevdir=/dev. Ported from LFS-SVN.
  - [manuel] - Install the binaries from Less to /usr/bin instead of /bin. Ported from LFS-SVN.
  - [manuel] - Removed SBUs and disk usage information.
  - [jim] - Added missing utmp group. Thank you William Zhou.
- November 8, 2005
  - [jim] - Removed make headers -C bfd from Binutils. Thank you Robert Day.
- November 7, 2005
  - [manuel] - Remove the optimization related warnings from the toolchain packages. Ported from LFS-SVN.
  - [manuel] - Install Vim's documentation to `/usr/share/doc/vim-7.0` instead of `/usr/share/vim/vim64/doc`. Ported from LFS-SVN.
  - [manuel] - Correct the instructions for running the Module-Init-Tools' testsuite. Ported from LFS-SVN.
  - [jim] - Removed unnecessary linking libc sed in gcc-final in cross-tools. Recommendation from Erik-Jan, via cross-lfs list.
- November 6, 2005
  - [jim] - Updated to Coreutils 5.93.
  - [jim] - Updated to Procps 3.2.6.
  - [jim] - Updated to Man-Pages 2.13.
- November 5, 2005
  - [jim] - Updated to psmisc 21.8.
- November 3, 2005
  - [jim] - Fixes added to coreutils. Coreutils was install locales to /locale. Added fixes for dircolors and md5sum also.
- October 31, 2005
  - [jim] - Now an official project. Resetting all Changelogs.

Branch Synced from the Release of LFS 6.0 on February 23rd, 2005

1.4. Changelog for x86_64-64

Below is a list of changes specifics for this architecture made since the previous release of the book. For general changes see Master Changelog,

Changelog Entries:

- August 3, 2006
  - [ken] - Fix the testsuite sed for gettext. This fixes ticket #80, thanks to Jonathan Davis for the initial report and Go Moko for the initial analysis.
- March 12, 2006
  - [ken] - Use Lilo, because the grub shell segfaults.
- December 13, 2005
  - [ken] - Reinstate lib64 symlink for ld testsuite erroneously removed in an earlier clean-up.
- November 20, 2005
  - [jim] - Add Patch to allow Grub to build in a Pure 64 environment. Great Work Joe Ciccone!!!.
- November 14, 2005
  - [ken] - Add lib64 symlink for ld testsuite. Thanks, Klaus Dimde.
- October 31, 2005
  - [jim] - Now an official project. Resetting all Changelogs.

1.5. Resources

1.5.1. FAQ

If during the building of the CLFS system you encounter any errors, have any questions, or think there is a typo in the book, please start by consulting the Frequently Asked Questions (FAQ) that is located at *http://trac.cross-lfs.org/wiki/faq*.

1.5.2. Mailing Lists

The `cross-lfs.org` server hosts a number of mailing lists used for the development of the CLFS project. These lists include the main development and support lists, among others. If the FAQ does not contain your answer, you can search the CLFS lists via The Mail Archive *http://www.mail-archive.com*. You can find the mail lists with the following link:

http://www.mail-archive.com/index.php?hunt=clfs

For information on the different lists, how to subscribe, archive locations, and additional information, visit *http://trac.cross-lfs.org/wiki/lists*.

1.5.3. News Server

Cross-LFS does not maintain it's own News Server, but we do provide access via `gmane.org` *http://gmane.org*. If you want to subscribe to the Cross-LFS via a newsreader you can utilize `gmane.org`. You can find the game search for CLFS with the following link:

http://dir.gmane.org/search.php?match=clfs

1.5.4. IRC

Several members of the CLFS community offer assistance on our community Internet Relay Chat (IRC) network. Before using this support, please make sure that your question is not already answered in the CLFS FAQ or the mailing list archives. You can find the IRC network at `chat.freenode.net`. The support channel for cross-lfs is named #cross-lfs. If you need to show people the output of your problems, please use http://pastebin.cross-lfs.org and reference the pastebin URL when asking your questions.

1.5.5. Mirror Sites

The CLFS project has a number of world-wide mirrors to make accessing the website and downloading the required packages more convenient. Please visit the CLFS website at *http://trac.cross-lfs.org/wiki/mirrors* for mirrors of CLFS.

1.5.6. Contact Information

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

1.6. Help

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

We also have a wonderful CLFS community that is willing to offer assistance through the mailing lists and IRC (see the Section 1.5, “Resources” section of this book). However, we get several support questions everyday and many of them can be easily answered by going to the FAQ and by searching the mailing lists first. So for us to offer the best assistance possible, you need to do some research on your own first. This allows us to focus on the more unusual support needs. If your searches do not produce a solution, please include all relevant information (mentioned below) in your request for help.

1.6.1. Things to Mention

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

- The version of the book being used (in this case 1.0.0rc6)
- The host distribution and version being used to create CLFS.
- The architecture of the host and target.
- The value of the ${CLFS_HOST}, ${CLFS_TARGET}, ${BUILD32}, and ${BUILD64} environment variables.
- The package or section in which the problem was encountered.
- The exact error message or symptom received. See Section 1.6.3, “Compilation Problems” below for an example.
- Note whether you have deviated from the book at all. A package version change or even a minor change to any command is considered deviation.

Note

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

1.6.2. Configure Script Problems

If something goes wrong while running the **configure** script, review the `config.log` file. This file may contain the errors you encountered during **configure**. It often logs errors that may have not been printed to the screen. Include only the *relevant* lines if you need to ask for help.

1.6.3. Compilation Problems

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

```
gcc -DALIASPATH=\"/mnt/clfs/usr/share/locale:.\"
-DLOCALEDIR=\"/mnt/clfs/usr/share/locale\"
-DLIBDIR=\"/mnt/clfs/usr/lib\"
-DINCLUDEDIR=\"/mnt/clfs/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':
/clfs/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 `/clfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/clfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2
```

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

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

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

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

Part II. Preparing for the Build

Chapter 2. Preparing a New Partition

2.1. Introduction

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

2.2. Creating a New Partition

Like most other operating systems, CLFS is usually installed on a dedicated partition. The recommended approach to building a CLFS system is to use an available empty partition or, if you have enough unpartitioned space, to create one. However, if your building for a different architecture you can simply build everything in "/mnt/clfs" and transfer it to your target machine.

A minimal system requires around 1.5 gigabytes (GB). This is enough to store all the source tarballs and compile the packages. However, if the CLFS system is intended to be the primary Linux system, additional software will probably be installed which will require additional space (2-3 GB). The CLFS system itself will not take up this much room. A large portion of this requirement is to provide sufficient free temporary storage. Compiling packages can require a lot of disk space which will be reclaimed after the package is installed.

Because there is not always enough Random Access Memory (RAM) available for compilation processes, it is a good idea to use a small disk partition as swap space. This is used by the kernel to store seldom-used data and leave more memory available for active processes. The swap partition for an CLFS system can be the same as the one used by the host system, in which case it is not necessary to create another one.

Start a disk partitioning program such as **cfdisk** or **fdisk** with a command line option naming the hard disk on which the new partition will be created—for example `/dev/hda` for the primary Integrated Drive Electronics (IDE) disk. Create a Linux native partition and a swap partition, if needed. Please refer to `cfdisk(8)` or `fdisk(8)` if you do not yet know how to use the programs.

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

2.3. Creating a File System on the Partition

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

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

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

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

Note

Some host distributions use custom features in their filesystem creation tools (E2fsprogs). This can cause problems when booting into your new CLFS system, as those features will not be supported by the CLFS-installed E2fsprogs; you will get an error similar to `unsupported filesystem features, upgrade your e2fsprogs`. To check if your host system uses custom enhancements, run the following command:

```
debugfs -R feature /dev/[xxx]
```

If the output contains features other than: dir_index; filetype; large_file; resize_inode or sparse_super then your host system may have custom enhancements. In that case, to avoid later problems, you should compile the stock E2fsprogs package and use the resulting binaries to re-create the filesystem on your CLFS partition:

```
cd /tmp
tar xjf /path/to/sources/e2fsprogs-1.39.tar.bz2
cd e2fsprogs-1.39
mkdir build
cd build
../configure
make #note that we intentionally don't 'make install' here!
./misc/mke2fs /dev/[xxx]
cd /tmp
rm -rf e2fsprogs-1.39
```

If a swap partition was created, it will need to be initialized for use by issuing the command below. If you are using an existing swap partition, there is no need to format it.

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

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

2.4. Mounting the New Partition

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

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

```
export CLFS=/mnt/clfs
```

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

```
mkdir -pv ${CLFS}
mount -v /dev/[xxx] ${CLFS}
```

Replace *`[xxx]`* with the designation of the CLFS partition.

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

```
mkdir -pv ${CLFS}
mount -v /dev/[xxx] ${CLFS}
mkdir -v ${CLFS}/usr
mount -v /dev/[yyy] ${CLFS}/usr
```

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

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

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

Chapter 3. Packages and Patches

3.1. Introduction

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

Download locations may not always be accessible. If a download location has changed since this book was published, Google (*http://www.google.com/*) provides a useful search engine for most packages. If this search is unsuccessful, try one of the alternative means of downloading discussed at *http://cross-lfs.org/files/packages/1.0.0rc6/*.

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

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

```
mkdir -v ${CLFS}/sources
```

Make this directory writable and sticky. When a directory is marked “sticky”, that means that even if multiple users have write permission on that directory, any file within that directory can only be deleted or modified by its owner. The following command will enable the write and sticky modes:

```
chmod -v a+wt ${CLFS}/sources
```

3.2. All Packages

Download or otherwise obtain the following packages:

- Autoconf (2.59) - 904 KB:
  Home page: *http://www.gnu.org/software/autoconf/*
  Download: *http://ftp.gnu.org/gnu/autoconf/autoconf-2.59.tar.bz2*
  MD5 sum: `1ee40f7a676b3cfdc0e3f7cd81551b5f`
- Automake (1.9.6) - 748 KB:
  Home page: *http://www.gnu.org/software/automake/*
  Download: *http://ftp.gnu.org/gnu/automake/automake-1.9.6.tar.bz2*
  MD5 sum: `c11b8100bb311492d8220378fd8bf9e0`
- Bash (3.1) - 2,475 KB:
  Home page: *http://www.gnu.org/software/bash/*
  Download: *http://ftp.gnu.org/gnu/bash/bash-3.1.tar.gz*
  MD5 sum: `ef5304c4b22aaa5088972c792ed45d72`
- Bash Documentation (3.1) - 2,013 KB:
  Home page:
  Download: *http://ftp.gnu.org/gnu/bash/bash-doc-3.1.tar.gz*
  MD5 sum: `a8c517c6a7b21b8b855190399c5935ae`
- Binutils (2.17) - 13,496 KB:
  Home page: *http://sources.redhat.com/binutils/*
  Download: *http://ftp.gnu.org/gnu/binutils/binutils-2.17.tar.bz2*
  MD5 sum: `e26e2e06b6e4bf3acf1dc8688a94c0d1`
- Bison (2.3) - 1,060 KB:
  Home page: *http://www.gnu.org/software/bison/*
  Download: *http://ftp.gnu.org/gnu/bison/bison-2.3.tar.bz2*
  MD5 sum: `c18640c6ec31a169d351e3117ecce3ec`
- Bzip2 (1.0.3) - 654 KB:
  Home page: *http://www.bzip.org/*
  Download: *http://www.bzip.org/1.0.3/bzip2-1.0.3.tar.gz*
  MD5 sum: `8a716bebecb6e647d2e8a29ea5d8447f`
- CLFS-Bootscripts (1.0) - 28 KB:
  Home page:
  Download: *http://cross-lfs.org/files/packages/1.0.0rc6/bootscripts-cross-lfs-1.0.tar.bz2*
  MD5 sum: `79bff3247411589d7de51fdaea9578bd`
- Coreutils (5.96) - 4,960 KB:
  Home page: *http://www.gnu.org/software/coreutils/*
  Download: *http://ftp.gnu.org/gnu/coreutils/coreutils-5.96.tar.bz2*
  MD5 sum: `bf55d069d82128fd754a090ce8b5acff`

- DejaGNU (1.4.4) - 1,056 KB:
  Home page: *http://www.gnu.org/software/dejagnu/*
  Download: *http://ftp.gnu.org/gnu/dejagnu/dejagnu-1.4.4.tar.gz*
  MD5 sum: `053f18fd5d00873de365413cab17a666`
- Diffutils (2.8.7) - 1,034 KB:
  Home page: *http://www.gnu.org/software/diffutils/*
  Download: *ftp://alpha.gnu.org/gnu/diffutils/diffutils-2.8.7.tar.gz*
  MD5 sum: `18d6587cb915e7aa110a5d463d6ed156`
- E2fsprogs (1.39) - 3,612 KB:
  Home page: *http://e2fsprogs.sourceforge.net/*
  Download: *http://prdownloads.sourceforge.net/e2fsprogs/e2fsprogs-1.39.tar.gz?download*
  MD5 sum: `06f7806782e357797fad1d34b7ced0c6`
- Expect (5.43.0) - 514 KB:
  Home page: *http://expect.nist.gov*
  Download: *http://expect.nist.gov/src/expect-5.43.0.tar.gz*
  MD5 sum: `43e1dc0e0bc9492cf2e1a6f59f276bc3`
- File (4.17) - 544 KB:
  Home page:
  Download: *ftp://ftp.gw.com/mirrors/pub/unix/file/file-4.17.tar.gz*
  MD5 sum: `50919c65e0181423d66bb25d7fe7b0fd`

Note

File (4.17) may no longer be available at the listed location. The site administrators of the master download location occasionally remove older versions when new ones are released. An alternative download location that may have the correct version available is *http://cross-lfs.org/files/packages/1.0.0rc6/*.

- Findutils (4.2.27) - 1,097 KB:
  Home page: *http://www.gnu.org/software/findutils/*
  Download: *http://ftp.gnu.org/gnu/findutils/findutils-4.2.27.tar.gz*
  MD5 sum: `f1e0ddf09f28f8102ff3b90f3b5bc920`
- Flex (2.5.33) - 680 KB:
  Home page: *http://flex.sourceforge.net*
  Download: *http://prdownloads.sourceforge.net/flex/flex-2.5.33.tar.bz2?download*
  MD5 sum: `343374a00b38d9e39d1158b71af37150`
- Gawk (3.1.5) - 1,716 KB:
  Home page: *http://www.gnu.org/software/gawk/*
  Download: *http://ftp.gnu.org/gnu/gawk/gawk-3.1.5.tar.bz2*
  MD5 sum: `5703f72d0eea1d463f735aad8222655f`
- GCC (4.1.1) - 38,300 KB:
  Home page: *http://gcc.gnu.org/*

Download: *http://ftp.gnu.org/gnu/gcc/gcc-4.1.1/gcc-4.1.1.tar.bz2*
MD5 sum: `ad9f97a4d04982ccf4fd67cb464879f3`

- Gettext (0.14.5) - 6,940 KB:
  Home page: *http://www.gnu.org/software/gettext/*
  Download: *http://ftp.gnu.org/gnu/gettext/gettext-0.14.5.tar.gz*
  MD5 sum: `e2f6581626a22a0de66dce1d81d00de3`

- Glibc (2.4) - 14,847 KB:
  Home page: *http://www.gnu.org/software/libc/*
  Download: *http://ftp.gnu.org/gnu/glibc/glibc-2.4.tar.bz2*
  MD5 sum: `7e9a88dcd41fbc53801dbe5bdacaf245`

- Grep (2.5.1a) - 516 KB:
  Home page: *http://www.gnu.org/software/grep/*
  Download: *http://ftp.gnu.org/gnu/grep/grep-2.5.1a.tar.bz2*
  MD5 sum: `52202fe462770fa6be1bb667bd6cf30c`

- Groff (1.19.2) - 2,836 KB:
  Home page: *http://www.gnu.org/software/groff/*
  Download: *http://ftp.gnu.org/gnu/groff/groff-1.19.2.tar.gz*
  MD5 sum: `f7c9cf2e4b9967d3af167d7c9fadaae4`

- Gzip (1.3.5) - 324 KB:
  Home page: *http://www.gzip.org/*
  Download: *ftp://alpha.gnu.org/gnu/gzip/gzip-1.3.5.tar.gz*
  MD5 sum: `3d6c191dfd2bf307014b421c12dc8469`

- Iana-Etc (2.10) - 184 KB:
  Home page: *http://www.sethwklein.net/projects/iana-etc/*
  Download: *http://www.sethwklein.net/projects/iana-etc/downloads/iana-etc-2.10.tar.bz2*
  MD5 sum: `53dea53262b281322143c744ca60ffbb`

- Inetutils (1.4.2) - 1019 KB:
  Home page: *http://www.gnu.org/software/inetutils/*
  Download: *http://ftp.gnu.org/gnu/inetutils/inetutils-1.4.2.tar.gz*
  MD5 sum: `df0909a586ddac2b7a0d62795eea4206`

- IPRoute2 (2.6.16-060323) - 378 KB:
  Home page: *http://linux-net.osdl.org/index.php/Iproute2*
  Download: *http://developer.osdl.org/dev/iproute2/download/iproute2-2.6.16-060323.tar.gz*
  MD5 sum: `f31d4516b35bbfeaa72c762f5959e97c`

- Kbd (1.12) - 618 KB:
  Home page:
  Download: *http://www.kernel.org/pub/linux/utils/kbd/kbd-1.12.tar.bz2*
  MD5 sum: `069d1175b4891343b107a8ac2b4a39f6`

- Less (394) - 286 KB:
  Home page: *http://www.greenwoodsoftware.com/less/*

Download: *http://www.greenwoodsoftware.com/less/less-394.tar.gz*
MD5 sum: `a9f072ccefa0d315b325f3e9cdbd4b97`

- Libtool (1.5.22) - 2,854 KB:
  Home page: *http://www.gnu.org/software/libtool/*
  Download: *http://ftp.gnu.org/gnu/libtool/libtool-1.5.22.tar.gz*
  MD5 sum: `8e0ac9797b62ba4dcc8a2fb7936412b0`

- Linux (2.6.17.13) - 40,320 KB:
  Home page: *http://www.kernel.org/*
  Download: *http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.17.13.tar.bz2*
  MD5 sum: `834885b3ad9988b966570bee92459572`

- Linux-Headers (2.6.17.13-09092006) - 1,576 KB:
  Home page: *http://headers.cross-lfs.org/*
  Download: *http://cross-lfs.org/files/packages/1.0.0rc6/linux-headers-2.6.17.13-09092006.tar.bz2*
  MD5 sum: `bfda0e9440dd76e6e35fdce79c9b0bf6`

- M4 (1.4.4) - 300 KB:
  Home page: *http://www.gnu.org/software/m4/*
  Download: *http://ftp.gnu.org/gnu/m4/m4-1.4.4.tar.bz2*
  MD5 sum: `eb93bfbcb12cf00165583302bb31a822`

- Make (3.81) - 1,125 KB:
  Home page: *http://www.gnu.org/software/make/*
  Download: *http://ftp.gnu.org/gnu/make/make-3.81.tar.bz2*
  MD5 sum: `354853e0b2da90c527e35aabb8d6f1e6`

- Man (1.6d) - 268 KB:
  Home page: *http://primates.ximian.com/~flucifredi/man/*
  Download: *http://primates.ximian.com/~flucifredi/man/man-1.6d.tar.gz*
  MD5 sum: `36d3f65bcc10f0754a3234e00d92ad6d`

- Man-pages (2.33) - 1,752 KB:
  Home page:
  Download: *http://www.kernel.org/pub/linux/docs/manpages/man-pages-2.33.tar.bz2*
  MD5 sum: `e9f61ec73b5390c582530da173c12b10`

- Mktemp (1.5) - 69 KB:
  Home page: *http://www.mktemp.org/*
  Download: *ftp://ftp.mktemp.org/pub/mktemp/mktemp-1.5.tar.gz*
  MD5 sum: `9a35c59502a228c6ce2be025fc6e3ff2`

- Module-Init-Tools (3.2.2) - 166 KB:
  Home page:
  Download: *http://www.kerneltools.org/pub/downloads/module-init-tools/module-init-tools-3.2.2.tar.bz2*
  MD5 sum: `a1ad0a09d3231673f70d631f3f5040e9`

- Ncurses (5.5) - 2,260 KB:
  Home page: *http://dickey.his.com/ncurses/*

Download: *ftp://invisible-island.net/ncurses/ncurses-5.5.tar.gz*
MD5 sum: `e73c1ac10b4bfc46db43b2ddfd6244ef`

- Patch (2.5.9) - 198 KB:
  Home page: *http://www.gnu.org/software/patch/*
  Download: *ftp://alpha.gnu.org/gnu/diffutils/patch-2.5.9.tar.gz*
  MD5 sum: `dacfb618082f8d3a2194601193cf8716`

- Perl (5.8.8) - 9,887 KB:
  Home page: *http://www.perl.com/*
  Download: *http://ftp.funet.fi/pub/CPAN/src/perl-5.8.8.tar.bz2*
  MD5 sum: `a377c0c67ab43fd96eeec29ce19e8382`

- Procps (3.2.6) - 273 KB:
  Home page: *http://procps.sourceforge.net/*
  Download: *http://procps.sourceforge.net/procps-3.2.6.tar.gz*
  MD5 sum: `7ce39ea27d7b3da0e8ad74dd41d06783`

- Psmisc (22.2) - 239 KB:
  Home page: *http://psmisc.sourceforge.net/*
  Download: *http://prdownloads.sourceforge.net/psmisc/psmisc-22.2.tar.gz?download*
  MD5 sum: `77737c817a40ef2c160a7194b5b64337`

- Readline (5.1) - 1,983 KB:
  Home page: *http://cnswww.cns.cwru.edu/php/chet/readline/rltop.html*
  Download: *http://ftp.gnu.org/gnu/readline/readline-5.1.tar.gz*
  MD5 sum: `7ee5a692db88b30ca48927a13fd60e46`

- Sed (4.1.5) - 781 KB:
  Home page: *http://www.gnu.org/software/sed/*
  Download: *http://ftp.gnu.org/gnu/sed/sed-4.1.5.tar.gz*
  MD5 sum: `7a1cbbbb3341287308e140bd4834c3ba`

- Shadow (4.0.16) - 1,412 KB:
  Home page:
  Download: *ftp://ftp.pld.org.pl/software/shadow/shadow-4.0.16.tar.bz2*
  MD5 sum: `1d91f7479143d1d705b94180c0d4874b`

Note

Shadow (4.0.16) may no longer be available at the listed location. The site administrators of the master download location occasionally remove older versions when new ones are released. An alternative download location that may have the correct version available is *http://cross-lfs.org/files/packages/1.0.0rc6/*.

- Sysklogd (1.4.1) - 80 KB:
  Home page: *http://www.infodrom.org/projects/sysklogd/*
  Download: *http://www.infodrom.org/projects/sysklogd/download/sysklogd-1.4.1.tar.gz*
  MD5 sum: `d214aa40beabf7bdb0c9b3c64432c774`

- Sysvinit (2.86) - 97 KB:
  Home page:
  Download: *ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/sysvinit-2.86.tar.gz*
  MD5 sum: `7d5d61c026122ab791ac04c8a84db967`

- Tar (1.15.1) - 1,574 KB:
  Home page: *http://www.gnu.org/software/tar/*
  Download: *http://ftp.gnu.org/gnu/tar/tar-1.15.1.tar.bz2*
  MD5 sum: `57da3c38f8e06589699548a34d5a5d07`

- Tcl (8.4.12) - 3,419 KB:
  Home page: *http://tcl.sourceforge.net/*
  Download: *http://prdownloads.sourceforge.net/tcl/tcl8.4.12-src.tar.gz?download*
  MD5 sum: `7480432d8730263f267952788eb4839b`

- Texinfo (4.8) - 1,487 KB:
  Home page: *http://www.gnu.org/software/texinfo/*
  Download: *http://ftp.gnu.org/gnu/texinfo/texinfo-4.8.tar.bz2*
  MD5 sum: `6ba369bbfe4afaa56122e65b3ee3a68c`

- Tree (1.5.0) - 26 KB:
  Home page:
  Download: *ftp://mama.indstate.edu/linux/tree/tree-1.5.0.tgz*
  MD5 sum: `e0d090c564e7ea5afa16bac80620c7e0`

- Udev (096) - 192 KB:
  Home page: *http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html*
  Download: *http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-096.tar.bz2*
  MD5 sum: `f4effef7807ce1dc91ab581686ef197b`

- Udev Cross-LFS Rules (1.0-3) - 12 KB:
  Home page:
  Download: *http://cross-lfs.org/files/packages/1.0.0rc6/udev-cross-lfs-1.0-3.tar.bz2*
  MD5 sum: `0c9b9e24a37b9501bcd4889da71cf313`

- Util-linux (2.12r) - 1,339 KB:
  Home page:
  Download: *http://www.kernel.org/pub/linux/utils/util-linux/util-linux-2.12r.tar.bz2*
  MD5 sum: `af9d9e03038481fbf79ea3ac33f116f9`

- Vim (7.0) - 6,422 KB:
  Home page: *http://www.vim.org*
  Download: *ftp://ftp.vim.org/pub/vim/unix/vim-7.0.tar.bz2*
  MD5 sum: `4ca69757678272f718b1041c810d82d8`

- Vim (7.0) language files (optional) - 1,153 KB:
  Home page:
  Download: *ftp://ftp.vim.org/pub/vim/extra/vim-7.0-lang.tar.gz*
  MD5 sum: `6d43efaff570b5c86e76b833ea0c6a04`

• Zlib (1.2.3) - 485 KB:
Home page: *http://www.zlib.net/*
Download: *http://www.zlib.net/zlib-1.2.3.tar.gz*
MD5 sum: `debc62758716a169df9f62e6ab2bc634`

Total size of these packages: about

3.3. Additional Packages for x86_64

- Bin86 (0.16.17) - 149 KB:
  Home page: *http://freshmeat.net/projects/bin86/*
  Download: *http://homepage.ntlworld.com/robert.debath/dev86/bin86-0.16.17.tar.gz*
  MD5 sum: `c9e8d72dd2e7457b52d0e3164fc199a1`
- Lilo (22.7.1) - 420 KB:
  Home page:
  Download: *ftp://ftp.metalab.unc.edu/pub/Linux/system/boot/lilo/lilo-22.7.1.src.tar.gz*
  MD5 sum: `1f3855c6f2c7b2beaa8a90bf9975a289`

Total size of these packages: about

3.4. Needed Patches

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

- Bash Fixes Patch - 23 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/bash-3.1-fixes-8.patch*
  MD5 sum: `bc337045fa4c5839babf0306cc9df6d0`
- Binutils Posix Patch - 4.9 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/binutils-2.17-posix-1.patch*
  MD5 sum: `7e42a8edc0c59246bbc58c428256113c`
- Bzip2 Documentation Patch - 1.7 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/bzip2-1.0.3-install_docs-1.patch
  MD5 sum: `9e5dfbf4814b71ef986b872c9af84488`
- Bzip2 Bzdiff Remove Tempfile - 1.8 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/bzip2-1.0.3-remove_tempfile-1.patch
  MD5 sum: `bcadb0ce282c96af15a86a2ccdac0765`
- Bzip2 Bzgrep Security Fixes Patch - 1.3 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/bzip2-1.0.3-bzgrep_security-1.patch
  MD5 sum: `4eae50e4fd690498f23d3057dfad7066`
- Coreutils Suppress Uptime, Kill, Su Patch - 13 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/coreutils-5.96-suppress_uptime_kill_su-1.pa
  MD5 sum: `227d41a6d0f13c31375153eae91e913d`
- Expect Spawn Patch - 6.9 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/expect-5.43.0-spawn-2.patch
  MD5 sum: `7706e1e8238c72eed8dc905d6f3b6aa9`
- Gawk Segfault Patch - 1.3 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/gawk-3.1.5-segfault_fix-1.patch
  MD5 sum: `7679530d88bf3eb56c42eb6aba342ddb`
- GCC Cross Search Paths Patch - 2 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/gcc-4.1.1-cross_search_paths-1.patch
  MD5 sum: `541fe39d228ddaa0d8396a35ec3a0ada`
- GCC PR20425 Patch - 36 KB:

Download:
http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/gcc-4.1.1-PR20425-1.patch
MD5 sum: `95535bda8e4d37d30251db0b121b5374`

- GCC Posix Patch - 9 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/gcc-4.1.1-posix-1.patch*
  MD5 sum: `0d88068740a0e00780891f2cb905b808`

- Glibc iconv Fix - 4 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/glibc-2.4-iconv_fix-1.patch*
  MD5 sum: `9c8e681226ccf7a1f25c6467674f915e`

- Glibc Disable linking with libgcc_eh.a - 1 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/glibc-2.4-libgcc_eh-1.patch
  MD5 sum: `e5122ea7b89a5f22615eaadf8e46b334`

- Glibc Localedef Segfault - 1.9 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/glibc-2.4-localedef_segfault-1.patch
  MD5 sum: `42452abc6196789e0a83afa1ca7e6e4e`

- Gzip Security Fix Patch - 2 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/gzip-1.3.5-security_fixes-1.patch
  MD5 sum: `f107844f01fc49446654ae4a8f8a0728`

- Inetutils inet_addr Patch - 4 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/inetutils-1.4.2-inet_addr_fix-1.patch
  MD5 sum: `a33267b7a4e0d303a6f8dfeafde7bd8e`

- Inetutils GCC 4.x Fixes Patch - 4.1 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/inetutils-1.4.2-gcc4_fixes-3.patch
  MD5 sum: `5204fbc503c9fb6a8e353583818db6b9`

- Inetutils No-Server-Man-Pages Patch - 1.3 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/inetutils-1.4.2-no_server_man_pages-1.patch
  MD5 sum: `eb477f532bc6d26e7025fcfc4452511d`

- KBD GCC 4.x Fixes Patch - 1.5 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/kbd-1.12-gcc4_fixes-1.patch
  MD5 sum: `615bc1e381ab646f04d8045751ed1f69`

- Linux Tulip Update Patch - 8 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/linux-2.6.17.13-tulip-1.patch
  MD5 sum: `0dd7027a8cb8e59c74c24ff0a8f45f3b`

- Mktemp Tempfile Patch - 3.6 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/mktemp-1.5-add_tempfile-3.patch
  MD5 sum: `65d73faabe3f637ad79853b460d30a19`
- Perl Libc Patch - 4 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/perl-5.8.8-libc-2.patch*
  MD5 sum: `3bf8aef1fb6eb6110405e699e4141f99`
- Readline Fixes Patch - 2.1 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/readline-5.1-fixes-3.patch*
  MD5 sum: `e30963cd5c6f6a11a23344af36cfa38c`
- Sysklogd Fixes Patch - 28 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/sysklogd-1.4.1-fixes-1.patch
  MD5 sum: `508104f058d1aef26b3bc8059821935f`
- Tar GCC-4.x Fix Patch - 1.2 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/tar-1.15.1-gcc4_fix_tests-1.patch
  MD5 sum: `8e286a1394e6bcf2907f13801770a72a`
- Tar Security Fix Patch - 3.9 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/tar-1.15.1-security_fixes-1.patch
  MD5 sum: `19876e726d9cec9ce1508e3af74dc22e`
- Tar Sparse Fix Patch - 1 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/tar-1.15.1-sparse_fix-1.patch
  MD5 sum: `9e3623f7c88d8766878ecb27c980d86a`
- Texinfo Tempfile Fix Patch - 2.2 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/texinfo-4.8-tempfile_fix-2.patch
  MD5 sum: `559bda136a2ac7777ecb67511227af85`
- Util-linux Cramfs Patch - 2.8 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/util-linux-2.12r-cramfs-1.patch
  MD5 sum: `1c3f40b30e12738eb7b66a35b7374572`
- Util-linux GCC 4.x Patch - 1 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/util-linux-2.12r-gcc4_fixes-1.patch
  MD5 sum: `6c030921dc9b92daf688f12a4ee6f6e0`
- Util-linux Missing Header Patch - 1 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/util-linux-2.12r-missing_header-1.patch

MD5 sum: `33ccc15d2e92caa6189b044f573fdcda`

- Vim Fixes Patch - 32 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/vim-7.0-fixes-5.patch*
  MD5 sum: `6e179cfe811d105de4fd9156a0ef6699`

- Zlib fPIC Patch - 3.2 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/zlib-1.2.3-fPIC-1.patch*
  MD5 sum: `545d60b20bfde6f53023de44438cef59`

Total size of these patches: about

In addition to the above required patches, there exist a number of optional patches created by the CLFS community. These optional patches solve minor problems or enable functionality that is not enabled by default. Feel free to peruse the patches database located at *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/* and acquire any additional patches to suit the system needs.

3.5. Additional Patches for x86_64

- Bin86 x86_64 Patch - 2 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/bin86-0.16.17-x86_64-1.patch
  MD5 sum: `92bdce7b0655cd2e9f83c83fc56d128e`
- Coreutils Uname Patch - 4.5 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/coreutils-5.96-uname-1.patch
  MD5 sum: `c05b735710fbd62239588c07084852a0`
- GCC Pure 64 Patch - 15 KB:
  Download: *http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/gcc-4.1.1-pure64-1.patch*
  MD5 sum: `cea9bf46663392d627de81e2456698e3`
- GCC Specs Patch - 6.4 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/gcc-4.1.1-pure64_specs-1.patch
  MD5 sum: `99e0ae890fce0614be210e83f0a5b975`
- Lilo x86_64 Cross Compile Patch - 1 KB:
  Download:
  http://svn.cross-lfs.org/svn/repos/cross-lfs/branches/1.0.0rc6/patches/lilo-22.7.1-cross_compile_x86_64-1.patch
  MD5 sum: `00487cebd84e9958dddb3adf357babeb`

Total size of these patches: about

Chapter 4. Final Preparations

4.1. About ${CLFS}

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

```
echo ${CLFS}
```

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

```
export CLFS=/mnt/clfs
```

Having this variable set is beneficial in that commands such as **install -dv ${CLFS}/tools** can be typed literally. The shell will automatically replace "${CLFS}" with "/mnt/clfs" (or whatever the variable was set to) when it processes the command line.

If you haven't created the `${CLFS}` directory, do so at this time by issuing the following commands:

```
install -dv ${CLFS}
```

Do not forget to check that `${CLFS}` is set whenever you leave and reenter the current working environment (as when doing a "su" to `root` or another user).

4.2. Creating the ${CLFS}/tools Directory

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

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

```
install -dv ${CLFS}/tools
```

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

```
ln -sv ${CLFS}/tools /
```

Note

The above command is correct. The **ln** command has a few syntactic variations, so be sure to check **info coreutils ln** and `ln(1)` before reporting what you may think is an error.

The created symlink enables the toolchain to be compiled so that it always refers to `/tools`, meaning that the compiler, assembler, and linker will work. This will provide a common place for our temporary tools system.

4.3. Creating the ${CLFS}/cross-tools Directory

The cross-binutils and cross-compiler built in Constructing Cross-Compile Tools will be installed under `${CLFS}/cross-tools` to keep them separate from the host programs. The programs compiled here are cross-tools and will not be a part of the final CLFS system or the temp-system. By keeping these programs in a separate directory, they can easily be discarded later after their use.

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

```
install -dv ${CLFS}/cross-tools
```

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

```
ln -sv ${CLFS}/cross-tools /
```

The symlink isn't technically necessary (though the book's instructions do assume its existence), but is there mainly for consistency (because /tools is also symlinked to ${CLFS}/tools) and to simplify the installation of the cross-compile tools.

4.4. Adding the CLFS User

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

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

The meaning of the command line options:

`-s /bin/bash`
This makes **bash** the default shell for user `clfs`.

`-g clfs`
This option adds user `clfs` to group `clfs`.

`-m`
This creates a home directory for `clfs`.

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

`clfs`
This is the actual name for the created group and user.

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

```
passwd clfs
```

Grant `clfs` full access to `${CLFS}/cross-tools` and `${CLFS}/tools` by making `clfs` the directorys' owner:

```
chown -v clfs ${CLFS}/tools
chown -v clfs ${CLFS}/cross-tools
```

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

```
chown -v clfs ${CLFS}/sources
```

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

```
su - clfs
```

The "`-`" instructs **su** to start a login shell as opposed to a non-login shell. The difference between these two types of shells can be found in detail in `bash(1)` and **info bash**.

4.5. Setting Up the Environment

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

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

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

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

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

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

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

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

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

By putting `/cross-tools/bin` at the beginning of the `PATH`, the cross-compiler built in Constructing Cross-Compile Tools will be picked up by the build process for the temp-system packages before anything that may be installed on the host. This, combined with turning off hashing, helps to ensure that you will be using the cross-compile tools to build the temp-system in /tools.

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

```
source ~/.bash_profile
```

4.6. About the Test Suites

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

It is not possible to run testsuites when cross-compiling, so package installation instructions do not explain how to run testsuites until Installing Basic System Software.

Part III. Make the Cross-Compile Tools

Chapter 5. Constructing Cross-Compile Tools

5.1. Introduction

This chapter shows you how to create cross platform tools.

If for some reason you have to stop and come back later, remember to use the **su - clfs** command, and it will setup the build environment that you left.

5.1.1. Common Notes

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

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

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

> **Important**
>
> After installing each package, both in this and the next chapters, delete its source and build directories, unless specifically instructed otherwise. Deleting the sources prevents mis-configuration when the same package is reinstalled later.

5.2. Build CFLAGS

`CFLAGS` and `CXXFLAGS` must not be set during the building of cross-tools.

To disable `CFLAGS` and `CXXFLAGS` use the following commands:

```
unset CFLAGS
unset CXXFLAGS
```

Now add these to `~/.bashrc`, just in case you have to exit and restart building later:

```
echo unset CFLAGS >> ~/.bashrc
echo unset CXXFLAGS >> ~/.bashrc
```

5.3. Build Flags

We will need to setup target specific flags for the compiler and linkers.

```
export BUILD64="-m64"
```

Lets add the build flags to `~/.bashrc` to prevent issues if we stop and come back later.

```
echo export BUILD64=\""${BUILD64}\"" >> ~/.bashrc
```

5.4. Build Variables

Setting Host and Target

During the building of the cross-compile tools you will need to set a few variables that will be dependent on your particular needs. The first variable will be the triplet of the `HOST` machine. You will need to set the `CHOST` triplet to match your particular needs. To set this information you can issue the following command:

```
export CLFS_HOST="$(echo $MACHTYPE | sed "s/$(echo $MACHTYPE | cut -d- -f2)/cro
```

Now we will set our Target Triplet:

```
export CLFS_TARGET="x86_64-unknown-linux-gnu"
```

Copy settings to Environment

Now we will add these to `~/.bashrc`, just in case you have to exit and restart building later:

```
echo export CLFS_HOST=\""${CLFS_HOST}\"" >> ~/.bashrc
echo export CLFS_TARGET=\""${CLFS_TARGET}\"" >> ~/.bashrc
```

5.5. Linux-Headers-2.6.17.13-09092006

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

5.5.1. Installation of Linux-Headers

Install the header files that are needed for the base build:

```
install -dv /tools/include
cp -av include/asm-x86_64 /tools/include/asm
cp -av include/{asm-generic,linux} /tools/include/
```

Details on this package are located in Section 10.5.2, “Contents of Linux-Headers.”

5.6. Cross Binutils-2.17

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

5.6.1. Installation of Cross Binutils

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

To make sure that the proper syntax is used for a couple of tools, apply the following patch:

```
patch -Np1 -i ../binutils-2.17-posix-1.patch
```

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

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

Prepare Binutils for compilation:

```
../binutils-2.17/configure --prefix=/cross-tools \
   --host=${CLFS_HOST} --target=${CLFS_TARGET} --with-lib-path=/tools/lib \
   --disable-nls --enable-shared --enable-64-bit-bfd --disable-multilib
```

The meaning of the configure options:

`--prefix=/cross-tools`
This tells the configure script to prepare to install the package in the `/cross-tools` directory.

`--host=${CLFS_HOST}`
When used with --target, this creates a cross-architecture executable that creates files for ${CLFS_TARGET} but runs on ${CLFS_HOST}.

`--target=${CLFS_TARGET}`
When used with --host, this creates a cross-architecture executable that creates files for ${CLFS_TARGET} but runs on ${CLFS_HOST}.

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

`--disable-nls`
This disables internationalization as i18n is not needed for the cross-compile tools.

`--enable-shared`
Enable the creation of the shared libraries.

`--disable-multilib`
This option disables the building of a multilib capable binutils.

--enable-64-bit-bfd
This adds 64 bit support to Binutils.

Compile the package:

```
make configure-host
make
```

The meaning of the make options:

configure-host
This checks the host environment and makes sure all the necessary tools are available to compile Binutils.

Install the package:

```
make install
```

Copy the `libiberty.h` file to `/tools/include` directory:

```
cp -v ../binutils-2.17/include/libiberty.h /tools/include
```

Details on this package are located in Section 10.8.2, “Contents of Binutils.”

5.7. Cross GCC-4.1.1 - Static

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

5.7.1. Installation of Cross GCC Compiler with Static libgcc and no Threads

Make a couple of essential adjustments to the specs file to ensure GCC uses our build environment:

```
patch -Np1 -i ../gcc-4.1.1-pure64_specs-1.patch
```

To make sure that a couple of tools use the proper syntax, apply the following patch:

```
patch -Np1 -i ../gcc-4.1.1-posix-1.patch
```

The following patch ensures that **gcc** does not search the `/usr` directory for `libgcc_s.so` when cross-compiling:

```
patch -Np1 -i ../gcc-4.1.1-cross_search_paths-1.patch
```

Change the StartFile Spec to point to the correct library location:

```
echo "
#undef STARTFILE_PREFIX_SPEC
#define STARTFILE_PREFIX_SPEC \"/tools/lib/\"" >> gcc/config/linux.h
```

Now alter **gcc**'s c preprocessor's default include search path to use `/tools` only:

```
cp -v gcc/Makefile.in{,.orig}
sed -e "s@\(^CROSS_SYSTEM_HEADER_DIR =\).*@\1 /tools/include@g" \
    gcc/Makefile.in.orig > gcc/Makefile.in
```

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

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

Prepare GCC for compilation:

```
../gcc-4.1.1/configure --prefix=/cross-tools \
    --host=${CLFS_HOST} --target=${CLFS_TARGET} --disable-multilib \
    --with-local-prefix=/tools --disable-nls --disable-shared \
    --disable-threads --enable-languages=c
```

The meaning of the configure options:

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

`--disable-shared`
Disables the creation of the shared libraries.

`--disable-threads`
This will prevent GCC from looking for the multi-thread include files, since they haven't been created for this architecture yet. GCC will be able to find the multi-thread information after the Glibc headers are created.

`--enable-languages=c`
This option ensures that only the C compiler is built.

Continue with compiling the package:

```
make all-gcc
```

Install the package:

```
make install-gcc
```

Details on this package are located in Section 10.9.2, “Contents of GCC.”

5.8. Glibc-2.4

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

5.8.1. Installation of Glibc

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

Disable linking to `libgcc_eh`:

```
patch -Np1 -i ../glibc-2.4-libgcc_eh-1.patch
```

The following patch fixes an issue that can cause **localdef** to segfault:

```
patch -Np1 -i ../glibc-2.4-localedef_segfault-1.patch
```

The following **sed** fixes a build issue with Glibc. This will prevent **nscd** from trying to link to libraries that don't exist:

```
cp -v nscd/Makefile{,.orig}
sed -e "/nscd_stat.o: sysincludes = # nothing/d" nscd/Makefile.orig > \
    nscd/Makefile
```

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

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

The following lines need to be added to `config.cache` for Glibc to support NPTL:

```
echo "libc_cv_forced_unwind=yes" > config.cache
echo "libc_cv_c_cleanup=yes" >> config.cache
```

Prepare Glibc for compilation:

```
BUILD_CC="gcc" CC="${CLFS_TARGET}-gcc ${BUILD64}" \
    AR="${CLFS_TARGET}-ar" RANLIB="${CLFS_TARGET}-ranlib" \
    ../glibc-2.4/configure --prefix=/tools \
    --host=${CLFS_TARGET} --build=${CLFS_HOST} \
    --disable-profile --enable-add-ons \
    --with-tls --enable-kernel=2.6.0 --with-__thread \
    --with-binutils=/cross-tools/bin --with-headers=/tools/include \
    --cache-file=config.cache
```

The meaning of the new configure options:

`BUILD_CC="gcc"`

This sets Glibc to use the current compiler on our system. This is used to create the tools Glibc uses during its build.

CC="${CLFS_TARGET}-gcc ${BUILD64}"
Forces Glibc to build using our target architecture GCC utilizing the 64 Bit flags.

AR="${CLFS_TARGET}-ar"
This forces Glibc to use the **ar** utility we made for our target architecture.

RANLIB="${CLFS_TARGET}-ranlib"
This forces Glibc to use the **ranlib** utility we made for our target architecture.

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

--enable-add-ons
This tells Glibc to utilize all add-ons that are available.

--with-tls
This tells Glibc to use Thread Local Storage.

--with-__thread
This tells Glibc to use use the __thread for libc and libpthread builds.

--with-binutils=/cross-tools/bin
This tells Glibc to use the Binutils that are specific to our target architecture.

--cache-file=config.cache
This tells Glibc to utilize a premade cache file.

During this stage the following warning might appear:

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

The missing or incompatible **msgfmt** program is generally harmless. This **msgfmt** program is part of the Gettext package which the host distribution should provide.

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.6.5, “Contents of Glibc.”

5.9. GCC-4.1.1 - Cross Compiler Final

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

5.9.1. Installation of GCC Cross Compiler

The following patch fixes the searching of multilib dirs for specs file. The patch standardizes the gcc drivers path iteration functions, ensuring multilib directories are searched in the correct order. This fixes various issues, most noticably with libtool on multilib systems:

```
patch -Np1 -i ../gcc-4.1.1-PR20425-1.patch
```

Make a couple of essential adjustments to the specs file to ensure GCC uses our build environment:

```
patch -Np1 -i ../gcc-4.1.1-pure64_specs-1.patch
```

To make sure that a couple of tools use the proper syntax, apply the following patch:

```
patch -Np1 -i ../gcc-4.1.1-posix-1.patch
```

The following patch ensures that **gcc** does not search the `/usr` directory for `libgcc_s.so` when cross-compiling:

```
patch -Np1 -i ../gcc-4.1.1-cross_search_paths-1.patch
```

Change the StartFile Spec to point to the correct library location:

```
echo "
#undef STARTFILE_PREFIX_SPEC
#define STARTFILE_PREFIX_SPEC \"/tools/lib/\"" >> gcc/config/linux.h
```

Now alter **gcc**'s c preprocessor's default include search path to use `/tools` only:

```
cp -v gcc/Makefile.in{,.orig}
sed -e "s@\(^CROSS_SYSTEM_HEADER_DIR =\).*@\1 /tools/include@g" \
    gcc/Makefile.in.orig > gcc/Makefile.in
```

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

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

Prepare GCC for compilation:

```
../gcc-4.1.1/configure --prefix=/cross-tools \
    --target=${CLFS_TARGET} --host=${CLFS_HOST} --disable-multilib \
    --with-local-prefix=/tools --disable-nls --enable-shared \
    --enable-languages=c,c++ --enable-__cxa_atexit \
    --enable-c99 --enable-long-long --enable-threads=posix
```

The meaning of the new configure options:

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

--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-c99
Enable C99 support for C programs.

--enable-long-long
Enables long long support in the compiler.

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

Continue with compiling the package:

```
make AS_FOR_TARGET="${CLFS_TARGET}-as" \
    LD_FOR_TARGET="${CLFS_TARGET}-ld"
```

Install the package:

```
make install
```

Details on this package are located in Section 10.9.2, “Contents of GCC.”

Part IV. Building the Basic Tools

Chapter 6. Constructing a Temporary System

6.1. Introduction

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

The tools in this chapter are cross-compiled using the toolchain in /cross-tools and will be installed under the `${CLFS}/tools` directory to keep them separate from the files installed in Installing Basic System Software and the host production directories. Since the packages compiled here are temporary, we do not want them to pollute the soon-to-be CLFS system.

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

```
echo ${CLFS}
```

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

During this section of the build you will see several WARNING messages like the one below. It is safe to ignore these messages.

```
configure: WARNING: If you wanted to set the --build type, don't use --hos
    If a cross compiler is detected then cross compile mode will be used.
```

6.2. Build Variables

Setup target-specific variables for the compiler and linkers:

```
export CC="${CLFS_TARGET}-gcc"
export CXX="${CLFS_TARGET}-g++"
export AR="${CLFS_TARGET}-ar"
export AS="${CLFS_TARGET}-as"
export RANLIB="${CLFS_TARGET}-ranlib"
export LD="${CLFS_TARGET}-ld"
export STRIP="${CLFS_TARGET}-strip"
```

Then add the build variables to `~/.bashrc` to prevent issues if you stop and come back later:

```
echo export CC=\""${CC}\"" >> ~/.bashrc
echo export CXX=\""${CXX}\"" >> ~/.bashrc
echo export AR=\""${AR}\"" >> ~/.bashrc
echo export AS=\""${AS}\"" >> ~/.bashrc
echo export RANLIB=\""${RANLIB}\"" >> ~/.bashrc
echo export LD=\""${LD}\"" >> ~/.bashrc
echo export STRIP=\""${STRIP}\"" >> ~/.bashrc
```

6.3. Binutils-2.17

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

6.3.1. Installation of Binutils

To make sure that the proper syntax is used for a couple of tools, apply the following patch:

```
patch -Np1 -i ../binutils-2.17-posix-1.patch
```

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

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

Prepare Binutils for compilation:

```
CC="${CC} ${BUILD64}" \
    ../binutils-2.17/configure --prefix=/tools \
    --build=${CLFS_HOST} --host=${CLFS_TARGET} --target=${CLFS_TARGET} \
    --disable-nls --enable-shared --enable-64-bit-bfd \
    --disable-multilib
```

The meaning of the new configure options:

`CC="${CC} ${BUILD64}"`
Tells the compiler to use our 64-bit build flags.

Compile the package:

```
make configure-host
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.8.2, “Contents of Binutils.”

6.4. GCC-4.1.1

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

6.4.1. Installation of GCC

The following patch fixes the searching of multilib dirs for specs file. The patch standardizes the gcc drivers path iteration functions, ensuring multilib directories are searched in the correct order. This fixes various issues, most noticably with libtool on multilib systems:

```
patch -Np1 -i ../gcc-4.1.1-PR20425-1.patch
```

Make a couple of essential adjustments to the specs file to ensure GCC uses our build environment:

```
patch -Np1 -i ../gcc-4.1.1-pure64_specs-1.patch
```

To make sure that a couple of tools use the proper syntax, apply the following patch:

```
patch -Np1 -i ../gcc-4.1.1-posix-1.patch
```

The following patch ensures that **gcc** does not search the `/usr` directory for `libgcc_s.so` when cross-compiling:

```
patch -Np1 -i ../gcc-4.1.1-cross_search_paths-1.patch
```

Now we will change **cpp**'s search path not to look in `/usr/include`:

```
cp -v gcc/cppdefault.c{,.orig}
sed -e '/#define STANDARD_INCLUDE_DIR/s@"/usr/include"@0@g' \
    gcc/cppdefault.c.orig > gcc/cppdefault.c
```

Also, we need to set the directory searched by the fixincludes process for system headers, so it won't look at the host's headers:

```
cp -v gcc/Makefile.in{,.orig}
sed -e 's@\(^NATIVE_SYSTEM_HEADER_DIR =\).*@\1 /tools/include@g' \
    gcc/Makefile.in.orig > gcc/Makefile.in
```

When searching for the multilibs, force the build to use the results of `--print-multi-lib` from our cross-compiler, not the native compiler **gcc** builds now:

```
cp -v gcc/Makefile.in{,.orig2}
sed -e "/MULTILIBS/s@\$(GCC_FOR_TARGET)@/cross-tools/bin/${CC}@g" \
   gcc/Makefile.in.orig2 > gcc/Makefile.in
```

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

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

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

Prepare GCC for compilation:

```
CC="${CC} ${BUILD64}" CXX="${CXX} ${BUILD64}" \
    ../gcc-4.1.1/configure --prefix=/tools --disable-multilib \
    --build=${CLFS_HOST} --host=${CLFS_TARGET} --target=${CLFS_TARGET} \
    --libexecdir=/tools/lib --with-local-prefix=/tools --enable-long-long \
    --enable-c99 --enable-shared --enable-threads=posix --enable-__cxa_atexit \
    --disable-nls --enable-languages=c,c++ --disable-libstdcxx-pch
```

The meaning of the new configure options:

`CXX="${CXX} ${BUILD64}"`
This forces the C++ compiler to use our 64 Bit flags.

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

Compile the package:

```
make AS_FOR_TARGET="${AS}" \
    LD_FOR_TARGET="${LD}"
```

Install the package:

```
make install
```

Now we copy the files that are placed in `/tools/lib64` to `/tools/lib`. We also delete the /tools/lib64 directory:

```
cp -va /tools/lib64/* /tools/lib
rm -rvf /tools/lib64
```

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

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

Details on this package are located in Section 10.9.2, “Contents of GCC.”

6.5. Ncurses-5.5

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

6.5.1. Installation of Ncurses

Prepare Ncurses for compilation:

```
CC="${CC} ${BUILD64}" CXX="${CXX} ${BUILD64}" \
   ./configure --prefix=/tools --with-shared --build=${CLFS_HOST} \
   --host=${CLFS_TARGET} --without-debug --without-ada \
   --enable-overwrite --with-build-cc=gcc
```

The meaning of the configure options:

`--with-shared`
This tells Ncurses to create a shared library.

`--without-debug`
This tells Ncurses not to build with debug information.

`--without-ada`
This ensures that Ncurses does not build support for the Ada compiler which may be present on the host but will not be available when building the final system.

`--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.

`--with-build-cc=gcc`
This tells Ncurses what type of compiler we are using.

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.14.2, “Contents of Ncurses.”

6.6. Bash-3.1

The Bash package contains the Bourne-Again SHell.

6.6.1. Installation of Bash

Remove a test that causes the build to fail:

```
echo "ac_cv_func_setvbuf_reversed=no" >> config.cache
```

When bash is cross-compiled, it cannot test for the presence of named pipes. If you used **su** to become an unprivileged user, this combination will cause Bash to build without *process substitution*, which will break one of the c++ test scripts in `glibc`. Prevent that by forcing the following definition:

```
echo "bash_cv_sys_named_pipes=yes" >> config.cache
```

Prepare Bash for compilation:

```
CC="${CC} ${BUILD64}" CXX="${CXX} ${BUILD64}" \
   ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET} \
   --without-bash-malloc --cache-file=config.cache
```

The meaning of the configure option:

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

Compile the package:

```
make
```

Install the package:

```
make install
```

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

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

Details on this package are located in Section 10.25.2, “Contents of Bash.”

6.7. Bzip2-1.0.3

The Bzip2 package contains programs for compressing and decompressing files. Compressing text files with **bzip2** yields a much better compression percentage than with the traditional **gzip**.

6.7.1. Installation of Bzip2

Bzip2's default Makefile target automatically runs the testsuite as well. Disable the tests since they won't work on a multi-architecture build:

```
cp -v Makefile{,.orig}
sed -e 's@^\(all:.*\) test@\1@g' Makefile.orig > Makefile
```

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

```
make CC="${CC} ${BUILD64}" AR="${AR}" RANLIB="${RANLIB}"
```

Install the package:

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

Details on this package are located in Section 10.26.2, “Contents of Bzip2.”

6.8. Coreutils-5.96

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

6.8.1. Installation of Coreutils

Coreutils has some issues when cross-compiling. So we define the items cross-compiling doesn't like:

```
echo "ac_cv_sys_restartable_syscalls=yes" > config.cache
echo "ac_cv_func_setvbuf_reversed=yes" >> config.cache
echo "utils_cv_sys_open_max=1024" >> config.cache
```

Prepare Coreutils for compilation:

```
CC="${CC} ${BUILD64}" \
   ./configure --prefix=/tools --cache-file=config.cache \
   --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.10.2, “Contents of Coreutils.”

6.9. Diffutils-2.8.7

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

6.9.1. Installation of Diffutils

Prepare Diffutils for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
  --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.27.2, “Contents of Diffutils.”

6.10. Findutils-4.2.27

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

6.10.1. Installation of Findutils

Findutils has an issue with cross-compiling where it can't find getline. Fix it here:

```
echo "am_cv_func_working_getline=yes" >> config.cache
```

Prepare Findutils for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET} --cache-file=config.cache
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.30.2, “Contents of Findutils.”

6.11. Gawk-3.1.5

The Gawk package contains programs for manipulating text files.

6.11.1. Installation of Gawk

Prepare Gawk for compilation:

```
CC="${CC} ${BUILD64}" \
   ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.31.2, “Contents of Gawk.”

6.12. Gettext-0.14.5

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

6.12.1. Installation of Gettext

Gettext has an issue with cross-compiling where it can't find getline. Fix it here:

```
cd gettext-tools
echo "am_cv_func_working_getline=yes" >> config.cache
```

Prepare Gettext for compilation:

```
CC="${CC} ${BUILD64}" CXX="${CXX} ${BUILD64}" \
   ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET} --disable-shared \
   --cache-file=config.cache
```

The meaning of the configure options:

`--disable-shared`
This tells Gettext not to create a shared library.

Only one program in the Gettext package needs to be built:

```
make -C lib
make -C src msgfmt
```

Install the **msgfmt** binary:

```
cp -v src/msgfmt /tools/bin
```

Details on this package are located in Section 10.32.2, “Contents of Gettext.”

6.13. Grep-2.5.1a

The Grep package contains programs for searching through files.

6.13.1. Installation of Grep

Prepare Grep for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET} \
   --disable-perl-regexp
```

The meaning of the configure options:

`--disable-perl-regexp`
This ensures that the **grep** program does not get linked against a Perl Compatible Regular Expression (PCRE) library that may be present on the host but will not be available when building the final system.

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.33.2, “Contents of Grep.”

6.14. Gzip-1.3.5

The Gzip package contains programs for compressing and decompressing files.

6.14.1. Installation of Gzip

Setup **configure** so it can use cross-compiled tools:

```
cp -v configure{,.orig}
sed -e "s@nm conftest@${CLFS_TARGET}-&@" configure.orig > configure
```

Prepare Gzip for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
  --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.35.2, “Contents of Gzip.”

6.15. Make-3.81

The Make package contains a program for compiling packages.

6.15.1. Installation of Make

Prepare Make for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.39.2, “Contents of Make.”

6.16. Patch-2.5.9

The Patch package contains a program for modifying or creating files by applying a “patch” file typically created by the **diff** program.

6.16.1. Installation of Patch

Prepare Patch for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
    --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.44.2, “Contents of Patch.”

6.17. Sed-4.1.5

The Sed package contains a stream editor.

6.17.1. Installation of Sed

Prepare Sed for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.16.2, “Contents of Sed.”

6.18. Tar-1.15.1

The Tar package contains an archiving program.

6.18.1. Installation of Tar

Tar has an issue with cross-compiling where it can't find `getline`. Fix it here:

```
echo "am_cv_func_working_getline=yes" >> config.cache
```

Prepare Tar for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET} \
   --cache-file=config.cache
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.49.2, “Contents of Tar.”

6.19. Texinfo-4.8

The Texinfo package contains programs for reading, writing, and converting info pages.

6.19.1. Installation of Texinfo

Prepare Texinfo for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/tools \
   --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.50.2, “Contents of Texinfo.”

6.20. To Boot or to Chroot?

There are two different ways you can proceed from this point to build the final system. You can build a kernel, a bootloader, and a few other utilities, boot into the temporary system, and build the rest there. Alternatively, you can chroot into the temporary system.

The boot method is needed when you are building on a different architecture. For example, if you are building a PowerPC system from an x86, you can't chroot. The chroot method is for when you are building on the same architecture. If you are building on, and for, an x86 system, you can simply chroot. The rule of thumb here is if the architectures match and you are running the same series kernel you can just chroot. If you aren't running the same series kernel, or are wanting to run a different ABI, you will need to use the boot option.

If you are in any doubt about this, you can try the following commands to see if you can chroot:

```
/tools/lib/libc.so.6
/tools/bin/gcc -v
```

If either of these commands fail, you will have to follow the boot method.

To chroot, you will also need a Linux Kernel-2.6.x (having been compiled with GCC-3.0 or greater). The reason for the kernel version requirement is that, without it, thread-local storage support in Binutils will not be built and the Native POSIX Threading Library (NPTL) test suite will segfault.

To check your kernel version, run **cat /proc/version** - if it does not say that you are running a 2.6.2 or later Linux kernel, compiled with GCC 3.0 or later, you cannot chroot.

For the boot method, follow If You Are Going to Boot.

For the chroot method, follow If You Are Going to Chroot.

Chapter 7. If You Are Going to Boot

7.1. Introduction

This chapter shows how to complete the build of temporary tools to create a minimal system that will be used to boot the target machine and to build the final system packages.

There are a few additional packages that will need to be installed to allow you to boot the minimal system. Some of these packages will be installed onto root or in /usr on the CLFS partition (${CLFS}/bin, ${CLFS}/usr/bin, etc...), rather than /tools, using the "DESTDIR" option with make. This will require the `clfs` user to have write access to the rest of the CLFS partition, so you will need to temporarily change the ownership of ${CLFS} to the `clfs` user. Run the following command as `root`:

```
chown -v clfs ${CLFS}
```

7.2. Creating Directories

It is time to create some structure in the CLFS file system. Create a standard directory tree by issuing the following commands:

```
mkdir -pv ${CLFS}/{bin,boot,dev,{etc/,}opt,home,lib,mnt}
mkdir -pv ${CLFS}/{proc,media/{floppy,cdrom},sbin,srv,sys}
mkdir -pv ${CLFS}/var/{lock,log,mail,run,spool}
mkdir -pv ${CLFS}/var/{opt,cache,lib/{misc,locate},local}
install -dv -m 0750 ${CLFS}/root
install -dv -m 1777 ${CLFS}{/var,}/tmp
mkdir -pv ${CLFS}/usr/{,local/}{bin,include,lib,sbin,src}
mkdir -pv ${CLFS}/usr/{,local/}share/{doc,info,locale,man}
mkdir -pv ${CLFS}/usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv ${CLFS}/usr/{,local/}share/man/man{1,2,3,4,5,6,7,8}
for dir in ${CLFS}/usr{,/local}; do
  ln -sv share/{man,doc,info} $dir
done
```

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

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

7.2.1. FHS Compliance Note

The directory tree is based on the Filesystem Hierarchy Standard (FHS) (available at *http://www.pathname.com/fhs/*). In addition to the tree created above, this standard stipulates the existence of `/usr/local/games` and `/usr/share/games`. The FHS is not precise as to the structure of the `/usr/local/share` subdirectory, so we create only the directories that are needed. However, feel free to create these directories if you prefer to conform more strictly to the FHS.

7.3. Creating Essential Symlinks

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

```
ln -sv /tools/bin/{bash,cat,grep,pwd,stty} ${CLFS}/bin
ln -sv /tools/lib/libgcc_s.so{,.1} ${CLFS}/usr/lib
ln -sv /tools/lib/libstd* ${CLFS}/usr/lib
ln -sv bash ${CLFS}/bin/sh
```

7.4. Zlib-1.2.3

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

7.4.1. Installation of Zlib

Prepare Zlib for compilation:

```
CC="${CC} ${BUILD64}" \
   ./configure --prefix=/tools --shared
```

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 10.22.2, “Contents of Zlib.”

7.5. E2fsprogs-1.39

The E2fsprogs package contains the utilities for handling the `ext2` file system. It also supports the `ext3` journaling file system.

7.5.1. Installation of E2fsprogs

The E2fsprogs documentation recommends that the package be built in a subdirectory of the source tree:

```
mkdir -v build
cd build
```

Prepare E2fsprogs for compilation:

```
CC="${CC} ${BUILD64}" ../configure --prefix=/tools \
--enable-elf-shlibs --disable-evms \
--host=${CLFS_TARGET}
```

The meaning of the configure options:

`--enable-elf-shlibs`
This creates the shared libraries which some programs in this package use.

`--disable-evms`
This disables the building of the Enterprise Volume Management System (EVMS) plugin. This plugin is not up-to-date with the latest EVMS internal interfaces and EVMS is not installed as part of a base CLFS system, so the plugin is not required. See the EVMS website at *http://evms.sourceforge.net/* for more information regarding EVMS.

```
make CC="${CC} ${BUILD64}"
```

Install the binaries and documentation:

```
make DESTDIR=${CLFS} install
```

The meaning of the make option:

`DESTDIR=${CLFS}`
The Makefile for e2fsprogs hard-codes a path to the `mke2fs.conf` file, attempting to install it into ${DESTDIR}/etc, causing the installation to fail as it tries to write to /etc. The DESTDIR parameter prevents this.

Install the shared libraries:

```
make install-libs
```

Create needed symlinks for a bootable system:

```
ln -sv /tools/sbin/{fsck.ext2,fsck.ext3,e2fsck} ${CLFS}/sbin
```

Details on this package are located in Section 10.28.2, “Contents of E2fsprogs.”

7.6. Sysvinit-2.86

The Sysvinit package contains programs for controlling the startup, running, and shutdown of the system.

7.6.1. Installation of Sysvinit

Make some modifications to allow you to boot into the minimal temp-system:

```
cp -v src/Makefile src/Makefile.orig
sed -e 's@root@0@g' \
    -e "s@/dev/initctl@${CLFS}&@g" \
    -e 's@\(mknod \)-m \([0-9]* \)\(.* \)p@\1\3p; chmod \2\3@g' \
    -e "s@/usr/lib@/tools/lib@" \
    src/Makefile.orig > src/Makefile
```

Compile the package:

```
make -C src clobber
make -C src CC="${CC} ${BUILD64}"
```

Install the package:

```
make -C src install INSTALL="install" ROOT="${CLFS}"
```

7.6.2. Configuring Sysvinit

Create a new file `${CLFS}/etc/inittab` by running the following:

```
cat > ${CLFS}/etc/inittab << "EOF"
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty -I '\033(K' tty1 9600
2:2345:respawn:/sbin/agetty -I '\033(K' tty2 9600
3:2345:respawn:/sbin/agetty -I '\033(K' tty3 9600
4:2345:respawn:/sbin/agetty -I '\033(K' tty4 9600
```

```
5:2345:respawn:/sbin/agetty -I '\033(K' tty5 9600
6:2345:respawn:/sbin/agetty -I '\033(K' tty6 9600

# End /etc/inittab
EOF
```

Details on this package are located in Section 10.48.3, “Contents of Sysvinit.”

7.7. Module-Init-Tools-3.2.2

The Module-Init-Tools package contains programs for handling kernel modules in Linux kernels greater than or equal to version 2.5.47.

7.7.1. Installation of Module-Init-Tools

Prepare Module-Init-Tools for compilation:

```
CC="${CC} ${BUILD64}" ./configure --prefix=/ \
   --build=${CLFS_HOST} --host=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR="${CLFS}" install
```

Details on this package are located in Section 10.43.2, “Contents of Module-Init-Tools.”

7.8. Util-linux-2.12r

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

7.8.1. Installation of Util-linux

Util-linux fails to compile against newer versions of Linux kernel headers. The following patch properly fixes this issue:

```
patch -Np1 -i ../util-linux-2.12r-cramfs-1.patch
```

The following patch fixes build issues with GCC 4.1.1:

```
patch -Np1 -i ../util-linux-2.12r-gcc4_fixes-1.patch
```

The following patch fixes swapon.c - it tries to find the variable R_OK, but the header that has R_OK is not included:

```
patch -Np1 -i ../util-linux-2.12r-missing_header-1.patch
```

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

```
cp -v configure{,.orig}
sed -e 's@/usr/include@/tools/include@g' configure.orig > configure
```

The Util-linux installation uses `-o root`. The following **sed** removes that since we don't have users setup yet:

```
cp -v MCONFIG{,.orig}
sed -e 's|-o root||' MCONFIG.orig > MCONFIG
```

Prepare Util-linux for compilation:

```
CC="${CC} ${BUILD64}" ./configure
```

Compile the package:

```
make HAVE_KILL=yes HAVE_SLN=yes \
    HAVE_SHADOW=no CPUOPT="" ARCH="" \
    CPU=""
```

The meaning of the make parameters:

HAVE_KILL=yes
: This prevents the **kill** program from being built.

HAVE_SLN=yes
: This prevents the **sln** program (a statically linked version of **ln** already installed by Glibc) from being built and installed again.

HAVE_SHADOW=no
This disables linking to shadow .

CPUOPT=""
This disables any compiler optimizations by CPU type.

ARCH=""
This disables the detection of the architecture.

CPU=""
This disables the detection of the CPU.

Install the package:

```
make HAVE_KILL=yes HAVE_SLN=yes HAVE_SHADOW=no \
    USE_TTY_GROUP=no CPUOPT="" ARCH="" \
    CPU="" DESTDIR=${CLFS} install
```

Details on this package are located in Section 10.52.3, “Contents of Util-linux.”

7.9. Udev-096

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

7.9.1. Installation of Udev

Compile the package:

```
make CROSS_COMPILE="${CLFS_TARGET}-" CC="${CC} ${BUILD64}" \
    LD="${CC} ${BUILD64}"
```

Install the package:

```
make DESTDIR=${CLFS} install
```

Details on this package are located in Section 10.51.2, “Contents of Udev.”

7.10. Creating the passwd, group, and log Files

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

Create the `${CLFS}/etc/passwd` file by running the following command:

```
cat > ${CLFS}/etc/passwd << "EOF"
root::0:0:root:/root:/bin/bash
EOF
```

The actual password for `root` (the “::” used here is just a placeholder and allow you to login with no password) will be set later.

Additional users you may want to add:

`bin:x:1:1:bin:/bin:/bin/false`
Can be useful for compatibility with legacy applications.

`daemon:x:2:6:daemon:/sbin:/bin/false`
It is often recommended to use an unprivileged User ID/Group ID for daemons to run as, in order to limit their access to the system.

`adm:x:3:16:adm:/var/adm:/bin/false`
Was used for programs that performed administrative tasks.

`lp:x:10:9:lp:/var/spool/lp:/bin/false`
Used by programs for printing

`mail:x:30:30:mail:/var/mail:/bin/false`
Often used by email programs

`news:x:31:31:news:/var/spool/news:/bin/false`
Often used for network news servers

`uucp:x:32:32:uucp:/var/spool/uucp:/bin/false`
Often used for Unix-to-Unix Copy of files from one server to the next

`operator:x:50:0:operator:/root:/bin/bash`
Often used to allow system operators to access the system

`postmaster:x:51:30:postmaster:/var/spool/mail:/bin/false`
Generally used as an account that receives all the information of troubles with the mail server

`nobody:x:65534:65534:nobody:/:/bin/false`
Used by NFS

Create the `${CLFS}/etc/group` file by running the following command:

```
cat > ${CLFS}/etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
```

```
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
EOF
```

Additional groups you may want to add

`adm:x:16:root,adm,daemon`
All users in this group are allowed to do administrative tasks

`console:x:17:`
This group has direct access to the console

`cdrw:x:18:`
This group is allowed to use the CDRW drive

`mail:x:30:mail`
Used by MTAs (Mail Transport Agents)

`news:x:31:news`
Used by Network News Servers

`uucp:x:32:uucp`
Used by the Unix-to-Unix copy users

`users:x:1000:`
The default GID used by shadow for new users

`nogroup:x:65533:`
This is a default group used by some programs that do not require a group

`nobody:x:65534:`
This is used by NFS

The created groups are not part of any standard—they are groups decided on in part by the requirements of the Udev configuration in the final system, and in part by common convention employed by a number of existing Linux distributions. The Linux Standard Base (LSB, available at *http://www.linuxbase.org*) recommends only that, besides the group “root” with a Group ID (GID) of 0, a group “bin” with a GID of 1 be present. All other group names and GIDs can be chosen freely by the system administrator since well-written programs do not depend on GID numbers, but rather use the group's name.

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

already exist. Initialize the log files and give them proper permissions:

```
touch ${CLFS}/var/run/utmp ${CLFS}/var/log/{btmp,lastlog,wtmp}
chmod -v 664 ${CLFS}/var/run/utmp ${CLFS}/var/log/lastlog
chmod -v 600 ${CLFS}/var/log/btmp
```

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

7.11. Linux-2.6.17.13

The Linux package contains the Linux kernel.

7.11.1. Installation of the kernel

Warning

Here a temporary cross-compiled kernel will be built. When configuring it, select the minimal amount of options required to boot the target machine and build the final system. I.e., no support for sound, printers, etc. will be needed.

Also, try to avoid the use of modules if possible, and don't use the resulting kernel image for production systems.

Building the kernel involves a few steps—configuration, compilation, and installation. Read the `README` file in the kernel source tree for alternative methods to the way this book configures the kernel.

The following patch fixes on intialization issue with the tulip network driver:

```
patch -Np1 -i ../linux-2.6.17.13-tulip-1.patch
```

To ensure that your system boots and you can properly run both 32 bit and 64 bit binaries, please make sure that you enable support for ELF and emulations for 32bit ELF into the kernel.

Prepare for compilation by running the following command:

```
make mrproper
```

This ensures that the kernel tree is absolutely clean. The kernel team recommends that this command be issued prior to each kernel compilation. Do not rely on the source tree being clean after un-tarring.

If your keyboard isn't a US one and you have Kbd installed on your host system, you can build the proper keymap for your keyboard layout inside the kernel. If you wish to do this, run the following command (replace [path to keymap] with the keymap location on the host - a common location is in /usr/share/kbd):

```
loadkeys -m [path to  keymap] > \
    drivers/char/defkeymap.c
```

For example, if using a Dutch keyboard, use `/usr/share/kbd/keymaps/i386/qwerty/nl.map.gz`.

Configure the kernel via a menu-driven interface:

```
make ARCH=x86_64 CROSS_COMPILE=${CLFS_TARGET}- menuconfig
```

Compile the kernel image and modules:

```
make ARCH=x86_64 CROSS_COMPILE=${CLFS_TARGET}-
```

If the use of kernel modules can't be avoided, an `/etc/modprobe.conf` file may be needed. Information pertaining to modules and kernel configuration is located in the kernel documentation in the `Documentation`

directory of the kernel sources tree. The `modprobe.conf` man page may also be of interest.

Be very careful when reading other documentation relating to kernel modules because it usually applies to 2.4.x kernels only. As far as we know, kernel configuration issues specific to Hotplug and Udev are not documented. The problem is that Udev will create a device node only if Hotplug or a user-written script inserts the corresponding module into the kernel, and not all modules are detectable by Hotplug. Note that statements like the one below in the `/etc/modprobe.conf` file do not work with Udev:

```
alias char-major-XXX some-module
```

Install the modules, if the kernel configuration uses them:

```
make ARCH=x86_64 CROSS_COMPILE=${CLFS_TARGET}- \
   INSTALL_MOD_PATH=${CLFS} modules_install
```

After kernel compilation is complete, additional steps are required to complete the installation. Some files need to be copied to the `${CLFS}/boot` directory.

Issue the following command to install the kernel:

```
cp -v arch/x86_64/boot/bzImage ${CLFS}/boot/clfskernel-2.6.17.13
```

`System.map` is a symbol file for the kernel. It maps the function entry points of every function in the kernel API, as well as the addresses of the kernel data structures for the running kernel. Issue the following command to install the map file:

```
cp -v System.map ${CLFS}/boot/System.map-2.6.17.13
```

The kernel configuration file `.config` produced by the **make menuconfig** step above contains all the configuration selections for the kernel that was just compiled. It is a good idea to keep this file for future reference:

```
cp -v .config ${CLFS}/boot/config-2.6.17.13
```

Details on this package are located in Section 12.3.2, “Contents of Linux.”

7.12. Building a bootloader

On x86 and x86_64 (multilib) architectures, the preferred bootloader is GRUB. Unfortunately, GRUB doesn't work on x86_64 Pure64 - the stage2 files can be correctly built as 32-bit, but the **grub** shell is a 64-bit program, and tries to execute some of the stage2 routines - this results in a segmentation fault. Therefore, in the final system we use Lilo as the bootloader.

If you already have a bootloader, such as GRUB, installed on the system then you should use that to make your new kernel bootable.

Note

We will now cross-compile Bin86 and Lilo - these instructions assume you are using an x86_64 machine (e.g. booted fron a Live CD), either pure64 or multilib. This approach will not work if you are running the machine as i686, because a 32-bit kernel will not be able to execute a 64-bit binary to install the bootloader. If that is the case, you will need to install an i686 bootloader on the host system.

7.13. Bin86-0.16.17

The Bin86 package contains an assembler and linker to produce 16 or 32-bit 8086 machine code.

7.13.1. Installation of Bin86

We are building Bin86 so that we can compile Lilo. Both **as86** and **ld86** need to run on the host system to assemble x86_64 code. We cannot compile the whole package like this, but fortunately these two programs are the only parts we require.

This patch updates Bin86 to compile on x86_64:

```
patch -Np1 -i ../bin86-0.16.17-x86_64-1.patch
```

The Bin86 package does not contain a **configure** script. Natively compile only the necessary parts with:

```
make CC=gcc -C as as86
make CC=gcc -C ld ld86
```

Install the assembler and linker where they will be on the PATH when we build Lilo, using prefixes to show that their output is not for a native system.

```
install -v -m 755 -s as/as86 /cross-tools/bin/${CLFS_TARGET}-as86
install -v -m 755 -s ld/ld86 /cross-tools/bin/${CLFS_TARGET}-ld86
```

Details on this package are located in Section 10.54.2, “Contents of Bin86.”

7.14. Lilo-22.7.1

The Lilo package contains the Linux Loader, a bootloader.

We have chosen to use Lilo because at the moment no other bootloader builds and runs on a pure64 system. If your machine has multiple systems (i.e. x86 or multilib) you may prefer to use the bootloader from the other system, such as GRUB.

7.14.1. Installation of Lilo

The following patch forces Lilo to use our cross-compiler (except for a couple of utilities which run during the compile), and to look for as86 and ld86 under the names by which we installed them.

```
patch -Np1 -i ../lilo-22.7.1-cross_compile_x86_64-1.patch
```

Compile the program which installs the loader, linking it statically so that it does not have to link to host libraries when it runs:

```
make lilo-static
```

Install only the parts we need:

```
install -v -m755 lilo-static /tools/bin
install -v -m755 keytab-lilo.pl /tools/bin
```

Details on this package are located in Section 10.55.2, “Contents of Lilo.”

7.15. Setting Up the Environment

The new instance of the shell that will start when the system is booted is a *login* shell, which will read `.bash_profile` file. Create the `.bash_profile` file now:

```
cat > ${CLFS}/root/.bash_profile << "EOF"
set +h
PS1='\u:\w\$ '
LC_ALL=POSIX
PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin:/tools/sbin
export LC_ALL PATH PS1
EOF
```

The `LC_ALL` variable controls the localization of certain programs, making their messages follow the conventions of a specified country. Setting `LC_ALL` to "POSIX" or "C" (the two are equivalent) ensures that everything will work as expected on your temporary system.

By putting `/tools/bin` at the end of the standard `PATH`, all the programs installed in Constructing a Temporary System are only picked up by the shell if they have not yet been built on the target system. This configuration forces use of the final system binaries as they are built over the temp-system, minimising the chance of final system programs being built against the temp-system.

7.16. Build Flags

We will need to copy our build variables into our new system:

```
echo export BUILD64=\""${BUILD64}\"" >> ${CLFS}/root/.bash_profile
```

7.17. Creating the /etc/fstab File

The `/etc/fstab` file is used by some programs to determine where file systems are to be mounted by default, which must be checked, and in which order. Create a new file systems table like this:

```
cat > ${CLFS}/etc/fstab << "EOF"
# Begin /etc/fstab

# file system  mount-point  type   options          dump  fsck
#                                                        order

/dev/[xxx]     /            [fff]  defaults         1     1
/dev/[yyy]     swap         swap   pri=1            0     0
proc           /proc        proc   defaults         0     0
sysfs          /sys         sysfs  defaults         0     0
devpts         /dev/pts     devpts gid=4,mode=620   0     0
shm            /dev/shm     tmpfs  defaults         0     0
# End /etc/fstab
EOF
```

Replace *[xxx]*, *[yyy]*, and *[fff]* with the values appropriate for the system, for example, `hda2`, `hda5`, and `ext2`. For details on the six fields in this file, see **man 5 fstab**.

The `/dev/shm` mount point for `tmpfs` is included to allow enabling POSIX-shared memory. The kernel must have the required support built into it for this to work (more about this is in the next section). Please note that very little software currently uses POSIX-shared memory. Therefore, consider the `/dev/shm` mount point optional. For more information, see `Documentation/filesystems/tmpfs.txt` in the kernel source tree.

7.18. CLFS-Bootscripts-1.0

The CLFS-Bootscripts package contains a set of scripts to start/stop the CLFS system at bootup/shutdown.

7.18.1. Installation of CLFS-Bootscripts

Install the package:

```
make ETCDIR=${CLFS}/etc minimal
```

The **setclock** script reads the time from the hardware clock, also known as the BIOS or the Complementary Metal Oxide Semiconductor (CMOS) clock. If the hardware clock is set to UTC, this script will convert the hardware clock's time to the local time using the `/etc/localtime` file (which tells the **hwclock** program which timezone the user is in). There is no way to detect whether or not the hardware clock is set to UTC, so this needs to be configured manually.

If you do not know whether or not the hardware clock is set to UTC, you can find out after you have booted the new machine by running the **`hwclock --localtime --show`** command, and if necessary editing the `/etc/sysconfig/clock` script. The worst that will happen if you make a wrong guess here is that the time displayed will be wrong.

Change the value of the `UTC` variable below to a value of `0` (zero) if the hardware clock is *not* set to UTC time.

```
cat > ${CLFS}/etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock

UTC=1

# End /etc/sysconfig/clock
EOF
```

Details on this package are located in Section 11.2.2, “Contents of CLFS-Bootscripts.”

7.19. Udev Rules-1.0-3

The Udev Cross-LFS rules package contains the necessary rules set for a basic functional system.

7.19.1. Installation of Udev-Rules

Install the package:

```
make DESTDIR=${CLFS} install-minimal
```

Details on this package are located in Section 11.3.2, “Contents of Udev Rules.”

7.20. Populating /dev

7.20.1. Creating Initial Device Nodes

Note

The commands in the remainder of the book should be run as the `root` user. Also, double-check that ${CLFS} is set as `root`.

When the kernel boots the system, it requires the presence of a few device nodes, in particular the `console` and `null` devices. The device nodes will be created on the hard disk so that they are available before **udev** has been started, and additionally when Linux is started in single user mode (hence the restrictive permissions on `console`). Create these by running the following commands:

```
mknod -m 600 ${CLFS}/dev/console c 5 1
mknod -m 666 ${CLFS}/dev/null c 1 3
```

7.21. Changing Ownership

Currently, the `${CLFS}` directory and all of its subdirectories are owned by the user `clfs`, a user that exists only on the host system. For security reasons, the ${CLFS} root directory and all of it subdirectories should be owned by `root`. Change the ownership for ${CLFS} and its subdirectories by running this command:

```
chown -Rv root:root ${CLFS}
```

7.22. Making the CLFS System Bootable

You are nearly ready to boot to the new temporary system. One of the last things to do is to ensure that the system can be booted. The instructions below apply only to x86_64 machines on which Lilo is going to be installed. Information on using a pre-installed GRUB on machines currently running as x86 or x86_64 multilib should be available in the usual resource-specific locations for those architectures. If you have installed an x86 version of Lilo, these instructions should be modified to refer to the **lilo** and **keytab-lilo.pl** commands you installed on the host system.

Boot loading can be a complex area, so a few cautionary words are in order. Be familiar with the current boot loader and any other operating systems present on the hard drive(s) that need to be bootable. Make sure that an emergency boot disk is ready to “rescue” the computer if the computer becomes unusable (un-bootable).

If you have multiple systems on your machine using a different bootloader such as GRUB, you may prefer to use that instead - consult the appropriate documentation. The rest of this section assumes you are going to use Lilo.

Earlier, we compiled and installed the Lilo boot loader software in preparation for this step. The procedure involves writing a boot image to a specific location on the hard drive. If you have a floppy disk drive, or if you have installed a cd recording program, we highly recommend using mkrescue to create a Lilo boot floppy, or CD (using e.g. dvdrecord from dvdrtools) as a test and as a backup.

Normally, you interact with Lilo by using the cursor and `enter` keys to select from the available option(s), but sometimes it is necessary to add other boot options, such as e.g. 'init=/bin/bash' to debug boot failures. The more your keyboard layout differs from the US qwerty layout, the harder it becomes to type boot options unless Lilo knows about your keyboard layout. So, we will create a key table for Lilo (.ktl) file - at one point in the documentation these are referred to as .klt files, which may be a typo, but has been followed by some distros. The name, and location, are not important but it is conventional to put these in /boot with the name representing the key layout. For a British keyboard layout, the following command will achieve this:

```
keytab-lilo.pl uk >i${CLFS}/boot/uk.ktl
```

The argument to the command is the name of the keymap, or if necessary you can specify the full path to the keymap. Use whatever is appropriate for your keyboard.

The next step is to create /etc/lilo.conf:

```
cat > ${CLFS}/etc/lilo.conf << "EOF"
# Begin /etc/lilo.conf
# lilo.conf
#
# global options
boot=/dev/<bootdisk>
keytable=/boot/<keytable>
lba32
map=/boot/map
prompt

# set the name of the default image to boot
default=clfs

# define an image
```

```
image=/boot/clfskernel
    label=clfs
    root=/dev/<partition>
    read-only
# optionally add parameters to pass, e.g.
#   append="video=radeonfb:1024x768-16@70"

# if you had an existing system, you could
# add its details here.
EOF
```

Replace <bootdisk> with the name of the disk (or partition) on which the boot sector is to be written, e.g. sda. Replace <keytable> with the name of the keytable file you created, and <partition> with the name of the root partition for the new system.

Warning

The following command will overwrite any current boot loader. Do not run the command if this is not desired. If you have cross-compiled on a different machine from the target, you must install the bootloader on the target machine, the installed boot block is not a file which can be copied with **tar**.

Run Lilo:

```
/tools/bin/lilo-static -v
```

Note

People who have been used to GRUB need to be aware that Lilo works differently - in particular, you cannot edit the available choices as you can in the **grub** shell, and Lilo records the block addresses of the kernels into the boot blocks each time /sbin/lilo is run. This means that when you compile a new kernel, you have to add it to /etc/lilo.conf and rerun /sbin/lilo. It also means that if you recompile an existing kernel and save it to the same name you still have to rerun /sbin/lilo in case it now occupies different blocks on the filesystem.

7.23. What to do next

Now you're at the point to get your ${CLFS} directory copied over to your target machine. The easiest method would be to tar it up and copy the file.

```
tar -jcvf ${CLFS}.tar.bz2 ${CLFS}
```

Some others have came up with other ideas on how to do this. Below is a table with the method and link to where the information is stored.

Table 7.1. Boot Methods

Boot Method	For Architectures
nfsroot	MIPS, PPC, Sparc, x86, and x86_64

Chapter 8. If You Are Going to Chroot

8.1. Introduction

This chapter shows how to prepare a **chroot** jail to build the final system packages into.

8.2. Util-linux-2.12r

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

8.2.1. Installation of Util-linux

The following patch fixes build issues with GCC 4.1.1:

```
patch -Np1 -i ../util-linux-2.12r-gcc4_fixes-1.patch
```

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

```
cp -v configure{,.orig}
sed -e 's@/usr/include@/tools/include@g' configure.orig > configure
```

Prepare Util-linux for compilation:

```
CC="${CC} ${BUILD64}" ./configure
```

Compile some support routines:

```
make ARCH="" CPU="" -C lib
```

The meaning of the make parameters:

ARCH=""
This disables the detection of the architecture.

CPU=""
This disables the detection of the CPU.

Only a few of the utilities contained in this package need to be built:

```
make ARCH="" CPU="" -C mount mount umount
make ARCH="" CPU="" -C text-utils more
```

Copy these programs to the temporary tools directory:

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

Details on this package are located in Section 10.52.3, “Contents of Util-linux.”

8.3. Mounting Virtual Kernel File Systems

Note

The commands in the remainder of the book should be run as the `root` user. Also, double-check that ${CLFS} is set as `root`.

Various file systems exported by the kernel are used to communicate to and from the kernel itself. These file systems are virtual in that no disk space is used for them. The content of the file systems resides in memory.

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

```
mkdir -pv ${CLFS}/{dev,proc,sys}
```

Now mount the file systems:

```
mount -vt proc proc ${CLFS}/proc
mount -vt sysfs sysfs ${CLFS}/sys
```

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

Two device nodes, /dev/console and /dev/null, are required to be present on the filesystem. These are needed by the kernel even before starting Udev early in the boot process, so we create them here:

```
mknod -m 600 ${CLFS}/dev/console c 5 1
mknod -m 666 ${CLFS}/dev/null c 1 3
```

Once the system is complete and booting, the rest of our device nodes are created by the Udev package. Since this package is not available to us right now, we must take other steps to provide device nodes under on the CLFS filesystem. We will use the “bind” option in the mount command to make our host system's /dev structure appear in the new CLFS filesystem:

```
mount -v -o bind /dev ${CLFS}/dev
```

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

```
mount -f -vt tmpfs tmpfs ${CLFS}/dev/shm
mount -f -vt devpts -o gid=4,mode=620 devpts ${CLFS}/dev/pts
```

8.4. Entering the Chroot Environment

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

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

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

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

Notice that `/tools/bin` comes last in the `PATH`. This means that a temporary tool will no longer be used once its final version is installed. This occurs when the shell does not "remember" the locations of executed binaries—for this reason, hashing is switched off by passing the *`+h`* option to **bash**.

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

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

8.5. Changing Ownership

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

To avoid this issue, add the `clfs` user to the new CLFS system later when creating the `/etc/passwd` file, taking care to assign it the same user and group IDs as on the host system. Alternatively, assign the contents of the `/tools` and `/cross-tools` directories to user `root` by running the following commands:

```
chown -Rv 0:0 /tools
chown -Rv 0:0 /cross-tools
```

The commands use *`0:0`* instead of *`root:root`*, because **chown** is unable to resolve the name “root” until the `passwd` file has been created.

8.6. Creating Directories

It is time to create some structure in the CLFS file system. Create a standard directory tree by issuing the following commands:

```
mkdir -pv /{bin,boot,dev,{etc/,}opt,home,lib,mnt}
mkdir -pv /{proc,media/{floppy,cdrom},sbin,srv,sys}
mkdir -pv /var/{lock,log,mail,run,spool}
mkdir -pv /var/{opt,cache,lib/{misc,locate},local}
install -dv -m 0750 /root
install -dv -m 1777 {/var,}/tmp
mkdir -pv /usr/{,local/}{bin,include,lib,sbin,src}
mkdir -pv /usr/{,local/}share/{doc,info,locale,man}
mkdir -pv /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
for dir in /usr{,/local}; do
  ln -sv share/{man,doc,info} $dir
done
```

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

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

8.6.1. FHS Compliance Note

The directory tree is based on the Filesystem Hierarchy Standard (FHS) (available at *http://www.pathname.com/fhs/*). In addition to the tree created above, this standard stipulates the existence of `/usr/local/games` and `/usr/share/games`. The FHS is not precise as to the structure of the `/usr/local/share` subdirectory, so we create only the directories that are needed. However, feel free to create these directories if you prefer to conform more strictly to the FHS.

8.7. Creating Essential Symlinks

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

```
ln -sv /tools/bin/{bash,cat,grep,pwd,stty} /bin
ln -sv /tools/lib/libgcc_s.so{,.1} /usr/lib
ln -sv /tools/lib/libstd* /usr/lib
ln -sv bash /bin/sh
```

8.8. Build Flags

We will need to setup target specific flags for the compiler and linkers.

```
export BUILD64="-m64"
```

To prevent errors when you come back to your build, we will export these variables to prevent any build issues in the future:

```
echo export BUILD64=\""${BUILD64}\"" >> ~/.bash_profile
```

8.9. Creating the passwd, group, and log Files

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

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

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

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

Additional users you may want to add:

`bin:x:1:1:bin:/bin:/bin/false`
Can be useful for compatibility with legacy applications.

`daemon:x:2:6:daemon:/sbin:/bin/false`
It is often recommended to use an unprivileged User ID/Group ID for daemons to run as, in order to limit their access to the system.

`adm:x:3:16:adm:/var/adm:/bin/false`
Was used for programs that performed administrative tasks.

`lp:x:10:9:lp:/var/spool/lp:/bin/false`
Used by programs for printing

`mail:x:30:30:mail:/var/mail:/bin/false`
Often used by email programs

`news:x:31:31:news:/var/spool/news:/bin/false`
Often used for network news servers

`uucp:x:32:32:uucp:/var/spool/uucp:/bin/false`
Often used for Unix-to-Unix Copy of files from one server to the next

`operator:x:50:0:operator:/root:/bin/bash`
Often used to allow system operators to access the system

`postmaster:x:51:30:postmaster:/var/spool/mail:/bin/false`
Generally used as an account that receives all the information of troubles with the mail server

`nobody:x:65534:65534:nobody:/:/bin/false`
Used by NFS

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

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
```

```
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
EOF
```

Additional groups you may want to add

`adm:x:16:root,adm,daemon`
All users in this group are allowed to do administrative tasks

`console:x:17:`
This group has direct access to the console

`cdrw:x:18:`
This group is allowed to use the CDRW drive

`mail:x:30:mail`
Used by MTAs (Mail Transport Agents)

`news:x:31:news`
Used by Network News Servers

`uucp:x:32:uucp`
Used by the Unix-to-Unix copy users

`users:x:1000:`
The default GID used by shadow for new users

`nogroup:x:65533:`
This is a default group used by some programs that do not require a group

`nobody:x:65534:`
This is used by NFS

The created groups are not part of any standard—they are groups decided on in part by the requirements of the Udev configuration in the final system, and in part by common convention employed by a number of existing Linux distributions. The Linux Standard Base (LSB, available at *http://www.linuxbase.org*) recommends only that, besides the group “root” with a Group ID (GID) of 0, a group “bin” with a GID of 1 be present. All other group names and GIDs can be chosen freely by the system administrator since well-written programs do not depend on GID numbers, but rather use the group's name.

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

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

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

Note the use of the *+h* directive. This tells **bash** not to use its internal path hashing. Without this directive, **bash** would remember the paths to binaries it has executed. To ensure the use of the newly compiled binaries as soon as they are installed, the *+h* directive will be used for the duration of the nexts chapters.

The created groups are not part of any standard—they are groups decided on in part by the requirements of the Udev configuration in the final system, and in part by common convention employed by a number of existing Linux distributions. The Linux Standard Base (LSB, available at *http://www.linuxbase.org*) recommends only that, besides the group “root” with a Group ID (GID) of 0, a group “bin” with a GID of 1 be present. All other group names and GIDs can be chosen freely by the system administrator since well-written programs do not depend on GID numbers, but rather use the group's name.

```
touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}
chgrp -v utmp /var/run/utmp /var/log/lastlog
chmod -v 664 /var/run/utmp /var/log/lastlog
chmod -v 600 /var/log/btmp
```

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

8.10. Mounting Kernel Filesystems

8.10.1. Mounting Additional Kernel Filesystems

Mount the proper virtual (kernel) file systems on the newly-created directories:

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

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

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

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

Part V. Building the CLFS System

Chapter 9. Constructing Testsuite Tools

9.1. Introduction

This chapter builds the tools needed to run the tests that the packages have. I.e., **make check**

9.2. Tcl-8.4.12

The Tcl package contains the Tool Command Language.

9.2.1. Installation of Tcl

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

First, fix a syntax error in the configure script:

```
cd unix
sed -i "s/relid'/relid/" configure
```

Prepare Tcl for compilation:

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

Build the package:

```
make
```

Install the package:

```
make install
```

Tcl's private header files are needed for the next package, Expect. Install them into /tools:

```
make install-private-headers
```

Now make a necessary symbolic link:

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

9.2.2. Contents of Tcl

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

Short Descriptions

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

9.3. Expect-5.43.0

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

9.3.1. Installation of Expect

The following sed tells **configure** to look for libraries in ${libdir}, not just in /tools/lib:

```
sed -i '/EXP_LIB_SPEC=/s@${exec_prefix}/lib@${libdir}@' configure
```

Fix a bug that can result in false failures during the GCC test suite run:

```
patch -Np1 -i ../expect-5.43.0-spawn-2.patch
```

Now prepare Expect for compilation:

```
./configure --prefix=/tools --with-tcl=/tools/lib \
    --with-tclinclude=/tools/include
```

The meaning of the configure options:

`--with-tcl=/tools/lib`
This ensures that the configure script finds the Tcl installation in the temporary testsuite-tools location.

`--with-tclinclude=/tools/include`
This explicitly tells Expect where to find Tcl's internal headers. Using this option avoids conditions where **configure** fails because it cannot automatically discover the location of the Tcl source directory.

Build the package:

```
make
```

Install the package:

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

The meaning of the make parameter:

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

9.3.2. Contents of Expect

Installed program: expect
Installed library: libexpect-5.43.a

Short Descriptions

expect	Communicates with other interactive programs according to a script
`libexpect-5.43.a`	Contains functions that allow Expect to be used as a Tcl extension or to be used directly from C or C++ (without Tcl)

9.4. File-4.17

The File package contains a utility for determining the type of a given file or files.

9.4.1. Installation of File

Prepare File for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Now we create a symlink so that the GCC testsuite can use **file**:

```
ln -sv /tools/bin/file /usr/bin
```

Details on this package are located in Section 10.29.2, “Contents of File.”

9.5. DejaGNU-1.4.4

The DejaGNU package contains a framework for testing other programs.

9.5.1. Installation of DejaGNU

Prepare DejaGNU for compilation:

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

Build and install the package:

```
make install
```

9.5.2. Contents of DejaGNU

Installed program: runtest

Short Descriptions

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

9.6. Tree-1.5.0

The Tree package contains a program that lists the directory structure in a graphical "tree" format. It is used by Udev for failures in its testsuite.

9.6.1. Installation of Tree

Compile the package:

```
make
```

Install the package:

```
make prefix=/tools install
```

The meaning of the make parameters:

prefix=/tools
This overrides the default prefix of /usr/local in the Makefile.

9.6.2. Contents of Tree

Installed program: tree

Short Descriptions

tree Tree will list contents of directories in a tree-like format.

Chapter 10. Installing Basic System Software

10.1. Introduction

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

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

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

To keep track of which package installs particular files, a package manager can be used. For a general overview of different styles of package managers, please take a look at the next page.

10.2. Package Management

Package Management is an often-requested addition to the CLFS Book. A Package Manager allows tracking the installation of files making it easy to remove and upgrade packages. Before you begin to wonder, NO—this section will not talk about nor recommend any particular package manager. What it provides is a roundup of the more popular techniques and how they work. The perfect package manager for you may be among these techniques or may be a combination of two or more of these techniques. This section briefly mentions issues that may arise when upgrading packages.

Some reasons why no specific package manager is recommended in CLFS or BLFS include:

- Dealing with package management takes the focus away from the goals of these books—teaching how a Linux system is built.
- There are multiple solutions for package management, each having its strengths and drawbacks. Including one that satisfies all audiences is difficult.

There are some hints written on the topic of package management. Visit the *Hints subproject* and see if one of them fits your need.

10.2.1. Upgrade Issues

A Package Manager makes it easy to upgrade to newer versions when they are released. Generally the instructions in the CLFS and BLFS Book can be used to upgrade to the newer versions. Here are some points that you should be aware of when upgrading packages, especially on a running system.

- If one of the toolchain packages (Glibc, GCC or Binutils) needs to be upgraded to a newer minor version, it is safer to rebuild CLFS. Though you *may* be able to get by rebuilding all the packages in their dependency order, we do not recommend it. For example, if glibc-2.2.x needs to be updated to glibc-2.3.x, it is safer to rebuild. For micro version updates, a simple reinstallation usually works, but is not guaranteed. For example, upgrading from glibc-2.3.4 to glibc-2.3.5 will not usually cause any problems.
- If a package containing a shared library is updated, and if the name of the library changes, then all the packages dynamically linked to the library need to be recompiled to link against the newer library. (Note that there is no correlation between the package version and the name of the library.) For example, consider a package foo-1.2.3 that installs a shared library with name `libfoo.so.1`. Say you upgrade the package to a newer version foo-1.2.4 that installs a shared library with name `libfoo.so.2`. In this case, all packages that are dynamically linked to `libfoo.so.1` need to be recompiled to link against `libfoo.so.2`. Note that you should not remove the previous libraries until the dependent packages are recompiled.
- If you are upgrading a running system, be on the lookout for packages that use **cp** instead of **install** to install files. The latter command is usually safer if the executable or library is already loaded in memory.

10.2.2. Package Management Techniques

The following are some common package management techniques. Before making a decision on a package manager, do some research on the various techniques, particularly the drawbacks of the particular scheme.

10.2.2.1. It is All in My Head!

Yes, this is a package management technique. Some folks do not find the need for a package manager because they know the packages intimately and know what files are installed by each package. Some users also do not need any package management because they plan on rebuilding the entire system when a package is changed.

10.2.2.2. Install in Separate Directories

This is a simplistic package management that does not need any extra package to manage the installations. Each package is installed in a separate directory. For example, package foo-1.1 is installed in `/usr/pkg/foo-1.1` and a symlink is made from `/usr/pkg/foo` to `/usr/pkg/foo-1.1`. When installing a new version foo-1.2, it is installed in `/usr/pkg/foo-1.2` and the previous symlink is replaced by a symlink to the new version.

Environment variables such as `PATH`, `LD_LIBRARY_PATH`, `MANPATH`, `INFOPATH` and `CPPFLAGS` need to be expanded to include `/usr/pkg/foo`. For more than a few packages, this scheme becomes unmanageable.

10.2.2.3. Symlink Style Package Management

This is a variation of the previous package management technique. Each package is installed similar to the previous scheme. But instead of making the symlink, each file is symlinked into the `/usr` hierarchy. This removes the need to expand the environment variables. Though the symlinks can be created by the user to automate the creation, many package managers have been written using this approach. A few of the popular ones include Stow, Epkg, Graft, and Depot.

The installation needs to be faked, so that the package thinks that it is installed in `/usr` though in reality it is installed in the `/usr/pkg` hierarchy. Installing in this manner is not usually a trivial task. For example, consider that you are installing a package libfoo-1.1. The following instructions may not install the package properly:

```
./configure --prefix=/usr/pkg/libfoo/1.1
make
make install
```

The installation will work, but the dependent packages may not link to libfoo as you would expect. If you compile a package that links against libfoo, you may notice that it is linked to `/usr/pkg/libfoo/1.1/lib/libfoo.so.1` instead of `/usr/lib/libfoo.so.1` as you would expect. The correct approach is to use the `DESTDIR` strategy to fake installation of the package. This approach works as follows:

```
./configure --prefix=/usr
make
make DESTDIR=/usr/pkg/libfoo/1.1 install
```

Most packages support this approach, but there are some which do not. For the non-compliant packages, you may either need to manually install the package, or you may find that it is easier to install some problematic packages into `/opt`.

10.2.2.4. Timestamp Based

In this technique, a file is timestamped before the installation of the package. After the installation, a simple use of the **find** command with the appropriate options can generate a log of all the files installed after the timestamp

file was created. A package manager written with this approach is install-log.

Though this scheme has the advantage of being simple, it has two drawbacks. If, during installation, the files are installed with any timestamp other than the current time, those files will not be tracked by the package manager. Also, this scheme can only be used when one package is installed at a time. The logs are not reliable if two packages are being installed on two different consoles.

10.2.2.5. LD_PRELOAD Based

In this approach, a library is preloaded before installation. During installation, this library tracks the packages that are being installed by attaching itself to various executables such as **cp**, **install**, **mv** and tracking the system calls that modify the filesystem. For this approach to work, all the executables need to be dynamically linked without the suid or sgid bit. Preloading the library may cause some unwanted side-effects during installation. Therefore, it is advised that one performs some tests to ensure that the package manager does not break anything and logs all the appropriate files.

10.2.2.6. Creating Package Archives

In this scheme, the package installation is faked into a separate tree as described in the Symlink style package management. After the installation, a package archive is created using the installed files. This archive is then used to install the package either on the local machine or can even be used to install the package on other machines.

This approach is used by most of the package managers found in the commercial distributions. Examples of package managers that follow this approach are RPM (which, incidentally, is required by the *Linux Standard Base Specification*), pkg-utils, Debian's apt, and Gentoo's Portage system. A hint describing how to adopt this style of package management for CLFS systems is located at *http://www.linuxfromscratch.org/hints/downloads/files//fakeroot.txt*.

10.3. About Test Suites, Again

In the final-system build, you are no longer cross-compiling so it is possible to run package testsuites. Some test suites are more important than others. For example, the test suites for the core toolchain packages—GCC, Binutils, and Glibc—are of the utmost importance due to their central role in a properly functioning system. The test suites for GCC and Glibc can take a very long time to complete, especially on slower hardware, but are strongly recommended.

A common issue with running the test suites for Binutils and GCC is running out of pseudo terminals (PTYs). This can result in a high number of failing tests. This may happen for several reasons, but the most likely cause (if you chrooted) is that the host system does not have the `devpts` file system set up correctly. This issue is discussed in greater detail at *http://trac.cross-lfs.org/wiki/faq#no-ptys*.

Sometimes package test suites will fail, but for reasons which the developers are aware of and have deemed non-critical. Consult the logs located at *http://trac.cross-lfs.org/clfs/build-logs/1.0.0rc6/* to verify whether or not these failures are expected. This site is valid for all tests throughout this book.

10.4. Temporary Perl-5.8.8

The Perl package contains the Practical Extraction and Report Language.

10.4.1. Installation of Perl

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

```
patch -Np1 -i ../perl-5.8.8-libc-2.patch
```

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

```
./configure.gnu --prefix=/tools \
   -Dstatic_ext='Data/Dumper IO Fcntl POSIX' -Dcc="gcc"
```

The meaning of the configure option:

`-Dstatic_ext='Data/Dumper IO Fcntl POSIX'`
This tells Perl to build the minimum set of static extensions needed for installing and testing the Glibc and Coreutils packages later in this chapter.

Now we are ready to make our perl utilities:

```
make perl utilities
```

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

Install these tools and their libraries:

```
cp -v perl pod/pod2man /tools/bin
install -dv /tools/lib/perl5/5.8.8
cp -Rv lib/* /tools/lib/perl5/5.8.8
```

Finally, create a necessary symlink:

```
ln -sv /tools/bin/perl /usr/bin
```

Details on this package are located in Section 10.20.2, “Contents of Perl.”

10.5. Linux-Headers-2.6.17.13-09092006

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

10.5.1. Installation of Linux Headers

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

Install the header files that are common to all architectures:

```
install -dv /usr/include
cp -av include/{asm-generic,linux,mtd,scsi,sound} /usr/include/
```

Install the header files that are specific to this architecture:

```
cp -av include/asm-x86_64 /usr/include/asm
```

10.5.2. Contents of Linux-Headers

Installed headers: /usr/include/{asm,asm-generic,linux,mtd,scsi,sound}/*.h

Short Descriptions

`/usr/include/{asm,asm-generic,linux,mtd,scsi,sound}/*.h` Kernel API headers

10.6. Glibc-2.4

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

10.6.1. Installation of Glibc

Note

Some packages outside of CLFS suggest installing GNU libiconv in order to translate data from one encoding to another. The project's home page (*http://www.gnu.org/software/libiconv/*) says "This library provides an `iconv()` implementation, for use on systems which don't have one, or whose implementation cannot convert from/to Unicode." Glibc provides an `iconv()` implementation and can convert from/to Unicode, therefore libiconv is not required on a CLFS system.

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

The following patch fixes an issue that can cause **localdef** to segfault:

```
patch -Np1 -i ../glibc-2.4-localedef_segfault-1.patch
```

The following patch fixes an issue with iconv:

```
patch -Np1 -i ../glibc-2.4-iconv_fix-1.patch
```

The following **sed** fixes a build issue with Glibc. This will prevent **nscd** from trying to link to libraries that don't exist:

```
cp -v nscd/Makefile{,.orig}
sed -e "/nscd_stat.o: sysincludes = # nothing/d" nscd/Makefile.orig > \
    nscd/Makefile
```

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

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

Tell Glibc to install its libraries into `/lib`:

```
echo "slibdir=/lib" >> configparms
```

Prepare Glibc for compilation:

```
../glibc-2.4/configure --prefix=/usr \
    --disable-profile --enable-add-ons --enable-kernel=2.6.0 \
```

```
    --libexecdir=/usr/lib/glibc --libdir=/usr/lib
```

The meaning of the new configure option:

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

Compile the package:

```
make
```

> **Important**
>
> The test suite for Glibc is considered critical. Do not skip it under any circumstance.

Test the results:

```
make -k check >glibc-check-log 2>&1 ; grep Error glibc-check-log
```

The Glibc test suite is highly dependent on certain functions of the host system, in particular the kernel. The posix/annexc test normally fails and you should see `Error 1 (ignored)` in the output. Apart from this, the Glibc test suite is always expected to pass. However, in certain circumstances, some failures are unavoidable. If a test fails because of a missing program (or missing symbolic link), or a segfault, you will see an error code greater than 127 and the details will be in the log. More commonly, tests will fail with `Error 2` - for these, the contents of the corresponding `.out` file, e.g. `posix/annexc.out` may be informative. Here is a list of the most common issues:

- The *math* tests sometimes fail. Certain optimization settings are known to be a factor here.
- If you have mounted the CLFS partition with the *`noatime`* option, the *atime* test will fail. As mentioned in Section 2.4, “Mounting the New Partition”, do not use the *`noatime`* option while building CLFS.
- When running on older and slower hardware, some tests can fail because of test timeouts being exceeded.

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

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

The install will finish by checking that everything is correctly installed. Unfortunately, it will test for a multilib installation. On x86_64 Pure64 this means it will try to test the non-existent 32-bit loader which has a different name from the 64-bit loader (unlike on other 64-bit architectures). We fool it by creating a symlink to the real loader.

```
ln -sv ld-2.4.so /lib/ld-linux.so.2
```

Install the package:

```
make install
```

Now we can remove this symlink. We also need to correct the **/usr/bin/ldd** script - if you look at this, you will see it references not only the 32-bit linker, but also /lib64 where it thinks the 64-bit linker is. The following **sed** will correct this:

```
rm -v /lib/ld-linux.so.2
cp -v /usr/bin/ldd{,.bak}
sed '/RTLDLIST/s%/ld-linux.so.2 /lib64%%' /usr/bin/ldd.bak >/usr/bin/ldd
```

Check the script to make sure the sed worked correctly, then delete the backup.

```
rm -v /usr/bin/ldd.bak
```

10.6.2. Internationalization

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

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

To save time, an alternative to running the previous command (which generates and installs every locale listed in the glibc-2.4/localedata/SUPPORTED file) is to install only those locales that are wanted and needed. This can be achieved by using the **localedef** command. Information on this command is located in the `INSTALL` file in the Glibc source. However, there are a number of locales that are essential in order for the tests of future packages to pass, in particular, the *libstdc++* tests from GCC. The following instructions, instead of the *`install-locales`* target used above, will install the minimum set of locales necessary for the tests to run successfully:

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

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

10.6.3. Configuring Glibc

The `/etc/nsswitch.conf` file needs to be created because, although Glibc provides defaults when this file is missing or corrupt, the Glibc defaults do not work well in a networked environment. The time zone also needs to be configured.

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

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

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

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

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

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

```
tzselect
```

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

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

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

The meaning of the cp option:

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

10.6.4. Configuring The Dynamic Loader

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

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

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

/usr/local/lib
/opt/lib

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

10.6.5. Contents of Glibc

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

Short Descriptions

catchsegv	Can be used to create a stack trace when a program terminates with a segmentation fault
gencat	Generates message catalogues
getconf	Displays the system configuration values for file system specific variables
getent	Gets entries from an administrative database
iconv	Performs character set conversion
iconvconfig	Creates fastloading **iconv** module configuration files
ldconfig	Configures the dynamic linker runtime bindings
ldd	Reports which shared libraries are required by each given program or shared library
lddlibc4	Assists **ldd** with object files
locale	Tells the compiler to enable or disable the use of POSIX locales for built-in operations
localedef	Compiles locale specifications
mtrace	Reads and interprets a memory trace file and displays a summary in human-readable format
nscd	A daemon that provides a cache for the most common name service requests
pcprofiledump	Dumps information generated by PC profiling
pt_chown	A helper program for **grantpt** to set the owner, group and access permissions of a slave pseudo terminal
rpcgen	Generates C code to implement the Remote Procecure Call (RPC) protocol
rpcinfo	Makes an RPC call to an RPC server

sln	A statically linked program that creates symbolic links
sprof	Reads and displays shared object profiling data
tzselect	Asks the user about the location of the system and reports the corresponding time zone description
xtrace	Traces the execution of a program by printing the currently executed function
zdump	The time zone dumper
zic	The time zone compiler
`ld.so`	The helper program for shared library executables
`libBrokenLocale`	Used by programs, such as Mozilla, to solve broken locales
`libSegFault`	The segmentation fault signal handler
`libanl`	An asynchronous name lookup library
`libbsd-compat`	Provides the portability needed in order to run certain Berkey Software Distribution (BSD) programs under Linux
`libc`	The main C library
`libcrypt`	The cryptography library
`libdl`	The dynamic linking interface library
`libg`	A runtime library for **g++**
`libieee`	The Institute of Electrical and Electronic Engineers (IEEE) floating point library
`libm`	The mathematical library
`libmcheck`	Contains code run at boot
`libmemusage`	Used by **memusage** (included in Glibc, but not built in a base CLFS system as it has additional dependencies) to help collect information about the memory usage of a program
`libnsl`	The network services library
`libnss`	The Name Service Switch libraries, containing functions for resolving host names, user names, group names, aliases, services, protocols, etc.
`libpcprofile`	Contains profiling functions used to track the amount of CPU time spent in specific source code lines
`libpthread`	The POSIX threads library
`libresolv`	Contains functions for creating, sending, and interpreting packets to the Internet domain name servers
`librpcsvc`	Contains functions providing miscellaneous RPC services
`librt`	Contains functions providing most of the interfaces specified by the POSIX.1b Realtime Extension

`libthread_db`	Contains functions useful for building debuggers for multi-threaded programs
`libutil`	Contains code for “standard” functions used in many different Unix utilities

10.7. Adjusting the Toolchain

Now we amend the GCC specs file so that it points to the new dynamic linker. A **perl** command accomplishes this:

```
gcc -dumpspecs | \
perl -p -e 's@/tools/lib/ld@/lib/ld@g;' \
     -e 's@\*startfile_prefix_spec:\n@$_/usr/lib/ @g;' > \
     $(dirname $(gcc --print-libgcc-file-name))/specs
```

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

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

Caution

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

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

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

```
[Requesting program interpreter: /lib/ld-linux-x86-64.so.2]
```

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

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

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

```
rm -v dummy.c a.out
```

10.8. Binutils-2.17

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

10.8.1. Installation of Binutils

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

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

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

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

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

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

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

Prepare Binutils for compilation:

```
../binutils-2.17/configure --prefix=/usr \
    --enable-shared --disable-multilib --enable-64-bit-bfd
```

Compile the package:

```
make configure-host
```

Important

During **make configure-host** you may receive the following error message. It is safe to ignore.

```
WARNING: `flex' is missing on your system. You should only
need it if you modified a `.l' file. You may need the `Flex'
package in order for those modifications to take effect. You
can get `Flex' from any GNU archive site.
```

```
make tooldir=/usr
```

The meaning of the make parameter:

`tooldir=/usr`
Normally, the tooldir (the directory where the executables will ultimately be located) is set to `$(exec_prefix)/$(target_alias)`. Because this is a custom system, this target-specific directory

in `/usr` is not required.

> **Important**
>
> The test suite for Binutils is considered critical. Do not skip it under any circumstance.

The ld test suite accesses `/lib64/ld-linux-x86-64.so` in some of the tests. The following symbolic link will allow this:

```
ln -sv /lib /lib64
```

Test the results:

```
make check
```

Now remove the temporary symlink:

```
rm -v /lib64
```

Install the package:

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

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

```
cp -v ../binutils-2.17/include/libiberty.h /usr/include
```

10.8.2. Contents of Binutils

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

Short Descriptions

addr2line	Translates program addresses to file names and line numbers; given an address and the name of an executable, it uses the debugging information in the executable to determine which source file and line number are associated with the address
ar	Creates, modifies, and extracts from archives
as	An assembler that assembles the output of **gcc** into object files
c++filt	Used by the linker to de-mangle C++ and Java symbols and to keep overloaded functions from clashing
gprof	Displays call graph profile data
ld	A linker that combines a number of object and archive files into a single file, relocating their data and tying up symbol references
nm	Lists the symbols occurring in a given object file

objcopy	Translates one type of object file into another
objdump	Displays information about the given object file, with options controlling the particular information to display; the information shown is useful to programmers who are working on the compilation tools
ranlib	Generates an index of the contents of an archive and stores it in the archive; the index lists all of the symbols defined by archive members that are relocatable object files
readelf	Displays information about ELF type binaries
size	Lists the section sizes and the total size for the given object files
strings	Outputs, for each given file, the sequences of printable characters that are of at least the specified length (defaulting to four); for object files, it prints, by default, only the strings from the initializing and loading sections while for other types of files, it scans the entire file
strip	Discards symbols from object files
`libiberty`	Contains routines used by various GNU programs, including **getopt**, **obstack**, **strerror**, **strtol**, and **strtoul**
`libbfd`	The Binary File Descriptor library
`libopcodes`	A library for dealing with opcodes—the “readable text” versions of instructions for the processor; it is used for building utilities like **objdump**.

10.9. GCC-4.1.1

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

10.9.1. Installation of GCC

The following patch fixes the searching of multilib dirs for specs file. The patch standardizes the gcc drivers path iteration functions, ensuring multilib directories are searched in the correct order. This fixes various issues, most noticably with libtool on multilib systems:

```
patch -Np1 -i ../gcc-4.1.1-PR20425-1.patch
```

Apply the following patch so that GCC links to /lib64 instead of /lib:

```
patch -Np1 -i ../gcc-4.1.1-pure64-1.patch
```

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

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

GCC provides a **gccbug** script which detects at compile time whether mktemp is present, and hardcodes the result in a test. If mktemp is not found, the script will fall back to using less random names for temporary files. We will be installing mktemp later, so the following sed will simulate its presence:

```
sed -i 's/@have_mktemp_command@/yes/' gcc/gccbug.in
```

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

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

Prepare GCC for compilation:

```
../gcc-4.1.1/configure --prefix=/usr \
    --libexecdir=/usr/lib --enable-shared --enable-threads=posix \
    --enable-__cxa_atexit --enable-c99 --enable-long-long \
    --enable-clocale=gnu --enable-languages=c,c++ \
    --disable-multilib --disable-libstdcxx-pch
```

Compile the package:

```
make bootstrap
```

> **Important**
>
> The test suite for GCC is considered critical. Do not skip it under any circumstance.

Test the results, but do not stop at errors:

```
make -k check
```

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

```
../gcc-4.1.1/contrib/test_summary
```

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

A few unexpected failures cannot always be avoided. The GCC developers are usually aware of these issues, but have not resolved them yet.

Install the package:

```
make install
```

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

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

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

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

10.9.2. Contents of GCC

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

Short Descriptions

cc	The C compiler
cpp	The C preprocessor; it is used by the compiler to expand the #include, #define, and similar statements in the source files
c++	The C++ compiler
g++	The C++ compiler
gcc	The C compiler
gccbug	A shell script used to help create useful bug reports
gcov	A coverage testing tool; it is used to analyze programs to determine where optimizations will have the most effect
libgcc	Contains run-time support for **gcc**
libmudflap	The libmudflap libraries are used by GCC for instrumenting pointer and array dereferencing operations.

`libstdc++` The standard C++ library

`libsupc++` Provides supporting routines for the C++ programming language

10.10. Coreutils-5.96

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

10.10.1. Installation of Coreutils

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

```
patch -Np1 -i ../coreutils-5.96-uname-1.patch
```

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

```
patch -Np1 -i ../coreutils-5.96-suppress_uptime_kill_su-1.patch
```

Now prepare Coreutils for compilation:

```
CC="gcc ${BUILD64}" ./configure --prefix=/usr
```

Compile the package:

```
make
```

The test suite of Coreutils makes several assumptions about the presence of system users and groups that are not valid within the minimal environment that exists at the moment. Therefore, additional items need to be set up before running the tests. Skip down to “Install the package” if not running the test suite.

Create two `dummy` groups and a `dummy` user:

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

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

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

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

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

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

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

Install the package:

```
make install
```

Move programs to the locations specified by the FHS:

```
mv -v /usr/bin/{cat,chgrp,chmod,chown,cp,date} /bin
```

```
mv -v /usr/bin/{dd,df,echo,false,hostname,ln,ls,mkdir,mknod} /bin
mv -v /usr/bin/{mv,pwd,rm,rmdir,stty,true,uname} /bin
mv -v /usr/bin/chroot /usr/sbin
```

Other Coreutils programs are used by some of the scripts in the CLFS-Bootscripts package. As `/usr` may not be available during the early stages of booting, those binaries need to be on the root partition:

```
mv -v /usr/bin/{[,basename,head,install,nice} /bin
mv -v /usr/bin/{readlink,sleep,sync,test,touch} /bin
ln -svf ../../bin/install /usr/bin
```

10.10.2. Contents of Coreutils

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

Short Descriptions

basename	Strips any path and a given suffix from a file name
cat	Concatenates files to standard output
chgrp	Changes the group ownership of files and directories
chmod	Changes the permissions of each file to the given mode; the mode can be either a symbolic representation of the changes to make or an octal number representing the new permissions
chown	Changes the user and/or group ownership of files and directories
chroot	Runs a command with the specified directory as the `/` directory
cksum	Prints the Cyclic Redundancy Check (CRC) checksum and the byte counts of each specified file
comm	Compares two sorted files, outputting in three columns the lines that are unique and the lines that are common
cp	Copies files
csplit	Splits a given file into several new files, separating them according to given patterns or line numbers and outputting the byte count of each new file
cut	Prints sections of lines, selecting the parts according to given fields or positions
date	Displays the current time in the given format, or sets the system date
dd	Copies a file using the given block size and count, while optionally performing conversions on it
df	Reports the amount of disk space available (and used) on all mounted file systems, or only on the file systems holding the selected files

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

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

dirname Strips the non-directory suffix from a file name

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

echo Displays the given strings

env Runs a command in a modified environment

expand Converts tabs to spaces

expr Evaluates expressions

factor Prints the prime factors of all specified integer numbers

false Does nothing, unsuccessfully; it always exits with a status code indicating failure

fmt Reformats the paragraphs in the given files

fold Wraps the lines in the given files

groups Reports a user's group memberships

head Prints the first ten lines (or the given number of lines) of each given file

hostid Reports the numeric identifier (in hexadecimal) of the host

hostname Reports or sets the name of the host

id Reports the effective user ID, group ID, and group memberships of the current user or specified user

install Copies files while setting their permission modes and, if possible, their owner and group

join Joins the lines that have identical join fields from two separate files

link Creates a hard link with the given name to a file

ln Makes hard links or soft (symbolic) links between files

logname Reports the current user's login name

ls Lists the contents of each given directory

md5sum Reports or checks Message Digest 5 (MD5) checksums

mkdir Creates directories with the given names

mkfifo Creates First-In, First-Outs (FIFOs), a “named pipe” in UNIX parlance, with the given names

mknod Creates device nodes with the given names; a device node is a character special file, a block special file, or a FIFO

mv Moves or renames files or directories

nice Runs a program with modified scheduling priority

nl	Numbers the lines from the given files
nohup	Runs a command immune to hangups, with its output redirected to a log file
od	Dumps files in octal and other formats
paste	Merges the given files, joining sequentially corresponding lines side by side, separated by tab characters
pathchk	Checks if file names are valid or portable
pinky	Is a lightweight finger client; it reports some information about the given users
pr	Paginates and columnates files for printing
printenv	Prints the environment
printf	Prints the given arguments according to the given format, much like the C printf function
ptx	Produces a permuted index from the contents of the given files, with each keyword in its context
pwd	Reports the name of the current working directory
readlink	Reports the value of the given symbolic link
rm	Removes files or directories
rmdir	Removes directories if they are empty
seq	Prints a sequence of numbers within a given range and with a given increment
sha1sum	Prints or checks 160-bit Secure Hash Algorithm 1 (SHA1) checksums
shred	Overwrites the given files repeatedly with complex patterns, making it difficult to recover the data
sleep	Pauses for the given amount of time
sort	Sorts the lines from the given files
split	Splits the given file into pieces, by size or by number of lines
stat	Displays file or filesystem status
stty	Sets or reports terminal line settings
sum	Prints checksum and block counts for each given file
sync	Flushes file system buffers; it forces changed blocks to disk and updates the super block
tac	Concatenates the given files in reverse
tail	Prints the last ten lines (or the given number of lines) of each given file
tee	Reads from standard input while writing both to standard output and to the given files
test or **[**	Compares values and checks file types
touch	Changes file timestamps, setting the access and modification times of the given files to the current time; files that do not exist are created with zero length

tr	Translates, squeezes, and deletes the given characters from standard input
true	Does nothing, successfully; it always exits with a status code indicating success
tsort	Performs a topological sort; it writes a completely ordered list according to the partial ordering in a given file
tty	Reports the file name of the terminal connected to standard input
uname	Reports system information
unexpand	Converts spaces to tabs
uniq	Discards all but one of successive identical lines
unlink	Removes the given file
users	Reports the names of the users currently logged on
vdir	Is the same as **ls -l**
wc	Reports the number of lines, words, and bytes for each given file, as well as a total line when more than one file is given
who	Reports who is logged on
whoami	Reports the user name associated with the current effective user ID
yes	Repeatedly outputs “y” or a given string until killed

10.11. Iana-Etc-2.10

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

10.11.1. Installation of Iana-Etc

The following command converts the raw data provided by IANA into the correct formats for the `/etc/protocols` and `/etc/services` data files:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

10.11.2. Contents of Iana-Etc

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

Short Descriptions

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

10.12. M4-1.4.4

The M4 package contains a macro processor.

10.12.1. Installation of M4

Prepare M4 for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.12.2. Contents of M4

Installed program: m4

Short Descriptions

m4 copies the given files while expanding the macros that they contain. These macros are either built-in or user-defined and can take any number of arguments. Besides performing macro expansion, **m4** has built-in functions for including named files, running Unix commands, performing integer arithmetic, manipulating text, recursion, etc. The **m4** program can be used either as a front-end to a compiler or as a macro processor in its own right.

10.13. Bison-2.3

The Bison package contains a parser generator.

10.13.1. Installation of Bison

Prepare Bison for compilation:

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

The configure system causes bison to be built without support for internationalization of error messages if a **bison** program is not already in $PATH. The following addition will correct this:

```
echo '#define YYENABLE_NLS 1' >> config.h
```

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.13.2. Contents of Bison

Installed programs: bison and yacc
Installed library: liby.a

Short Descriptions

bison Generates, from a series of rules, a program for analyzing the structure of text files; Bison is a replacement for Yacc (Yet Another Compiler Compiler)

yacc A wrapper for **bison**, meant for programs that still call **yacc** instead of **bison**; it calls **bison** with the `-y` option

`liby.a` The Yacc library containing implementations of Yacc-compatible *yyerror* and *main* functions; this library is normally not very useful, but POSIX requires it

10.14. Ncurses-5.5

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

10.14.1. Installation of Ncurses

Prepare Ncurses for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Move the Ncurses static libraries to the proper location:

```
mv -v /lib/lib{panel,menu,form,ncurses,ncurses++,curses}.a /usr/lib
```

Create symlinks in `/usr/lib`:

```
rm -v /lib/lib{ncurses,menu,panel,form,curses}.so
ln -svf ../../lib/libncurses.so.5 /usr/lib/libcurses.so
ln -svf ../../lib/libncurses.so.5 /usr/lib/libncurses.so
ln -svf ../../lib/libmenu.so.5 /usr/lib/libmenu.so
ln -svf ../../lib/libpanel.so.5 /usr/lib/libpanel.so
ln -svf ../../lib/libform.so.5 /usr/lib/libform.so
```

Give the Ncurses libraries execute permissions:

```
chmod -v 755 /lib/lib{panel,menu,form,ncurses}.so.5.5
```

10.14.2. Contents of Ncurses

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

Short Descriptions

captoinfo	Converts a termcap description into a terminfo description
clear	Clears the screen, if possible

infocmp	Compares or prints out terminfo descriptions
infotocap	Converts a terminfo description into a termcap description
reset	Reinitializes a terminal to its default values
tack	The terminfo action checker; it is mainly used to test the accuracy of an entry in the terminfo database
tic	The terminfo entry-description compiler that translates a terminfo file from source format into the binary format needed for the ncurses library routines. A terminfo file contains information on the capabilities of a certain terminal
toe	Lists all available terminal types, giving the primary name and description for each
tput	Makes the values of terminal-dependent capabilities available to the shell; it can also be used to reset or initialize a terminal or report its long name
tset	Can be used to initialize terminals
`libcurses`	A link to `libncurses`
`libncurses`	Contains functions to display text in many complex ways on a terminal screen; a good example of the use of these functions is the menu displayed during the kernel's **make menuconfig**
`libform`	Contains functions to implement forms
`libmenu`	Contains functions to implement menus
`libpanel`	Contains functions to implement panels

10.15. Procps-3.2.6

The Procps package contains programs for monitoring processes.

10.15.1. Installation of Procps

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

10.15.2. Contents of Procps

Installed programs: free, kill, pgrep, pkill, pmap, ps, pwdx, skill, slabtop, snice, sysctl, tload, top, uptime, vmstat, w, and watch
Installed library: libproc.so

Short Descriptions

free	Reports the amount of free and used memory (both physical and swap memory) in the system
kill	Sends signals to processes
pgrep	Looks up processes based on their name and other attributes
pkill	Signals processes based on their name and other attributes
pmap	Reports the memory map of the given process
ps	Lists the current running processes
pwdx	Reports the current working directory of a process
skill	Sends signals to processes matching the given criteria
slabtop	Displays detailed kernel slab cache information in real time
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 a list of the most CPU intensive processes; it provides an ongoing look at processor activity in real time
uptime	Reports how long the system has been running, how many users are logged on, and the system load averages

vmstat Reports virtual memory statistics, giving information about processes, memory, paging, block Input/Output (IO), traps, and CPU activity

w Shows which users are currently logged on, where, and since when

watch Runs a given command repeatedly, displaying the first screen-full of its output; this allows a user to watch the output change over time

`libproc` Contains the functions used by most programs in this package

10.16. Sed-4.1.5

The Sed package contains a stream editor.

10.16.1. Installation of Sed

Prepare Sed for compilation:

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

The meaning of the new configure option:

`--enable-html`
This option tells Sed to build and install its HTML documentation.

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.16.2. Contents of Sed

Installed program: sed

Short Descriptions

sed Filters and transforms text files in a single pass

10.17. Libtool-1.5.22

The Libtool package contains the GNU generic library support script. It wraps the complexity of using shared libraries in a consistent, portable interface.

10.17.1. Installation of Libtool

Prepare Libtool for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.17.2. Contents of Libtool

Installed programs: libtool and libtoolize
Installed libraries: libltdl.[a,so]

Short Descriptions

libtool	Provides generalized library-building support services
libtoolize	Provides a standard way to add **libtool** support to a package
`libltdl`	Hides the various difficulties of dlopening libraries

10.18. Flex-2.5.33

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

10.18.1. Installation of Flex

Prepare Flex for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

There are some packages that expect to find the `lex` library in `/usr/lib`. Create a symlink to account for this:

```
ln -sv libfl.a /usr/lib/libl.a
```

A few programs do not know about **flex** yet and try to run its predecessor, **lex**. To support those programs, create a wrapper script named `lex` that calls `flex` in **lex** emulation mode:

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Begin /usr/bin/lex

exec /usr/bin/flex -l "$@"

# End /usr/bin/lex
EOF
chmod -v 755 /usr/bin/lex
```

10.18.2. Contents of Flex

Installed programs: flex and lex
Installed library: libfl.a

Short Descriptions

flex	A tool for generating programs that recognize patterns in text; it allows for the versatility to specify the rules for pattern-finding, eradicating the need to develop a specialized program
lex	A script that runs **flex** in **lex** emulation mode
`libfl.a`	The `flex` library

10.19. IPRoute2-2.6.16-060323

The IPRoute2 package contains programs for basic and advanced IPV4-based networking.

10.19.1. Installation of IPRoute2

The **arpd** binary included in this package is dependent on Berkeley DB. Because **arpd** is not a very common requirement on a base Linux system, remove the dependency on Berkeley DB by applying the **sed** command below. If the **arpd** binary is needed, instructions for compiling Berkeley DB can be found in the BLFS Book at *http://www.linuxfromscratch.org/blfs/view/svn/server/databases.html#db*.

```
sed -i '/^TARGETS/s@arpd@@g' misc/Makefile
```

Compile the package:

```
make SBINDIR=/sbin
```

The meaning of the make option:

SBINDIR=/sbin
: This ensures that the IPRoute2 binaries will install into `/sbin`. This is the correct location according to the FHS, because some of the IPRoute2 binaries are used by the CLFS-Bootscripts package.

This package does not come with a test suite.

Install the package:

```
make SBINDIR=/sbin install
```

10.19.2. Contents of IPRoute2

Installed programs: ctstat (link to lnstat), ifcfg, ifstat, ip, lnstat, nstat, routef, routel, rtacct, rtmon, rtpr, rtstat (link to lnstat), ss, and tc

Short Descriptions

ctstat Connection status utility

ifcfg A shell script wrapper for the **ip** command

ifstat Shows the interface statistics, including the amount of transmitted and received packets by interface

ip The main executable. It has several different functions:

ip link *[device]* allows users to look at the state of devices and to make changes

ip addr allows users to look at addresses and their properties, add new addresses, and delete old ones

ip neighbor allows users to look at neighbor bindings and their properties, add new neighbor entries, and delete old ones

ip rule allows users to look at the routing policies and change them

ip route allows users to look at the routing table and change routing table rules

ip tunnel allows users to look at the IP tunnels and their properties, and change them

ip maddr allows users to look at the multicast addresses and their properties, and change them

ip mroute allows users to set, change, or delete the multicast routing

ip monitor allows users to continously monitor the state of devices, addresses and routes

lnstat Provides Linux network statistics. It is a generalized and more feature-complete replacement for the old **rtstat** program

nstat Shows network statistics

routef A component of **ip route**. This is for flushing the routing tables

routel A component of **ip route**. This is for listing the routing tables

rtacct Displays the contents of `/proc/net/rt_acct`

rtmon Route monitoring utility

rtpr Converts the output of **ip -o** back into a readable form

rtstat Route status utility

ss Similar to the **netstat** command; shows active connections

tc Traffic Controlling Executable; this is for Quality Of Service (QOS) and Class Of Service (COS) implementations

tc qdisc allows users to setup the queueing discipline

tc class allows users to setup classes based on the queuing discipline scheduling

tc estimator allows users to estimate the network flow into a network

tc filter allows users to setup the QOS/COS packet filtering

tc policy allows users to setup the QOS/COS policies

10.20. Perl-5.8.8

The Perl package contains the Practical Extraction and Report Language.

10.20.1. Installation of Perl

The following sed causes `DynaLoader.a` to be built with -fPIC so it can be linked into a shared library later:

```
sed -i -e "s@pldlflags=''@pldlflags=\"\$cccdlflags\"@g" \
    -e "s@static_target='static'@static_target='static_pic'@g" Makefile.SH
```

Note

If you are following the boot method you will need to enable the loopback device as well as set a hostname for some of the tests:

```
ip link set lo up
hostname clfs
```

Before starting to configure, create a basic `/etc/hosts` file which will be referenced in one of Perl's configuration files as well as being used used by the testsuite if you run that:

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

To have full control over the way Perl is set up, you can run the interactive **Configure** script and hand-pick the way this package is built. If you prefer instead to use the defaults that Perl auto-detects, prepare Perl for compilation with:

```
./configure.gnu --prefix=/usr \
   -Dman1dir=/usr/share/man/man1 \
   -Dman3dir=/usr/share/man/man3 \
   -Dpager="/bin/less -isR" \
   -Dusethreads
```

The meaning of the configure option:

`-Dpager="/bin/less -isR"`
This corrects an error in the way that **perldoc** invokes the **less** program.

`-Dman1dir=/usr/share/man/man1 -Dman3dir=/usr/share/man/man3`
Since Groff is not installed yet, **configure.gnu** thinks that we do not want man pages for Perl. Issuing these parameters overrides this decision.

`-Dusethreads`
This tells Perl to use threads.

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.20.2. Contents of Perl

Installed programs: a2p, c2ph, cpan, dprofpp, enc2xs, find2perl, h2ph, h2xs, instmodsh, libnetcfg, perl, perl5.8.8 (link to perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, prove, psed (link to s2p), pstruct (link to c2ph), s2p, splain, and xsubpp
Installed libraries: Several hundred which cannot all be listed here

Short Descriptions

a2p	Translates awk to Perl
c2ph	Dumps C structures as generated from **cc -g -S**
cpan	Shell script that provides a command interface to CPAN.pm
dprofpp	Displays Perl profile data
enc2xs	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`
instmodsh	A shell script for examining installed Perl modules, and can even create a tarball from an installed module
perl	Combines some of the best features of C, **sed**, **awk** and **sh** into a single swiss-army-knife language
perl5.8.8	A hard link to **perl**
perlbug	Used to generate bug reports about Perl, or the modules that come with it, and mail them
perlcc	Generates executables from Perl programs
perldoc	Displays a piece of documentation in pod format that is embedded in the Perl installation tree or in a Perl script
perlivp	The Perl Installation Verification Procedure; it can be used to verify that Perl and its libraries have been installed correctly
piconv	A Perl version of the character encoding converter **iconv**
pl2pm	A rough tool for converting Perl4 `.pl` files to Perl5 `.pm` modules

pod2html	Converts files from pod format to HTML format
pod2latex	Converts files from pod format to LaTeX format
pod2man	Converts pod data to formatted *roff input
pod2text	Converts pod data to formatted ASCII text
pod2usage	Prints usage messages from embedded pod docs in files
podchecker	Checks the syntax of pod format documentation files
podselect	Displays selected sections of pod documentation
prove	A command-line tool for running tests against Test::Harness
psed	A Perl version of the stream editor **sed**
pstruct	Dumps C structures as generated from **cc -g -S** stabs
s2p	Translates **sed** to Perl
splain	Is used to force verbose warning diagnostics in Perl
xsubpp	Converts Perl XS code into C code

10.21. Readline-5.1

The Readline package is a set of libraries that offers command-line editing and history capabilities.

10.21.1. Installation of Readline

The following patch contains updates from the maintainer. The maintainer of Readline only releases these patches to fix serious issues.

```
patch -Np1 -i ../readline-5.1-fixes-3.patch
```

Prepare Readline for compilation:

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

Compile the package:

```
make SHLIB_XLDFLAGS=-lncurses
```

The meaning of the make option:

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

This package does not come with a test suite.

Install the package:

```
make install
```

Give Readline's dynamic libraries more appropriate permissions:

```
chmod -v 755 /lib/lib{readline,history}.so*
```

Now move the static libraries to a more appropriate location:

```
mv -v /lib/lib{readline,history}.a /usr/lib
```

Next, remove the `.so` files in `/lib` and relink them into `/usr/lib`.

```
rm -v /lib/lib{readline,history}.so
ln -svf ../../lib/libreadline.so.5 /usr/lib/libreadline.so
ln -svf ../../lib/libhistory.so.5 /usr/lib/libhistory.so
```

10.21.2. Contents of Readline

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

Short Descriptions

`libhistory` Provides a consistent user interface for recalling lines of history

`libreadline` Aids in the consistency of user interface across discrete programs that need to provide a command line interface

10.22. Zlib-1.2.3

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

10.22.1. Installation of Zlib

This patch will add -fPIC to our build and allow us to build a static and shared library at the same time:

```
patch -Np1 -i ../zlib-1.2.3-fPIC-1.patch
```

Prepare Zlib for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

The previous command installed a `.so` file in `/usr/lib`. We will move it into `/lib` and then relink it to `/usr/lib`:

```
mv -v /usr/lib/libz.so.* /lib
ln -svf ../../lib/libz.so.1 /usr/lib/libz.so
```

Now we fix the permissions on the static library:

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

10.22.2. Contents of Zlib

Installed libraries: libz.[a,so]

Short Descriptions

`libz` Contains compression and decompression functions used by some programs

10.23. Autoconf-2.59

The Autoconf package contains programs for producing shell scripts that can automatically configure source code.

10.23.1. Installation of Autoconf

Prepare Autoconf for compilation:

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

Compile the package:

```
make
```

To test the results, issue: **make check**. 2 tests are skipped that use Automake. For full test coverage, Autoconf can be re-tested after Automake has been installed.

Install the package:

```
make install
```

10.23.2. Contents of Autoconf

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

Short Descriptions

autoconf	Produces shell scripts that automatically configure software source code packages to adapt to many kinds of Unix-like systems. The configuration scripts it produces are independent—running them does not require the **autoconf** program.
autoheader	A tool for creating template files of C *#define* statements for configure to use
autom4te	A wrapper for the M4 macro processor
autoreconf	Automatically runs **autoconf**, **autoheader**, **aclocal**, **automake**, **gettextize**, and **libtoolize** in the correct order to save time when changes are made to **autoconf** and **automake** template files
autoscan	Helps to create a `configure.in` file for a software package; it examines the source files in a directory tree, searching them for common portability issues, and creates a `configure.scan` file that serves as as a preliminary `configure.in` file for the package
autoupdate	Modifies a `configure.in` file that still calls **autoconf** macros by their old names to use the current macro names
ifnames	Helps when writing `configure.in` files for a software package; it prints the identifiers that the package uses in C preprocessor conditionals. If a package has already been set up to have some portability, this program can help determine what **configure** needs to check for. It

can also fill in gaps in a `configure.in` file generated by **autoscan**

10.24. Automake-1.9.6

The Automake package contains programs for generating Makefiles for use with Autoconf.

10.24.1. Installation of Automake

Prepare Automake for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.24.2. Contents of Automake

Installed programs: acinstall, aclocal, aclocal-1.9, automake, automake-1.9, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, symlink-tree, and ylwrap

Short Descriptions

acinstall	A script that installs aclocal-style M4 files
aclocal	Generates `aclocal.m4` files based on the contents of `configure.in` files
aclocal-1.9	A hard link to **aclocal**
automake	A tool for automatically generating `Makefile.in` files from `Makefile.am` files. To create all the `Makefile.in` files for a package, run this program in the top-level directory. By scanning the `configure.in` file, it automatically finds each appropriate `Makefile.am` file and generates the corresponding `Makefile.in` file
automake-1.9	A hard link to **automake**
compile	A wrapper for compilers
config.guess	A script that attempts to guess the canonical triplet for the given build, host, or target architecture
config.sub	A configuration validation subroutine script
depcomp	A script for compiling a program so that dependency information is generated in addition to the desired output
elisp-comp	Byte-compiles Emacs Lisp code
install-sh	A script that installs a program, script, or data file

mdate-sh	A script that prints the modification time of a file or directory
missing	A script acting as a common stub for missing GNU programs during an installation
mkinstalldirs	A script that creates a directory tree
py-compile	Compiles a Python program
symlink-tree	A script to create a symlink tree of a directory tree
ylwrap	A wrapper for **lex** and **yacc**

10.25. Bash-3.1

The Bash package contains the Bourne-Again SHell.

10.25.1. Installation of Bash

If you downloaded the Bash documentation tarball and wish to install HTML documentation, issue the following commands:

```
tar -xvf ../bash-doc-3.1.tar.gz &&
sed -i "s|htmldir = @htmldir@|htmldir = /usr/share/doc/bash-3.1|" \
    Makefile.in
```

The following patch contains updates from the maintainer. The maintainer of Bash only releases these patches to fix serious issues.

```
patch -Np1 -i ../bash-3.1-fixes-8.patch
```

Prepare Bash for compilation:

```
./configure --prefix=/usr --bindir=/bin \
    --without-bash-malloc --with-installed-readline
```

The meaning of the configure option:

`--with-installed-readline`
This option tells Bash to use the `readline` library that is already installed on the system rather than using its own readline version.

Compile the package:

```
make
```

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

Install the package:

```
make install
```

Run the newly compiled **bash** program (replacing the one that is currently being executed):

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

Note

The parameters used make the **bash** process an interactive login shell and continue to disable hashing so that new programs are found as they become available.

10.25.2. Contents of Bash

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

Short Descriptions

bash A widely-used command interpreter; it performs many types of expansions and substitutions on a given command line before executing it, thus making this interpreter a powerful tool

bashbug A shell script to help the user compose and mail standard formatted bug reports concerning **bash**

sh A symlink to the **bash** program; when invoked as **sh**, **bash** tries to mimic the startup behavior of historical versions of **sh** as closely as possible, while conforming to the POSIX standard as well

10.26. Bzip2-1.0.3

The Bzip2 package contains programs for compressing and decompressing files. Compressing text files with **bzip2** yields a much better compression percentage than with the traditional **gzip**.

10.26.1. Installation of Bzip2

Apply a patch to install the documentation for this package:

```
patch -Np1 -i ../bzip2-1.0.3-install_docs-1.patch
```

The **bzgrep** command does not escape '|' and '&' in filenames passed to it. This allows arbitrary commands to be executed with the privileges of the user running **bzgrep**. Apply the following patch to address this:

```
patch -Np1 -i ../bzip2-1.0.3-bzgrep_security-1.patch
```

The **bzdiff** script depends on tempfile. We remove that dependency with this patch:

```
patch -Np1 -i ../bzip2-1.0.3-remove_tempfile-1.patch
```

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

Recompile the package using a non-shared library and test it:

```
make
```

Note

If reinstalling Bzip2, perform **`rm -vf /usr/bin/bz*`** first, otherwise the following **make install** will fail.

Install the programs:

```
make install
```

Install the shared **bzip2** binary into the `/bin` directory, make some necessary symbolic links, and clean up:

```
cp -v bzip2-shared /bin/bzip2
cp -av libbz2.so* /lib
ln -sv ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm -v /usr/bin/{bunzip2,bzcat,bzip2}
ln -sv bzip2 /bin/bunzip2
ln -sv bzip2 /bin/bzcat
```

10.26.2. Contents of Bzip2

Installed programs: bunzip2 (link to bzip2), bzcat (link to bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless, and bzmore
Installed libraries: libbz2.a, libbz2.so (link to libbz2.so.1.0), libbz2.so.1.0 (link to libbz2.so.1.0.3), and libbz2.so.1.0.3

Short Descriptions

bunzip2	Decompresses bzipped files
bzcat	Decompresses to standard output
bzcmp	Runs **cmp** on bzipped files
bzdiff	Runs **diff** on bzipped files
bzgrep	Runs **grep** on bzipped files
bzegrep	Runs **egrep** on bzipped files
bzfgrep	Runs **fgrep** on bzipped files
bzip2	Compresses files using the Burrows-Wheeler block sorting text compression algorithm with Huffman coding; the compression rate is better than that achieved by more conventional compressors using “Lempel-Ziv” algorithms, like **gzip**
bzip2recover	Tries to recover data from damaged bzipped files
bzless	Runs **less** on bzipped files
bzmore	Runs **more** on bzipped files
`libbz2*`	The library implementing lossless, block-sorting data compression, using the Burrows-Wheeler algorithm

10.27. Diffutils-2.8.7

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

10.27.1. Installation of Diffutils

Prepare Diffutils for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

10.27.2. Contents of Diffutils

Installed programs: cmp, diff, diff3, and sdiff

Short Descriptions

cmp Compares two files and reports whether or in which bytes they differ

diff Compares two files or directories and reports which lines in the files differ

diff3 Compares three files line by line

sdiff Merges two files and interactively outputs the results

10.28. E2fsprogs-1.39

The E2fsprogs package contains the utilities for handling the `ext2` file system. It also supports the `ext3` journaling file system.

10.28.1. Installation of E2fsprogs

The E2fsprogs documentation recommends that the package be built in a subdirectory of the source tree:

```
mkdir -v build
cd build
```

Prepare E2fsprogs for compilation:

```
../configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs --disable-evms
```

The meaning of the configure options:

`--with-root-prefix=""`
Certain programs (such as the **e2fsck** program) are considered essential programs. When, for example, `/usr` is not mounted, these programs still need to be available. They belong in directories like `/lib` and `/sbin`. If this option is not passed to E2fsprogs' configure, the programs are installed into the `/usr` directory.

`--enable-elf-shlibs`
This creates the shared libraries which some programs in this package use.

`--disable-evms`
This disables the building of the Enterprise Volume Management System (EVMS) plugin. This plugin is not up-to-date with the latest EVMS internal interfaces and EVMS is not installed as part of a base CLFS system, so the plugin is not required. See the EVMS website at *http://evms.sourceforge.net/* for more information regarding EVMS.

Compile the package:

```
make
```

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

Install the binaries and documentation:

```
make install
```

Install the shared libraries:

```
make install-libs
```

10.28.2. Contents of E2fsprogs

Installed programs: badblocks, blkid, chattr, compile_et, debugfs, dumpe2fs, e2fsck, e2image, e2label, filefrag, findfs, fsck, fsck.ext2, fsck.ext3, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mklost+found, resize2fs, tune2fs, and uuidgen.
Installed libraries: libblkid.[a,so], libcom_err.[a,so], libe2p.[a,so], libext2fs.[a,so], libss.[a,so], and libuuid.[a,so]

Short Descriptions

badblocks	Searches a device (usually a disk partition) for bad blocks
blkid	A command line utility to locate and print block device attributes
chattr	Changes the attributes of files on an `ext2` file system; it also changes `ext3` file systems, the journaling version of `ext2` file systems
compile_et	An error table compiler; it converts a table of error-code names and messages into a C source file suitable for use with the `com_err` library
debugfs	A file system debugger; it can be used to examine and change the state of an `ext2` file system
dumpe2fs	Prints the super block and blocks group information for the file system present on a given device
e2fsck	Is used to check, and optionally repair `ext2` file systems and `ext3` file systems
e2image	Is used to save critical `ext2` file system data to a file
e2label	Displays or changes the file system label on the `ext2` file system present on a given device
filefrag	Reports on how badly fragmented a particular file might be
findfs	Finds a file system by label or Universally Unique Identifier (UUID)
fsck	Is used to check, and optionally repair, file systems
fsck.ext2	By default checks `ext2` file systems
fsck.ext3	By default checks `ext3` file systems
logsave	Saves the output of a command in a log file
lsattr	Lists the attributes of files on a second extended file system
mk_cmds	Converts a table of command names and help messages into a C source file suitable for use with the `libss` subsystem library
mke2fs	Creates an `ext2` or `ext3` file system on the given device
mkfs.ext2	By default creates `ext2` file systems
mkfs.ext3	By default creates `ext3` file systems
mklost+found	Used to create a `lost+found` directory on an `ext2` file system; it pre-allocates disk blocks to this directory to lighten the task of **e2fsck**
resize2fs	Can be used to enlarge or shrink an `ext2` file system

tune2fs	Adjusts tunable file system parameters on an `ext2` file system
uuidgen	Creates new UUIDs. Each new UUID can reasonably be considered unique among all UUIDs created, on the local system and on other systems, in the past and in the future
`libblkid`	Contains routines for device identification and token extraction
`libcom_err`	The common error display routine
`libe2p`	Used by **dumpe2fs**, **chattr**, and **lsattr**
`libext2fs`	Contains routines to enable user-level programs to manipulate an `ext2` file system
`libss`	Used by **debugfs**
`libuuid`	Contains routines for generating unique identifiers for objects that may be accessible beyond the local system

10.29. File-4.17

The File package contains a utility for determining the type of a given file or files.

10.29.1. Installation of File

Prepare File for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

10.29.2. Contents of File

Installed programs: file
Installed library: libmagic.[a,so]

Short Descriptions

file	Tries to classify each given file; it does this by performing several tests—file system tests, magic number tests, and language tests
`libmagic`	Contains routines for magic number recognition, used by the **file** program

10.30. Findutils-4.2.27

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

10.30.1. Installation of Findutils

Prepare Findutils for compilation:

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

The meaning of the configure options:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

The **find** program is used by some of the scripts in the CLFS-Bootscripts package. As `/usr` may not be available during the early stages of booting, the **find** binary needs to be on the root partition:

```
mv -v /usr/bin/find /bin
```

The **updatedb** script needs to be modified to point to the new location for **find**:

```
sed -i 's@find:=${BINDIR}@find:=/bin@' /usr/bin/updatedb
```

10.30.2. Contents of Findutils

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

Short Descriptions

bigram	Was formerly used to produce **locate** databases
code	Was formerly used to produce **locate** databases; it is the ancestor of **frcode**.
find	Searches given directory trees for files matching the specified criteria

frcode Is called by **updatedb** to compress the list of file names; it uses front-compression, reducing the database size by a factor of four to five.

locate Searches through a database of file names and reports the names that contain a given string or match a given pattern

updatedb Updates the **locate** database; it scans the entire file system (including other file systems that are currently mounted, unless told not to) and puts every file name it finds into the database

xargs Can be used to apply a given command to a list of files

10.31. Gawk-3.1.5

The Gawk package contains programs for manipulating text files.

10.31.1. Installation of Gawk

Patch Gawk to fix a bug which causes it to segfault when invoked on a non-existent file:

```
patch -Np1 -i ../gawk-3.1.5-segfault_fix-1.patch
```

Prepare Gawk for compilation:

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

Now fix an issue that will cause the Gettext testsuite to fail:

```
echo '#define HAVE_LC_MESSAGES 1' >> config.h
```

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.31.2. Contents of Gawk

Installed programs: awk (link to gawk), gawk, gawk-3.1.5, grcat, igawk, pgawk, pgawk-3.1.5, and pwcat

Short Descriptions

awk	A link to **gawk**
gawk	A program for manipulating text files; it is the GNU implementation of **awk**
gawk-3.1.5	A hard link to **gawk**
grcat	Dumps the group database `/etc/group`
igawk	Gives **gawk** the ability to include files
pgawk	The profiling version of **gawk**
pgawk-3.1.5	Hard link to **pgawk**
pwcat	Dumps the password database `/etc/passwd`

10.32. Gettext-0.14.5

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

10.32.1. Installation of Gettext

Gettext fails to pass `-fPIC` to the testsuite for its internal libtool. On x86_64 it is not possible to link statically linked archives compiled without -fPIC against shared objects and the suite stops without running the main body of the tests. Use the following **sed** to add this into the `TESTS_ENVIRONMENT` parameters:

```
sed -i \
    "s/CC='@CC@' CFLAGS='@CFLAGS@'/CC='@CC@' CFLAGS='@CFLAGS@ -fPIC'/" \
    autoconf-lib-link/tests/Makefile.in
```

Prepare Gettext for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.32.2. Contents of Gettext

Installed programs: autopoint, config.charset, config.rpath, envsubst, gettext, gettext.sh, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, and xgettext
Installed libraries: libasprintf.[a,so], libgettextlib.so, libgettextpo.[a,so], and libgettextsrc.so

Short Descriptions

autopoint	Copies standard Gettext infrastructure files into a source package
config.charset	Outputs a system-dependent table of character encoding aliases
config.rpath	Outputs a system-dependent set of variables, describing how to set the runtime search path of shared libraries in an executable
envsubst	Substitutes environment variables in shell format strings
gettext	Translates a natural language message into the user's language by looking up the translation in a message catalog
gettext.sh	Primarily serves as a shell function library for gettext

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 a translation catalog
msggrep	Extracts all messages of a translation catalog that match a given pattern or belong to some given source files
msginit	Creates a new `.po` file, initializing the meta information with values from the user's environment
msgmerge	Combines two raw translations into a single file
msgunfmt	Decompiles a binary message catalog into raw translation text
msguniq	Unifies duplicate translations in a translation catalog
ngettext	Displays native language translations of a textual message whose grammatical form depends on a number
xgettext	Extracts the translatable message lines from the given source files to make the first translation template
`libasprintf`	defines the *autosprintf* class, which makes C formatted output routines usable in C++ programs, for use with the <*string*> strings and the <*iostream*> streams
`libgettextlib`	a private library containing common routines used by the various Gettext programs; these are not intended for general use
`libgettextpo`	Used to write specialized programs that process `.po` files; this library is used when the standard applications shipped with Gettext (such as **msgcomm**, **msgcmp**, **msgattrib**, and **msgen**) will not suffice
`libgettextsrc`	A private library containing common routines used by the various Gettext programs; these are not intended for general use

10.33. Grep-2.5.1a

The Grep package contains programs for searching through files.

10.33.1. Installation of Grep

Prepare Grep for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.33.2. Contents of Grep

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

Short Descriptions

egrep Prints lines matching an extended regular expression

fgrep Prints lines matching a list of fixed strings

grep Prints lines matching a basic regular expression

10.34. Groff-1.19.2

The Groff package contains programs for processing and formatting text.

10.34.1. Installation of Groff

Groff expects the environment variable PAGE to contain the default paper size. For users in the United States, *PAGE=letter* is appropriate. Elsewhere, *PAGE=A4* may be more suitable.

Prepare Groff for compilation:

```
PAGE=[paper_size] ./configure --prefix=/usr
```

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

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

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

10.34.2. Contents of Groff

Installed programs: addftinfo, afmtodit, eqn, eqn2graph, gdiffmk, geqn (link to eqn), grap2graph, grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (link to tbl), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pdfroff, pfbtops, pic, pic2graph, post-grohtml, pre-grohtml, refer, soelim, tbl, tfmtodit, troff, and zsoelim (link to soelim)

Short Descriptions

addftinfo	Reads a troff font file and adds some additional font-metric information that is used by the **groff** system
afmtodit	Creates a font file for use with **groff** and **grops**
eqn	Compiles descriptions of equations embedded within troff input files into commands that are understood by **troff**
eqn2graph	Converts a troff EQN (equation) into a cropped image
gdiffmk	Marks differences between groff/nroff/troff files
geqn	A link to **eqn**

grap2graph	Converts a grap diagram into a cropped bitmap image
grn	A **groff** preprocessor for gremlin files
grodvi	A driver for **groff** that produces TeX dvi format
groff	A front-end to the groff document formatting system; normally, it runs the **troff** program and a post-processor appropriate for the selected device
groffer	Displays groff files and man pages on X and tty terminals
grog	Reads files and guesses which of the **groff** options `-e`, `-man`, `-me`, `-mm`, `-ms`, `-p`, `-s`, and `-t` are required for printing files, and reports the **groff** command including those options
grolbp	Is a **groff** driver for Canon CAPSL printers (LBP-4 and LBP-8 series laser printers)
grolj4	Is a driver for **groff** that produces output in PCL5 format suitable for an HP LaserJet 4 printer
grops	Translates the output of GNU **troff** to PostScript
grotty	Translates the output of GNU **troff** into a form suitable for typewriter-like devices
gtbl	A link to **tbl**
hpftodit	Creates a font file for use with **groff -Tlj4** from an HP-tagged font metric file
indxbib	Creates an inverted index for the bibliographic databases with a specified file for use with **refer**, **lookbib**, and **lkbib**
lkbib	Searches bibliographic databases for references that contain specified keys and reports any references found
lookbib	Prints a prompt on the standard error (unless the standard input is not a terminal), reads a line containing a set of keywords from the standard input, searches the bibliographic databases in a specified file for references containing those keywords, prints any references found on the standard output, and repeats this process until the end of input
mmroff	A simple preprocessor for **groff**
neqn	Formats equations for American Standard Code for Information Interchange (ASCII) output
nroff	A script that emulates the **nroff** command using **groff**
pdfroff	Creates pdf documents using groff
pfbtops	Translates a PostScript font in `.pfb` format to ASCII
pic	Compiles descriptions of pictures embedded within troff or TeX input files into commands understood by TeX or **troff**
pic2graph	Converts a PIC diagram into a cropped image
post-grohtml	Translates the output of GNU **troff** to HTML
pre-grohtml	Translates the output of GNU **troff** to HTML
refer	Copies the contents of a file to the standard output, except that lines between *.[* and *.]* are

	interpreted as citations, and lines between *.R1* and *.R2* are interpreted as commands for how citations are to be processed
soelim	Reads files and replaces lines of the form *.so file* by the contents of the mentioned *file*
tbl	Compiles descriptions of tables embedded within troff input files into commands that are understood by **troff**
tfmtodit	Creates a font file for use with **groff -Tdvi**
troff	Is highly compatible with Unix **troff**; it should usually be invoked using the **groff** command, which will also run preprocessors and post-processors in the appropriate order and with the appropriate options
zsoelim	A link to **soelim**

10.35. Gzip-1.3.5

The Gzip package contains programs for compressing and decompressing files.

10.35.1. Installation of Gzip

The following patch fixes two security vulnerabilities in Gzip: a path traversal bug when using the `-N` option (CAN-2005-1228), and a race condition in the file permission restore code (CAN-2005-0998):

```
patch -Np1 -i ../gzip-1.3.5-security_fixes-1.patch
```

The **gzexe** command calls **tail** with options that do not conform to newer versions of the POSIX standard, and therefore are not accepted by current versions of Coreutils. Fix this problem by issuing the following command:

```
sed -i 's/tail +/tail -n +/' gzexe.in
```

Prepare Gzip for compilation:

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

The **gzexe** script has the location of the **gzip** binary hard-wired into it. Because the location of the binary is changed later, the following command ensures that the new location gets placed into the script:

```
sed -i 's@"BINDIR"@/bin@g' gzexe.in
```

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Move the **gzip** program to the `/bin` directory and create some commonly used symlinks to it:

```
mv -v /usr/bin/gzip /bin
rm -v /usr/bin/{gunzip,zcat}
ln -sv gzip /bin/gunzip
ln -sv gzip /bin/zcat
ln -sv gzip /bin/compress
ln -sv gunzip /bin/uncompress
```

10.35.2. Contents of Gzip

Installed programs: compress (link to gzip), 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

compress	Compresses and decompresses files
gunzip	Decompresses gzipped files
gzexe	Creates self-decompressing executable files
gzip	Compresses the given files using Lempel-Ziv (LZ77) coding
uncompress	Decompresses compressed files
zcat	Decompresses the given gzipped files to standard output
zcmp	Runs **cmp** on gzipped files
zdiff	Runs **diff** on gzipped files
zegrep	Runs **egrep** on gzipped files
zfgrep	Runs **fgrep** on gzipped files
zforce	Forces a `.gz` extension on all given files that are gzipped files, so that **gzip** will not compress them again; this can be useful when file names were truncated during a file transfer
zgrep	Runs **grep** on gzipped files
zless	Runs **less** on gzipped files
zmore	Runs **more** on gzipped files
znew	Re-compresses files from **compress** format to **gzip** format—`.Z` to `.gz`

10.36. Inetutils-1.4.2

The Inetutils package contains programs for basic networking.

10.36.1. Installation of Inetutils

Not all programs that come with Inetutils will be installed. However, the Inetutils build system will insist on installing all the man pages anyway. The following patch will correct this situation:

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

This patch addresses build issues with GCC 4.1.1:

```
patch -Np1 -i ../inetutils-1.4.2-gcc4_fixes-3.patch
```

This patch addresses an issue where telnet on certain machines will only send to 255.255.255.255. This was due to change in the way glibc handles inet_addr. Apply the following patch to correct this issue:

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

Prepare Inetutils for compilation:

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

The meaning of the configure options:

`--disable-logger`
This option prevents Inetutils from installing the **logger** program, which is used by scripts to pass messages to the System Log Daemon. Do not install it because Util-linux installs a better version later.

`--disable-syslogd`
This option prevents Inetutils from installing the System Log Daemon, which is installed with the Sysklogd package.

`--disable-whois`
This option disables the building of the Inetutils **whois** client, which is out of date. Instructions for a better **whois** client are in the BLFS book.

`--disable-servers`
This disables the installation of the various network servers included as part of the Inetutils package. These servers are deemed not appropriate in a basic CLFS system. Some are insecure by nature and are only considered safe on trusted networks. More information can be found at *http://www.linuxfromscratch.org/blfs/view/svn/basicnet/inetutils.html*. Note that better replacements are available for many of these servers.

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Move the **ping** program to its FHS-compliant place:

```
mv -v /usr/bin/ping /bin
```

10.36.2. Contents of Inetutils

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

Short Descriptions

ftp	Is the file transfer protocol program
ping	Sends echo-request packets and reports how long the replies take
rcp	Performs remote file copy
rlogin	Performs remote login
rsh	Runs a remote shell
talk	Is used to chat with another user
telnet	An interface to the TELNET protocol
tftp	A trivial file transfer program

10.37. Kbd-1.12

The Kbd package contains key-table files and keyboard utilities.

10.37.1. Installation of Kbd

The following patch fixes build issues with GCC 4.1.1:

```
patch -Np1 -i ../kbd-1.12-gcc4_fixes-1.patch
```

Prepare Kbd for compilation:

```
./configure --datadir=/lib/kbd
```

The meaning of the configure options:

--datadir
This option places the Kbd data and keymap files into /lib/kbd, as they are used by some of the scripts in the CLFS-Bootscripts package and must be on the root partition.

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Some of the programs from Kbd are used by scripts in the CLFS-Bootscripts package. As `/usr` may not be available during the early stages of booting, those binaries need to be on the root partition:

```
mv -v /usr/bin/{kbd_mode,openvt,setfont} /bin
```

10.37.2. Contents of Kbd

Installed programs: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (link to psfxtable), psfgettable (link to psfxtable), psfstriptable (link to psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setmetamode, 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
kbd_mode	Reports or sets the keyboard mode
kbdrate	Sets the keyboard repeat and delay rates
loadkeys	Loads the keyboard translation tables
loadunimap	Loads the kernel unicode-to-font mapping table
mapscrn	An obsolete program that used to load a user-defined output character mapping table into the console driver; this is now done by **setfont**
openvt	Starts a program on a new virtual terminal (VT)
psfaddtable	A link to **psfxtable**
psfgettable	A link to **psfxtable**
psfstriptable	A link to **psfxtable**
psfxtable	Handle Unicode character tables for console fonts
resizecons	Changes the kernel idea of the console size
setfont	Changes the Enhanced Graphic Adapter (EGA) and Video Graphics Array (VGA) fonts on the console
setkeycodes	Loads kernel scancode-to-keycode mapping table entries; this is useful if there are unusual keys on the keyboard
setleds	Sets the keyboard flags and Light Emitting Diodes (LEDs)
setmetamode	Defines the keyboard meta-key handling
showconsolefont	Shows the current EGA/VGA console screen font
showkey	Reports the scancodes, keycodes, and ASCII codes of the keys pressed on the keyboard
unicode_start	Puts the keyboard and console in UNICODE mode. Never use it on CLFS, because applications are not configured to support UNICODE.
unicode_stop	Reverts keyboard and console from UNICODE mode

10.38. Less-394

The Less package contains a text file viewer.

10.38.1. Installation of Less

Prepare Less for compilation:

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

The meaning of the configure option:

`--sysconfdir=/etc`
This option tells the programs created by the package to look in `/etc` for the configuration files.

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Move less to /bin:

```
mv -v /usr/bin/less /bin
```

10.38.2. Contents of Less

Installed programs: less, lessecho, and lesskey

Short Descriptions

less	A file viewer or pager; it displays the contents of the given file, letting the user scroll, find strings, and jump to marks
lessecho	Needed to expand meta-characters, such as * and ?, in filenames on Unix systems
lesskey	Used to specify the key bindings for **less**

10.39. Make-3.81

The Make package contains a program for compiling packages.

10.39.1. Installation of Make

Prepare Make for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.39.2. Contents of Make

Installed program: make

Short Descriptions

make Automatically determines which pieces of a package need to be (re)compiled and then issues the relevant commands

10.40. Man-1.6d

The Man package contains programs for finding and viewing man pages.

10.40.1. Installation of Man

A few adjustments need to be made to the sources of Man.

First, a **sed** substitution is needed to add the `-R` switch to the `PAGER` variable so that escape sequences are properly handled by Less:

```
sed -i 's@-is@&R@g' configure
```

Another **sed** substitution comments out the "MANPATH /usr/man" line in the `man.conf` file to prevent redundant results when using programs such as **whatis**:

```
sed -i 's@MANPATH./usr/man@#&@g' src/man.conf.in
```

Prepare Man for compilation:

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

The meaning of the configure options:

`-confdir=/etc`
This tells the **man** program to look for the `man.conf` configuration file in the `/etc` directory.

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Note

If you will be working on a terminal that does not support text attributes such as color and bold, you can disable Select Graphic Rendition (SGR) escape sequences by editing the `man.conf` file and adding the `-c` option to the `NROFF` variable. If you use multiple terminal types for one computer it may be better to selectively add the `GROFF_NO_SGR` environment variable for the terminals that do not support SGR.

If the character set of the locale uses 8-bit characters, search for the line beginning with "NROFF" in `/etc/man.conf`, and verify that it matches the following:

```
NROFF /usr/bin/nroff -Tlatin1 -mandoc
```

Note that “latin1” should be used even if it is not the character set of the locale. The reason is that, according to the specification, **groff** has no means of typesetting characters outside International Organization for Standards (ISO) 8859-1 without some strange escape codes. When formatting man pages, **groff** thinks that they are in the ISO 8859-1 encoding and this `-Tlatin1` switch tells **groff** to use the same encoding for output. Since **groff** does no recoding of input characters, the f ormatted result is really in the same encoding as input, and therefore it is usable as the input for a pager.

This does not solve the problem of a non-working **man2dvi** program for localized man pages in non-ISO 8859-1 locales. Also, it does not work with multibyte character sets. The first problem does not currently have a solution. The second issue is not of concern because the CLFS installation does not support multibyte character sets.

Additional information with regards to the compression of man and info pages can be found in the BLFS book at *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/compressdoc.html*.

10.40.2. Contents of Man

Installed programs: apropos, makewhatis, man, man2dvi, man2html, and whatis

Short Descriptions

apropos	Searches the **whatis** database and displays the short descriptions of system commands that contain a given string
makewhatis	Builds the **whatis** database; it reads all the man pages in the `MANPATH` and writes the name and a short description in the **whatis** database for each page
man	Formats and displays the requested on-line man page
man2dvi	Converts a man page into dvi format
man2html	Converts a man 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

10.41. Man-pages-2.33

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

10.41.1. Installation of Man-pages

Install Man-pages by running:

```
make install
```

10.41.2. Contents of Man-pages

Installed files: various man pages

Short Descriptions

`man pages` This package contains man pages that describe the following: POSIX headers (section 0p), POSIX utilities (section 1p), POSIX functions (section 3p), user commands (section 1), system calls (section 2), libc calls (section 3), device information (section 4), file formats (section 5), games (section 6), conventions and macro packages (section 7), system administration (section 8), and kernel (section 9).

10.42. Mktemp-1.5

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

10.42.1. Installation of Mktemp

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

```
patch -Np1 -i ../mktemp-1.5-add_tempfile-3.patch
```

Prepare Mktemp for compilation:

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

The meaning of the configure option:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

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

10.42.2. Contents of Mktemp

Installed programs: mktemp and tempfile

Short Descriptions

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

10.43. Module-Init-Tools-3.2.2

The Module-Init-Tools package contains programs for handling kernel modules in Linux kernels greater than or equal to version 2.5.47.

10.43.1. Installation of Module-Init-Tools

Issue the following commands to perform the tests (note that the **make distclean** command is required to clean up the source tree, as the source gets recompiled as part of the testing process):

```
./configure &&
make check &&
make distclean
```

Prepare Module-Init-Tools for compilation:

```
./configure --prefix=/ --enable-zlib
```

The meaning of the configure options:

`--enable-zlib`
This allows the Module-Init-Tools package to handle compressed kernel modules.

Compile the package:

```
make
```

Install the package:

```
make INSTALL=install install
```

The meaning of the make parameter:

`INSTALL=install`
Normally, **make install** will not install the binaries if they already exist. This option overrides that behavior by calling **install** instead of using the default wrapper script.

10.43.2. Contents of Module-Init-Tools

Installed programs: depmod, generate-modprobe.conf, insmod, insmod.static, 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

generate-modprobe.conf	Creates a modprobe.conf file from an existing 2.2 or 2.4 module setup
insmod	Installs a loadable module in the running kernel
insmod.static	A statically compiled version of **insmod**
lsmod	Lists currently loaded modules
modinfo	Examines an object file associated with a kernel module and displays any information that it can glean
modprobe	Uses a dependency file, created by **depmod**, to automatically load relevant modules
rmmod	Unloads modules from the running kernel

10.44. Patch-2.5.9

The Patch package contains a program for modifying or creating files by applying a “patch” file typically created by the **diff** program.

10.44.1. Installation of Patch

Prepare Patch for compilation:

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

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

10.44.2. Contents of Patch

Installed program: patch

Short Descriptions

patch Modifies files according to a patch file. A patch file is normally a difference listing created with the **diff** program. By applying these differences to the original files, **patch** creates the patched versions.

10.45. Psmisc-22.2

The Psmisc package contains programs for displaying information about running processes.

10.45.1. Installation of Psmisc

Prepare Psmisc for compilation:

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

The meaning of the configure option:

`--exec-prefix=""`
This ensures that the Psmisc binaries will install into `/bin` instead of `/usr/bin`. This is the correct location according to the FHS, because some of the Psmisc binaries are used by the CLFS-Bootscripts package.

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

There is no reason for the **pstree** and **pstree.x11** programs to reside in `/bin`. Therefore, move them to `/usr/bin`:

```
mv -v /bin/pstree* /usr/bin
```

By default, Psmisc's **pidof** program is not installed. This usually is not a problem because it is installed later in the Sysvinit package, which provides a better **pidof** program. If Sysvinit will not be used for a particular system, complete the installation of Psmisc by creating the following symlink:

```
ln -sv killall /bin/pidof
```

10.45.2. Contents of Psmisc

Installed programs: fuser, killall, pstree, and pstree.x11 (link to pstree)

Short Descriptions

fuser	Reports the Process IDs (PIDs) of processes that use the given files or file systems
killall	Kills processes by name; it sends a signal to all processes running any of the given commands
oldfuser	Reports the Process IDs (PIDs) of processes that use the given files or file systems

pstree Displays running processes as a tree

pstree.x11 Same as **pstree**, except that it waits for confirmation before exiting

10.46. Shadow-4.0.16

The Shadow package contains programs for handling passwords in a secure way.

10.46.1. Installation of Shadow

Note

If you would like to enforce the use of strong passwords, refer to *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/cracklib.html* for installing Cracklib prior to building Shadow. Then add *--with-libcrack* to the **configure** command below.

Prepare Shadow for compilation:

```
./configure --libdir=/lib --sysconfdir=/etc --enable-shared \
    --without-libpam --without-audit --without-selinux
```

The meaning of the configure options:

--sysconfdir=/etc
Tells Shadow to install its configuration files into /etc, rather than /usr/etc.

--without-libpam
Support for Linux-PAM is enabled by default in Shadow, however PAM is not installed on a base CLFS system, so this switch disables PAM support in Shadow. For instructions to install PAM and link Shadow to it, you can look at *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/shadow.html*.

--without-audit
Support for auditing is enabled by default, but a library that it needs is not installed in a base CLFS system. This switch disables auditing support.

--without-selinux
Support for selinux is enabled by default, but selinux is not built in a base CLFS system and configure will fail without this switch.

Disable the installation of the **groups** program and its man pages, as Coreutils provides a better version:

```
sed -i 's/groups$(EXEEXT) //' src/Makefile
sed -i '/groups/d' man/Makefile
```

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

Instead of using the default *crypt* method, use the more secure *MD5* method of password encryption, which also allows passwords longer than 8 characters. It is also necessary to change the obsolete `/var/spool/mail` location for user mailboxes that Shadow uses by default to the `/var/mail` location used currently. Use the following sed command to make these changes to the appropriate configuration file:

```
sed -i /etc/login.defs \
    -e 's@#MD5_CRYPT_ENAB.no@MD5_CRYPT_ENAB yes@' \
    -e 's@/var/spool/mail@/var/mail@'
```

Note

If you built Shadow with Cracklib support, execute this **sed** to correct the path to the Cracklib dictionary:

```
sed -i 's@DICTPATH.*@DICTPATH\t/lib/cracklib/pw_dict@' /etc/login.defs
```

Move a misplaced program to its proper location:

```
mv -v /usr/bin/passwd /bin
```

Move Shadow's dynamic libraries to a more appropriate location:

```
mv -v /lib/libshadow.*a /usr/lib
rm -v /lib/libshadow.so
ln -svf ../../lib/libshadow.so.0 /usr/lib/libshadow.so
```

10.46.2. Configuring Shadow

This package contains utilities to add, modify, and delete users and groups; set and change their passwords; and perform other administrative tasks. For a full explanation of what *password shadowing* means, see the `doc/HOWTO` file within the unpacked source tree. If using Shadow support, keep in mind that programs which need to verify passwords (display managers, FTP programs, pop3 daemons, etc.) must be Shadow-compliant. That is, they need to be able to work with shadowed passwords.

To enable shadowed passwords, run the following command:

```
pwconv
```

To enable shadowed group passwords, run:

```
grpconv
```

To view or change the default settings for new user accounts that you create, you can edit `/etc/default/useradd`. See **man useradd** or *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/skel.html* for more information.

10.46.3. Setting the root password

Choose a password for user `root` and set it by running:

```
passwd root
```

10.46.4. Contents of Shadow

Installed programs: chage, chfn, chpasswd, chgpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, grpck, grpconv, grpunconv, lastlog, login, logoutd, newgrp, newusers, nologin, passwd, pwck, pwconv, pwunconv, sg (link to newgrp), useradd, userdel, usermod, vigr (link to vipw), and vipw
Installed libraries: libshadow.[a,so]

Short Descriptions

chage	Used to change the maximum number of days between obligatory password changes
chfn	Used to change a user's full name and other information
chgpasswd	Used to update group passwords in batch mode
chpasswd	Used to update the passwords of an entire series of user accounts
chsh	Used to change a user's default login shell
expiry	Checks and enforces the current password expiration policy
faillog	Is used to examine the log of login failures, to set a maximum number of failures before an account is blocked, or to reset the failure count
gpasswd	Is used to add and delete members and administrators to groups
groupadd	Creates a group with the given name
groupdel	Deletes the group with the given name
groupmod	Is used to modify the given group's name or GID
grpck	Verifies the integrity of the group files `/etc/group` and `/etc/gshadow`
grpconv	Creates or updates the shadow group file from the normal group file
grpunconv	Updates `/etc/group` from `/etc/gshadow` and then deletes the latter
lastlog	Reports the most recent login of all users or of a given user
login	Is used by the system to let users sign on
logoutd	Is a daemon used to enforce restrictions on log-on time and ports
newgrp	Is used to change the current GID during a login session
newusers	Is used to create or update an entire series of user accounts
nologin	Displays a message that an account is not available. Designed to be used as the default shell for accounts that have been disabled
passwd	Is used to change the password for a user or group account
pwck	Verifies the integrity of the password files `/etc/passwd` and `/etc/shadow`
pwconv	Creates or updates the shadow password file from the normal password file
pwunconv	Updates `/etc/passwd` from `/etc/shadow` and then deletes the latter

sg	Executes a given command while the user's GID is set to that of the given group
su	Runs a shell with substitute user and group IDs
useradd	Creates a new user with the given name, or updates the default new-user information
userdel	Deletes the given user account
usermod	Is used to modify the given user's login name, User Identification (UID), shell, initial group, home directory, etc.
vigr	Edits the `/etc/group` or `/etc/gshadow` files
vipw	Edits the `/etc/passwd` or `/etc/shadow` files
`libshadow`	Contains functions used by most programs in this package

10.47. Sysklogd-1.4.1

The Sysklogd package contains programs for logging system messages, such as those given by the kernel when unusual things happen.

10.47.1. Installation of Sysklogd

The following patch fixes various issues, including a problem building Sysklogd with Linux 2.6 series kernels:

```
patch -Np1 -i ../sysklogd-1.4.1-fixes-1.patch
```

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

10.47.2. Configuring Sysklogd

Create a new `/etc/syslog.conf` file 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
*.=info;*.=notice;*.=warn;
auth,authpriv.none;
cron,daemon.none;
mail,news.none                  -/var/log/messages.log

*.emerg                         *

# log the bootscript output:
local2.*                         -/var/log/boot.log

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

10.47.3. Contents of Sysklogd

Installed programs: klogd and syslogd

Short Descriptions

klogd A system daemon for intercepting and logging kernel messages

syslogd Logs the messages that system programs offer for logging. 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.

10.48. Sysvinit-2.86

The Sysvinit package contains programs for controlling the startup, running, and shutdown of the system.

10.48.1. Installation of Sysvinit

When run-levels are changed (for example, when halting the system), **init** sends termination signals to those processes that **init** itself started and that should not be running in the new run-level. While doing this, **init** outputs messages like “Sending processes the TERM signal” which seem to imply that it is sending these signals to all currently running processes. To avoid this misinterpretation, modify the source so that these messages read like “Sending processes started by init the TERM signal” instead:

```
sed -i 's@Sending processes@& started by init@g' \
    src/init.c
```

Compile the package:

```
make -C src clobber
make -C src
```

Install the package:

```
make -C src install
```

10.48.2. Configuring Sysvinit

Create a new file `/etc/inittab` by running the following:

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty -I '\033(K' tty1 9600
2:2345:respawn:/sbin/agetty -I '\033(K' tty2 9600
3:2345:respawn:/sbin/agetty -I '\033(K' tty3 9600
4:2345:respawn:/sbin/agetty -I '\033(K' tty4 9600
```

```
5:2345:respawn:/sbin/agetty -I '\033(K' tty5 9600
6:2345:respawn:/sbin/agetty -I '\033(K' tty6 9600

# End /etc/inittab
EOF
```

The *-I '\033(K'* option tells **agetty** to send this escape sequence to the terminal before doing anything else. This escape sequence switches the console character set to a user-defined one, which can be modified by running the **setfont** program. The **console** initscript from the CLFS-Bootscripts package calls the **setfont** program during system startup. Sending this escape sequence is necessary for people who use non-ISO 8859-1 screen fonts, but it does not affect native English speakers.

10.48.3. Contents of Sysvinit

Installed programs: bootlogd, halt, init, killall5, last, lastb (link to last), mesg, mountpoint, pidof (link to killall5), poweroff (link to halt), reboot (link to halt), runlevel, shutdown, sulogin, telinit (link to init), utmpdump, and wall

Short Descriptions

bootlogd	Logs boot messages to a log file
halt	Normally invokes **shutdown** with the *-h* option, except when already in run-level 0, then it tells the kernel to halt the system; it notes in the file `/var/log/wtmp` that the system is being brought down
init	The first process to be started when the kernel has initialized the hardware which takes over the boot process and starts all the proceses it is instructed to
killall5	Sends a signal to all processes, except the processes in its own session so it will not kill the shell running the script that called it
last	Shows which users last logged in (and out), searching back through the `/var/log/wtmp` file; it also shows system boots, shutdowns, and run-level changes
lastb	Shows the failed login attempts, as logged in `/var/log/btmp`
mesg	Controls whether other users can send messages to the current user's terminal
mountpoint	Tells you whether or not a directory is a mount point.
pidof	Reports the PIDs of the given programs
poweroff	Tells the kernel to halt the system and switch off the computer (see **halt**)
reboot	Tells the kernel to reboot the system (see **halt**)
runlevel	Reports the previous and the current run-level, as noted in the last run-level record in `/var/run/utmp`
shutdown	Brings the system down in a secure way, signaling all processes and notifying all logged-in users
sulogin	Allows *root* to log in; it is normally invoked by **init** when the system goes into single user

	mode
telinit	Tells **init** which run-level to change to
utmpdump	Displays the content of the given login file in a more user-friendly format
wall	Writes a message to all logged-in users

10.49. Tar-1.15.1

The Tar package contains an archiving program.

10.49.1. Installation of Tar

Apply a patch to fix some issues with the test suite when using GCC-4.1.1:

```
patch -Np1 -i ../tar-1.15.1-gcc4_fix_tests-1.patch
```

Tar has a bug when the `-S` option is used with files larger than 4 GB. The following patch properly fixes this issue:

```
patch -Np1 -i ../tar-1.15.1-sparse_fix-1.patch
```

This patch fixes a security vulnerability in Tar:

```
patch -Np1 -i ../tar-1.15.1-security_fixes-1.patch
```

Prepare Tar for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

10.49.2. Contents of Tar

Installed programs: rmt and tar

Short Descriptions

rmt Remotely manipulates a magnetic tape drive through an interprocess communication connection

tar Creates, extracts files from, and lists the contents of archives, also known as tarballs

10.50. Texinfo-4.8

The Texinfo package contains programs for reading, writing, and converting info pages.

10.50.1. Installation of Texinfo

Texinfo allows local users to overwrite arbitrary files via a symlink attack on temporary files. Apply the following patch to fix this:

```
patch -Np1 -i ../texinfo-4.8-tempfile_fix-2.patch
```

Prepare Texinfo for compilation:

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

Compile the package:

```
make
```

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

Install the package:

```
make install
```

The Info documentation system uses a plain text file to hold its list of menu entries. The file is located at `/usr/share/info/dir`. Unfortunately, due to occasional problems in the Makefiles of various packages, it can sometimes get out of sync with the info pages installed on the system. If the `/usr/share/info/dir` file ever needs to be recreated, the following optional commands will accomplish the task:

```
cd /usr/share/info
rm dir
for f in *
do install-info $f dir 2>/dev/null
done
```

10.50.2. Contents of Texinfo

Installed programs: info, infokey, install-info, makeinfo, texi2dvi, texi2pdf, and texindex

Short Descriptions

info	Used to read info pages which are similar to man pages, but often go much deeper than just explaining all the command line options. For example, compare **man bison** and **info bison**.
infokey	Compiles a source file containing Info customizations into a binary format
install-info	Used to install info pages; it updates entries in the **info** index file
makeinfo	Translates the given Texinfo source documents into info pages, plain text, or HTML

texi2dvi	Used to format the given Texinfo document into a device-independent file that can be printed
texi2pdf	Used to format the given Texinfo document into a Portable Document Format (PDF) file
texindex	Used to sort Texinfo index files

10.51. Udev-096

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

10.51.1. Installation of Udev

Compile the package:

```
make EXTRAS="extras/floppy extras/cdrom_id extras/firmware \
    extras/scsi_id extras/volume_id extras/ata_id extras/usb_id \
    extras/edd_id extras/dasd_id extras/path_id"
```

The meaning of the make parameter:

EXTRAS="extras/floppy extras/cdrom_id extras/firmware extras/scsi_id extras/volume_id extras/ata_id extras/usb_id extras/edd_id extras/dasd_id extras/path_id"
This builds the helper applications that are used with udev. The helper programs assist in correct handling of devices.

If you want to run the testsuite, you need to change a hardcoded reference to the **test** program:

```
sed -i 's@/usr/bin/test@/bin/test@' test/udev-test.pl
```

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

Install the package:

```
make DESTDIR=/ \
    EXTRAS="extras/floppy extras/cdrom_id extras/firmware \
    extras/scsi_id extras/volume_id extras/ata_id extras/usb_id \
    extras/edd_id extras/dasd_id extras/path_id" install
```

The meaning of the make parameter:

DESTDIR=/
This prevents the Udev build process from killing any **udevd** processes that may be running on the system.

Install a necessary helper script:

```
install -v extras/eventrecorder.sh /lib/udev
```

Install the documentation that explains how to create Udev rules:

```
install -v -m644 -D docs/writing_udev_rules/index.html \
    /usr/share/doc/udev-096/index.html
```

Create a directory for storing firmware that can be loaded by **udev**:

```
install -dv /lib/firmware
```

10.51.2. Contents of Udev

Installed programs: udevcontrol, udevd, udevinfo, udevmonitor, udevsend, udevtest, and udevtrigger
Installed directory: /etc/udev

Short Descriptions

udevcontrol	Configures a number of options for the running **udevd** daemon, such as the log level.
udevd	A daemon that reorders hotplug events before submitting them to **udev**, thus avoiding various race conditions
udevinfo	Allows users to query the **udev** database for information on any device currently present on the system; it also provides a way to query any device in the `sysfs` tree to help create udev rules
udevmonitor	Prints the event received from the kernel and the event which **udev** sends out after rule processing
udevsettle	Watches the Udev event queue and exits if all current uevents have been handled
udevtest	Simulates a **udev** run for the given device, and prints out the name of the node the real **udev** would have created or the name of the renamed network interface
udevtrigger	Walks the sysfs tree for devices that need to be added to the system.
ata_id	Provides Udev with a unique string and additional information (uuid, label) for an ATA drive
cdrom_id	Print the capabilities of a CDROM or DVDROM drive.
create_floppy_devices	Creates all possible floppy devices based on the CMOS type
dasd_id	Read the label from an s390 block device.
edd_id	Identify x86 disk drives from Enhanced Disk Drive calls.
firmware.sh	Script to load firmware for a device
path_id	Provide the shortest possible unique hardware path to a device
scsi_id	Retrieve or generate a unique SCSI identifier.
usb_id	Identify a USB block device.
vol_id	Probe filesystem type and read label and uuid.
`/etc/udev`	Contains **udev** configuration files, device permissions, and rules for device naming
`/lib/udev`	Contains **udev** helper programs and static devices which get copied to /dev when booted.

10.52. Util-linux-2.12r

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

10.52.1. FHS compliance notes

The FHS recommends using the `/var/lib/hwclock` directory instead of the usual `/etc` directory as the location for the `adjtime` file. To make the **hwclock** program FHS-compliant, run the following:

```
sed -i 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
    hwclock/hwclock.c
mkdir -pv /var/lib/hwclock
```

10.52.2. Installation of Util-linux

Util-linux fails to compile against newer versions of Linux kernel headers. The following patch properly fixes this issue:

```
patch -Np1 -i ../util-linux-2.12r-cramfs-1.patch
```

The following patch fixes build issues with GCC 4.1.1:

```
patch -Np1 -i ../util-linux-2.12r-gcc4_fixes-1.patch
```

The following patch fixes swapon.c - it tries to find the variable R_OK, but the header that has R_OK is not included:

```
patch -Np1 -i ../util-linux-2.12r-missing_header-1.patch
```

Prepare Util-linux for compilation:

```
./configure
```

Compile the package:

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

The meaning of the make parameters:

HAVE_KILL=yes
: This prevents the **kill** program (already installed by Procps) from being built and installed again.

HAVE_SLN=yes
: This prevents the **sln** program (a statically linked version of **ln** already installed by Glibc) from being built and installed again.

This package does not come with a test suite.

Install the package and move the **logger** binary to `/bin` as it is needed by the CLFS-Bootscripts package:

```
make HAVE_KILL=yes HAVE_SLN=yes install
mv -v /usr/bin/logger /bin
```

10.52.3. Contents of Util-linux

Installed programs: agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, flock, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, ipcrm, ipcs, isosize, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, pg, pivot_root, ramsize (link to rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (link to rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (link to swapon), swapon, tailf, tunelp, ul, umount, vidmode (link to rdev), whereis, and write

Short Descriptions

agetty	Opens a tty port, prompts for a login name, and then invokes the **login** program
arch	Reports the machine's architecture
blockdev	Allows users to call block device ioctls from the command line
cal	Displays a simple calendar
cfdisk	Manipulates the partition table of the given device
chkdupexe	Finds duplicate executables
col	Filters out reverse line feeds
colcrt	Filters **nroff** output for terminals that lack some capabilities, such as overstriking and half-lines
colrm	Filters out the given columns
column	Formats a given file into multiple columns
ctrlaltdel	Sets the function of the Ctrl+Alt+Del key combination to a hard or a soft reset
cytune	Tunes the parameters of the serial line drivers for Cyclades cards
ddate	Gives the Discordian date or converts the given Gregorian date to a Discordian one
dmesg	Dumps the kernel boot messages
elvtune	Tunes the performance and interactivity of a block device
fdformat	Low-level formats a floppy disk
fdisk	Manipulates the partition table of the given device
flock	Acquires a file lock and then executes a command with the lock held
fsck.cramfs	Performs a consistency check on the Cramfs file system on the given device
fsck.minix	Performs a consistency check on the Minix file system on the given device
getopt	Parses options in the given command line

hexdump	Dumps the given file in hexadecimal or in another given format
hwclock	Reads or sets the system's hardware clock, also called the Real-Time Clock (RTC) or Basic Input-Output System (BIOS) clock
ipcrm	Removes the given Inter-Process Communication (IPC) resource
ipcs	Provides IPC status information
isosize	Reports the size of an iso9660 file system
line	Copies a single line
logger	Enters the given message into the system log
look	Displays lines that begin with the given string
losetup	Sets up and controls loop devices
mcookie	Generates magic cookies (128-bit random hexadecimal numbers) for **xauth**
mkfs	Builds a file system on a device (usually a hard disk partition)
mkfs.bfs	Creates a Santa Cruz Operations (SCO) bfs file system
mkfs.cramfs	Creates a cramfs file system
mkfs.minix	Creates a Minix file system
mkswap	Initializes the given device or file to be used as a swap area
more	A filter for paging through text one screen at a time
mount	Attaches the file system on the given device to a specified directory in the file-system tree
namei	Shows the symbolic links in the given pathnames
pg	Displays a text file one screen full at a time
pivot_root	Makes the given file system the new root file system of the current process
ramsize	Sets the size of the RAM disk in a bootable image
raw	Used to bind a Linux raw character device to a block device
rdev	Queries and sets the root device, among other things, in a bootable image
readprofile	Reads kernel profiling information
rename	Renames the given files, replacing a given string with another
renice	Alters the priority of running processes
rev	Reverses the lines of a given file
rootflags	Sets the rootflags in a bootable image
script	Makes a typescript of a terminal session
setfdprm	Sets user-provided floppy disk parameters
setsid	Runs the given program in a new session

setterm	Sets terminal attributes
sfdisk	A disk partition table manipulator
swapoff	Disables devices and files for paging and swapping
swapon	Enables devices and files for paging and swapping and lists the devices and files currently in use
tailf	Tracks the growth of a log file. Displays the last 10 lines of a log file, then continues displaying any new entries in the log file as they are created
tunelp	Tunes the parameters of the line printer
ul	A filter for translating underscores into escape sequences indicating underlining for the terminal in use
umount	Disconnects a file system from the system's file tree
vidmode	Sets the video mode in a bootable image
whereis	Reports the location of the binary, source, and man page for the given command
write	Sends a message to the given user *if* that user has not disabled receipt of such messages

10.53. Vim-7.0

The Vim package contains a powerful text editor.

Alternatives to Vim

If you prefer another editor—such as Emacs, Joe, or Nano—please refer to *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html* for suggested installation instructions.

10.53.1. Installation of Vim

First, unpack both `vim-7.0.tar.bz2` and (optionally) `vim-7.0-lang.tar.gz` archives into the same directory.

The following patch contains updates from the maintainer. The maintainer of Vim only releases these patches to fix serious issues.

```
patch -Np1 -i ../vim-7.0-fixes-5.patch
```

Change the default location of the `vimrc` configuration file to `/etc`:

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

Prepare Vim for compilation:

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

The meaning of the configure options:

`--enable-multibyte`
This optional but highly recommended switch enables support for editing files in multibyte character encodings. This is needed if using a locale with a multibyte character set. This switch is also helpful to be able to edit text files initially created in Linux distributions like Fedora Core that use UTF-8 as a default character set.

Compile the package:

```
make
```

To test the results, issue: **`make test`**. However, this test suite outputs a lot of binary data to the screen, which can cause issues with the settings of the current terminal. This can be resolved by redirecting the output to a log file.

Install the package:

```
make install
```

Many users are accustomed to using **vi** instead of **vim**. Some programs, such as **vigr** and **vipw**, also use **vi**.

Create a symlink to permit execution of **vim** when users habitually enter **vi** and allow programs that use **vi** to work:

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

By default, Vim's documentation is installed in `/usr/share/vim`. The following symlink allows the documentation to be accessed via `/usr/share/doc/vim-7.0`, making it consistent with the location of documentation for other packages:

```
ln -sv ../vim/vim70/doc /usr/share/doc/vim-7.0
```

If an X Window System is going to be installed on the CLFS system, you may want to recompile Vim after installing X. Vim comes with a GUI version of the editor that requires X and some additional libraries to be installed. For more information, refer to the Vim documentation and the Vim installation page in the BLFS book at *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html#postlfs-editors-vim*.

10.53.2. Configuring Vim

By default, **vim** runs in vi-incompatible mode. This may be new to users who have used other editors in the past. The “nocompatible” setting is included below to highlight the fact that a new behavior is being used. It also reminds those who would change to “compatible” mode that it should be the first setting in the configuration file. This is necessary because it changes other settings, and overrides must come after this setting. Create a default **vim** configuration file by running the following:

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

set nocompatible
set backspace=2
syntax on
if (&term == "iterm") || (&term == "putty")
  set background=dark
endif

" End /etc/vimrc
EOF
```

The *`set nocompatible`* makes **vim** behave in a more useful way (the default) than the vi-compatible manner. Remove the “no” to keep the old **vi** behavior. The *`set backspace=2`* allows backspacing over line breaks, autoindents, and the start of insert. The *`syntax on`* enables vim's syntax highlighting. Finally, the *if* statement with the *`set background=dark`* corrects **vim**'s guess about the background color of some terminal emulators. This gives the highlighting a better color scheme for use on the black background of these programs.

Documentation for other available options can be obtained by running the following command:

```
vim -c ':options'
```

10.53.3. Contents of Vim

Installed programs: efm_filter.pl, efm_perl.pl, ex (link to vim), less.sh, mve.awk, pltags.pl, ref, rview (link to vim), rvim (link to vim), shtags.pl, tcltags, vi (link to vim), view (link to vim), vim, vim132, vim2html.pl,

vimdiff (link to vim), vimm, vimspell.sh, vimtutor, and xxd

Short Descriptions

efm_filter.pl	A filter for creating an error file that can be read by **vim**
efm_perl.pl	Reformats the error messages of the Perl interpreter for use with the “quickfix” mode of **vim**
ex	Starts **vim** in ex mode
less.sh	A script that starts **vim** with less.vim
mve.awk	Processes **vim** errors
pltags.pl	Creates a tags file for Perl code for use by **vim**
ref	Checks the spelling of arguments
rview	Is a restricted version of **view**; no shell commands can be started and **view** cannot be suspended
rvim	Is a restricted version of **vim**; no shell commands can be started and **vim** cannot be suspended
shtags.pl	Generates a tags file for Perl scripts
tcltags	Generates a tags file for TCL code
view	Starts **vim** in read-only mode
vi	Link to **vim**
vim	Is the editor
vim132	Starts **vim** with the terminal in 132-column mode
vim2html.pl	Converts Vim documentation to HypterText Markup Language (HTML)
vimdiff	Edits two or three versions of a file with **vim** and show differences
vimm	Enables the DEC locator input model on a remote terminal
vimspell.sh	Spell checks a file and generates the syntax statements necessary to highlight in **vim**. This script requires the old Unix **spell** command, which is provided neither in CLFS nor in BLFS
vimtutor	Teaches the basic keys and commands of **vim**
xxd	Creates a hex dump of the given file; it can also do the reverse, so it can be used for binary patching

10.54. Bin86-0.16.17

The Bin86 package contains an assembler and linker to produce 16 or 32-bit 8086 machine code.

10.54.1. Installation of Bin86

We are building Bin86 because it is required to compile Lilo, and at the moment no other bootloader builds and runs on a pure64 system. If your machine has multiple systems (i.e. x86 or multilib) you may prefer to use the bootloader from those systems, such as GRUB.

This patch updates Bin86 to compile on x86_64:

```
patch -Np1 -i ../bin86-0.16.17-x86_64-1.patch
```

Compile the package:

```
make
```

Install the package:

```
make PREFIX=/usr install
```

10.54.2. Contents of Bin86

Installed programs: as86, ld86, nm86, objdump86, size86

Short Descriptions

as86	An 8086 assembler
ld86	A linker for the object files produced by as86
nm86	Symbolic link to objdump86
objdump86	A utility to examine and report on the output from as86 and ld86
size86	Symbolic link to objdump86

10.55. Lilo-22.7.1

The Lilo package contains the Linux Loader, a bootloader.

We have chosen to use Lilo because at the moment no other bootloader builds and runs on a pure64 system. If your machine has multiple systems (i.e. x86 or multilib) you may prefer to use the bootloader from the other system, such as GRUB.

10.55.1. Installation of Lilo

Compile the package:

```
make
```

Install the package:

```
make install
```

At the end of the installation the make install process will print a message stating that /sbin/lilo has to be executed to complete the update. Don't do this as it has no use. The /etc/lilo.conf isn't present yet. We will complete the installation of Lilo in chapter 12.

10.55.2. Contents of Lilo

Installed programs: diag1.img, lilo, mkrescue, keytab-lilo.pl

Short Descriptions

diag1.img	A diagnostic disk boot image.
lilo	Lilo installs the Linux boot loader which is used to start a Linux system.
mkrescue	A script to make a bootable floppy or CD image using the default settings from the configuration file.
keytab-lilo.pl	A perl script to create a keyboard translation table to allow the bootloader to process keystrokes to match your keyboard layout.

10.56. About Debugging Symbols

Most programs and libraries are, by default, compiled with debugging symbols included (with **gcc**'s `-g` option). This means that when debugging a program or library that was compiled with debugging information included, the debugger can provide not only memory addresses, but also the names of the routines and variables.

However, the inclusion of these debugging symbols enlarges a program or library significantly. The following is an example of the amount of space these symbols occupy:

- a bash binary with debugging symbols: 1200 KB
- a bash binary without debugging symbols: 480 KB
- Glibc and GCC files (`/lib` and `/usr/lib`) with debugging symbols: 87 MB
- Glibc and GCC files without debugging symbols: 16 MB

Sizes may vary depending on which compiler and C library were used, but when comparing programs with and without debugging symbols, the difference will usually be a factor between two and five.

Because most users will never use a debugger on their system software, a lot of disk space can be regained by removing these symbols. The next section shows how to strip all debugging symbols from the programs and libraries. Additional information on system optimization can be found at *http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt*.

10.57. Stripping

If the intended user is not a programmer and does not plan to do any debugging on the system software, the system size can be decreased by about 200 MB by removing the debugging symbols from binaries and libraries. This causes no inconvenience other than not being able to debug the software fully anymore.

Most people who use the command mentioned below do not experience any difficulties. However, it is easy to make a typo and render the new system unusable, so before running the **strip** command, it is a good idea to make a backup of the current situation.

Before performing the stripping, take special care to ensure that none of the binaries that are about to be stripped are running. If unsure whether the user entered chroot with the command given in If You Are Going to Chroot first exit from chroot:

```
logout
```

Then reenter it with:

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

Now the binaries and libraries can be safely stripped:

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

A large number of files will be reported as having their file format not recognized. These warnings can be safely ignored. These warnings indicate that those files are scripts instead of binaries.

If disk space is very tight, the `--strip-all` option can be used on the binaries in `/{,usr/}{bin,sbin}` to gain several more megabytes. Do not use this option on libraries—they will be destroyed.

Chapter 11. Setting Up System Bootscripts

11.1. Introduction

This chapter details how to install and configure the CLFS-Bootscripts package. Most of these scripts will work without modification, but a few require additional configuration files because they deal with hardware-dependent information.

System-V style init scripts are employed in this book because they are widely used. For additional options, a hint detailing the BSD style init setup is available at *http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt*. Searching the LFS mailing lists for “depinit” will also offer additional choices.

If using an alternative style of init scripts, skip this chapter and move on to Making the CLFS System Bootable.

11.2. CLFS-Bootscripts-1.0

The CLFS-Bootscripts package contains a set of scripts to start/stop the CLFS system at bootup/shutdown.

11.2.1. Installation of CLFS-Bootscripts

Install the package:

```
make install
```

11.2.2. Contents of CLFS-Bootscripts

Installed scripts: checkfs, cleanfs, console, functions, halt, ifdown, ifup, localnet, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysklogd, and template.

Short Descriptions

checkfs	Checks the integrity of the file systems before they are mounted (with the exception of journal and network based file systems)
cleanfs	Removes files that should not be preserved between reboots, such as those in `/var/run/` and `/var/lock/`; it re-creates `/var/run/utmp` and removes the possibly present `/etc/nologin`, `/fastboot`, and `/forcefsck` files
console	Loads the correct keymap table for the desired keyboard layout; it also sets the screen font
functions	Contains common functions, such as error and status checking, that are used by several bootscripts
halt	Halts the system
ifdown	Assists the network script with stopping network devices
ifup	Assists the network script with starting network devices
localnet	Sets up the system's hostname and local loopback device
mountfs	Mounts all file systems, except ones that are marked *noauto* or are network based
mountkernfs	Mounts virtual kernel file systems, such as `proc`
network	Sets up network interfaces, such as network cards, and sets up the default gateway (where applicable)
rc	The master run-level control script; it is responsible for running all the other bootscripts one-by-one, in a sequence determined by the name of the symbolic links being processed
reboot	Reboots the system
sendsignals	Makes sure every process is terminated before the system reboots or halts
setclock	Resets the kernel clock to local time in case the hardware clock is not set to UTC time
static	Provides the functionality needed to assign a static Internet Protocol (IP) address to a

	network interface
swap	Enables and disables swap files and partitions
sysklogd	Starts and stops the system and kernel log daemons
template	A template to create custom bootscripts for other daemons

11.3. Udev Rules-1.0-3

The Udev Cross-LFS rules package contains the necessary rules set for a basic functional system.

11.3.1. Installation of Udev-Rules

Install the package:

```
make install
```

11.3.2. Contents of Udev Rules

Installed Rules: 05-udev-early.rules, 35-helper.rules, 40-modprobe.rules, 50-udev.rules, 55-sound.rules, 60-persistent-disk.rules, 61-persistent-input.rules, 90-user.rules, 95-debug.rules, load_floppy_module.sh, show_event_log, udev, and udev_retry

Short Descriptions

05-udev-early.rules	Rules that are run early in the boot process. These rules check for the ability to do hotplug events connect to the network, and make sure that sysfs is ready before running any other rules.
35-helper.rules	Rules that use helper programs.
40-modprobe.rules	Rules that use modprobe to load kernel modules.
50-udev.rules	Creates basic system devices and permissions.
55-sound.rules	Gives proper permissions to audio device and restores sound volumes.
60-persistent-disk.rules	Allows persistent naming of disk drives.
61-persistent-input.rules	Allows persistent naming of input devices.
90-user.rules	Rules defined by the user.
95-debug.rules	Rules for debugging udev.
load_floppy_module.sh	Checks to see if a floppy drive exists before loading the floppy drive module into memory.
show_event_log	Displays Udev log messages from /var/log/messages.
udev	To be Written.

11.4. How Do These Bootscripts Work?

Linux uses a special booting facility named SysVinit that is based on a concept of *run-levels*. It can be quite different from one system to another, so it cannot be assumed that because things worked in one particular Linux distribution, they should work the same in CLFS too. CLFS has its own way of doing things, but it respects generally accepted standards.

SysVinit (which will be referred to as “init” from now on) works using a run-levels scheme. There are seven (numbered 0 to 6) run-levels (actually, there are more run-levels, but they are for special cases and are generally not used. See `init(8)` for more details), and each one of those corresponds to the actions the computer is supposed to perform when it starts up. The default run-level is 3. Here are the descriptions of the different run-levels as they are implemented:

```
0: halt the computer
1: single-user mode
2: multi-user mode without networking
3: multi-user mode with networking
4: reserved for customization, otherwise does the same as 3
5: same as 4, it is usually used for GUI login (like X's xdm or KDE's kdm)
6: reboot the computer
```

The command used to change run-levels is **init *[runlevel]***, where *[runlevel]* is the target run-level. For example, to reboot the computer, a user could issue the **init 6** command, which is an alias for the **reboot** command. Likewise, **init 0** is an alias for the **halt** command.

There are a number of directories under `/etc/rc.d` that look like `rc?.d` (where ? is the number of the run-level) and `rcsysinit.d`, all containing a number of symbolic links. Some begin with a *K*, the others begin with an *S*, and all of them have two numbers following the initial letter. The K means to stop (kill) a service and the S means to start a service. The numbers determine the order in which the scripts are run, from 00 to 99—the lower the number the earlier it gets executed. When **init** switches to another run-level, the appropriate services are either started or stopped, depending on the runlevel chosen.

The real scripts are in `/etc/rc.d/init.d`. They do the actual work, and the symlinks all point to them. Killing links and starting links point to the same script in `/etc/rc.d/init.d`. This is because the scripts can be called with different parameters like `start`, `stop`, `restart`, `reload`, and `status`. When a K link is encountered, the appropriate script is run with the `stop` argument. When an S link is encountered, the appropriate script is run with the `start` argument.

There is one exception to this explanation. Links that start with an *S* in the `rc0.d` and `rc6.d` directories will not cause anything to be started. They will be called with the parameter `stop` to stop something. The logic behind this is that when a user is going to reboot or halt the system, nothing needs to be started. The system only needs to be stopped.

These are descriptions of what the arguments make the scripts do:

`start`
: The service is started.

`stop`
: The service is stopped.

`restart`
: The service is stopped and then started again.

`reload`
: The configuration of the service is updated. This is used after the configuration file of a service was modified, when the service does not need to be restarted.

`status`
: Tells if the service is running and with which PIDs.

Feel free to modify the way the boot process works (after all, it is your own CLFS system). The files given here are an example of how it can be done.

11.5. Device and Module Handling on a CLFS System

In Installing Basic System Software, we installed the Udev package. Before we go into the details regarding how this works, a brief history of previous methods of handling devices is in order.

Linux systems in general traditionally use a static device creation method, whereby a great many device nodes are created under `/dev` (sometimes literally thousands of nodes), regardless of whether the corresponding hardware devices actually exist. This is typically done via a **MAKEDEV** script, which contains a number of calls to the **mknod** program with the relevant major and minor device numbers for every possible device that might exist in the world.

Using the Udev method, only those devices which are detected by the kernel get device nodes created for them. Because these device nodes will be created each time the system boots, they will be stored on a `tmpfs` file system (a virtual file system that resides entirely in system memory). Device nodes do not require much space, so the memory that is used is negligible.

11.5.1. History

In February 2000, a new filesystem called `devfs` was merged into the 2.3.46 kernel and was made available during the 2.4 series of stable kernels. Although it was present in the kernel source itself, this method of creating devices dynamically never received overwhelming support from the core kernel developers.

The main problem with the approach adopted by `devfs` was the way it handled device detection, creation, and naming. The latter issue, that of device node naming, was perhaps the most critical. It is generally accepted that if device names are allowed to be configurable, then the device naming policy should be up to a system administrator, not imposed on them by any particular developer(s). The `devfs` file system also suffers from race conditions that are inherent in its design and cannot be fixed without a substantial revision to the kernel. It has also been marked as deprecated due to a lack of recent maintenance.

With the development of the unstable 2.5 kernel tree, later released as the 2.6 series of stable kernels, a new virtual filesystem called `sysfs` came to be. The job of `sysfs` is to export a view of the system's hardware configuration to userspace processes. With this userspace-visible representation, the possibility of seeing a userspace replacement for `devfs` became much more realistic.

11.5.2. Udev Implementation

11.5.2.1. Sysfs

The `sysfs` filesystem was mentioned briefly above. One may wonder how `sysfs` knows about the devices present on a system and what device numbers should be used for them. Drivers that have been compiled into the kernel directly register their objects with `sysfs` as they are detected by the kernel. For drivers compiled as modules, this registration will happen when the module is loaded. Once the `sysfs` filesystem is mounted (on `/sys`), data which the built-in drivers registered with `sysfs` are available to userspace processes and to **udevd** for device node creation.

11.5.2.2. Udev Bootscript

The **S10udev** initscript takes care of creating device nodes when Linux is booted. The script unsets the uevent handler from the default of **/sbin/hotplug**. This is done because the kernel no longer needs to call out to an external binary. Instead **udevd** will listen on a netlink socket for uevents that the kernel raises. Next, the

bootscript copies any static device nodes that exist in `/lib/udev/devices` to `/dev`. This is necessary because some devices, directories, and symlinks are needed before the dynamic device handling processes are available during the early stages of booting a system. Creating static device nodes in `/lib/udev/devices` also provides an easy workaround for devices that are not supported by the dynamic device handling infrastructure. The bootscript then starts the Udev daemon, **udevd**, which will act on any uevents it receives. Finally, the bootscript forces the kernel to replay uevents for any devices that have already been registered and then waits for **udevd** to handle them.

11.5.2.3. Device Node Creation

To obtain the right major and minor number for a device, Udev relies on the information provided by `sysfs` in `/sys`. For example, `/sys/class/tty/vcs/dev` contains the string "7:0". This string is used by **udevd** to create a device node with major number *7* and minor *0*. The names and permissions of the nodes created under the `/dev` directory are determined by rules specified in the files within the `/etc/udev/rules.d/` directory. These are numbered in a similar fashion to the CLFS-Bootscripts package. If **udevd** can't find a rule for the device it is creating, it will default permissions to *660* and ownership to *root:root*. Documentation on the syntax of the Udev rules configuration files are available in `/usr/share/doc/udev-096/index.html`

11.5.2.4. Module Loading

Device drivers compiled as modules may have aliases built into them. Aliases are visible in the output of the **modinfo** program and are usually related to the bus-specific identifiers of devices supported by a module. For example, the *snd-fm801* driver supports PCI devices with vendor ID 0x1319 and device ID 0x0801, and has an alias of "pci:v00001319d00000801sv*sd*bc04sc01i*". For most devices, the bus driver exports the alias of the driver that would handle the device via `sysfs`. E.g., the `/sys/bus/pci/devices/0000:00:0d.0/modalias` file might contain the string "pci:v00001319d00000801sv00001319sd00001319bc04sc01i00". The rules that CLFS installs will cause **udevd** to call out to **/sbin/modprobe** with the contents of the `MODALIAS` uevent environment variable (that should be the same as the contents of the `modalias` file in sysfs), thus loading all modules whose aliases match this string after wildcard expansion.

In this example, this means that, in addition to *snd-fm801*, the obsolete (and unwanted) *forte* driver will be loaded if it is available. See below for ways in which the loading of unwanted drivers can be prevented.

The kernel itself is also able to load modules for network protocols, filesystems and NLS support on demand.

11.5.2.5. Handling Hotpluggable/Dynamic Devices

When you plug in a device, such as a Universal Serial Bus (USB) MP3 player, the kernel recognizes that the device is now connected and generates a uevent. This uevent is then handled by **udevd** as described above.

11.5.3. Problems with Loading Modules and Creating Devices

There are a few possible problems when it comes to automatically creating device nodes.

11.5.3.1. A kernel module is not loaded automatically

Udev will only load a module if it has a bus-specific alias and the bus driver properly exports the necessary aliases to `sysfs`. In other cases, one should arrange module loading by other means. With Linux-2.6.17.13, Udev is known to load properly-written drivers for INPUT, IDE, PCI, USB, SCSI, SERIO and FireWire devices.

To determine if the device driver you require has the necessary support for Udev, run **modinfo** with the module name as the argument. Now try locating the device directory under `/sys/bus` and check whether there is a `modalias` file there.

If the `modalias` file exists in `sysfs`, the driver supports the device and can talk to it directly, but doesn't have the alias, it is a bug in the driver. Load the driver without the help from Udev and expect the issue to be fixed later.

If there is no `modalias` file in the relevant directory under `/sys/bus`, this means that the kernel developers have not yet added modalias support to this bus type. With Linux-2.6.17.13, this is the case with ISA busses. Expect this issue to be fixed in later kernel versions.

Udev is not intended to load “wrapper” drivers such as *snd-pcm-oss* and non-hardware drivers such as *loop* at all.

11.5.3.2. A kernel module is not loaded automatically, and Udev is not intended to load it

If the “wrapper” module only enhances the functionality provided by some other module (e.g., *snd-pcm-oss* enhances the functionality of *snd-pcm* by making the sound cards available to OSS applications), configure **modprobe** to load the wrapper after Udev loads the wrapped module. To do this, add an “install” line in `/etc/modprobe.conf`. For example:

```
install snd-pcm /sbin/modprobe -i snd-pcm ; \
    /sbin/modprobe snd-pcm-oss ; true
```

If the module in question is not a wrapper and is useful by itself, configure the **S05modules** bootscript to load this module on system boot. To do this, add the module name to the `/etc/sysconfig/modules` file on a separate line. This works for wrapper modules too, but is suboptimal in that case.

11.5.3.3. Udev loads some unwanted module

Either don't build the module, or blacklist it in `/etc/modprobe.conf` file as done with the *forte* module in the example below:

```
blacklist forte
```

Blacklisted modules can still be loaded manually with the explicit **modprobe** command.

11.5.3.4. Udev creates a device incorrectly, or makes a wrong symlink

This usually happens if a rule unexpectedly matches a device. For example, a poorly-writen rule can match both a SCSI disk (as desired) and the corresponding SCSI generic device (incorrectly) by vendor. Find the offending rule and make it more specific.

11.5.3.5. Udev rule works unreliably

This may be another manifestation of the previous problem. If not, and your rule uses `sysfs` attributes, it may be a kernel timing issue, to be fixed in later kernels. For now, you can work around it by creating a rule that waits for the used `sysfs` attribute and appending it to the `/etc/udev/rules.d/10-wait_for_sysfs.rules` file. Please notify the CLFS Development list if you do so and it helps.

11.5.3.6. Udev does not create a device

Further text assumes that the driver is built statically into the kernel or already loaded as a module, and that you have already checked that Udev doesn't create a misnamed device.

Udev has no information needed to create a device node if a kernel driver does not export its data to `sysfs`. This is most common with third party drivers from outside the kernel tree. Create a static device node in `/lib/udev/devices` with the appropriate major/minor numbers (see the file `devices.txt` inside the kernel documentation or the documentation provided by the third party driver vendor). The static device node will be copied to `/dev` by the **S10udev** bootscript.

11.5.3.7. Device naming order changes randomly after rebooting

This is due to the fact that Udev, by design, handles uevents and loads modules in parallel, and thus in an unpredictable order. This will never be “fixed”. You should not rely upon the kernel device names being stable. Instead, create your own rules that make symlinks with stable names based on some stable attributes of the device, such as a serial number or the output of various *_id utilities installed by Udev. See Section 11.13, “Creating custom symlinks to devices” and Section 11.14, “Configuring the network Script” for examples.

11.5.4. Useful Reading

Additional helpful documentation is available at the following sites:

- A Userspace Implementation of `devfs`
  http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf
- udev FAQ
  http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ
- The `sysfs` Filesystem
  http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

11.6. Configuring the setclock Script

The **setclock** script reads the time from the hardware clock, also known as the BIOS or the Complementary Metal Oxide Semiconductor (CMOS) clock. If the hardware clock is set to UTC, this script will convert the hardware clock's time to the local time using the `/etc/localtime` file (which tells the **hwclock** program which timezone the user is in). There is no way to detect whether or not the hardware clock is set to UTC, so this needs to be configured manually.

If you cannot remember whether or not the hardware clock is set to UTC, find out by running the **`hwclock --localtime --show`** command. This will display what the current time is according to the hardware clock. If this time matches whatever your watch says, then the hardware clock is set to local time. If the output from **hwclock** is not local time, chances are it is set to UTC time. Verify this by adding or subtracting the proper amount of hours for the timezone to the time shown by **hwclock**. For example, if you are currently in the MST timezone, which is also known as GMT -0700, add seven hours to the local time.

Change the value of the `UTC` variable below to a value of `0` (zero) if the hardware clock is *not* set to UTC time.

Create a new file `/etc/sysconfig/clock` by running the following:

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock

UTC=1

# End /etc/sysconfig/clock
EOF
```

A good hint explaining how to deal with time on CLFS is available at *http://www.linuxfromscratch.org/hints/downloads/files/time.txt*. It explains issues such as time zones, UTC, and the `TZ` environment variable.

11.7. Configuring the Linux Console

This section discusses how to configure the **console** bootscript that sets up the keyboard map and the console font. If non-ASCII characters (e.g., the British pound sign and Euro character) will not be used and the keyboard is a U.S. one, skip this section. Without the configuration file, the **console** bootscript will do nothing.

The **console** script reads the `/etc/sysconfig/console` file for configuration information. Decide which keymap and screen font will be used. Various language-specific HOWTO's can also help with this (see *http://www.tldp.org/HOWTO/HOWTO-INDEX/other-lang.html*. A pre-made `/etc/sysconfig/console` file with known settings for several countries was installed with the CLFS-Bootscripts package, so the relevant section can be uncommented if the country is supported. If still in doubt, look in the `/usr/share/kbd` directory for valid keymaps and screen fonts. Read `loadkeys(1)` and `setfont(8)` to determine the correct arguments for these programs. Once decided, create the configuration file with the following command:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="[arguments for loadkeys]"
FONT="[arguments for setfont]"
EOF
```

For example, for Spanish users who also want to use the Euro character (accessible by pressing AltGr+E), the following settings are correct:

```
cat >/etc/sysconfig/console <<"EOF"
KEYMAP="es euro2"
FONT="lat9-16 -u iso01"
EOF
```

Note

The `FONT` line above is correct only for the ISO 8859-15 character set. If using ISO 8859-1 and, therefore, a pound sign instead of Euro, the correct `FONT` line would be:

```
FONT="lat1-16"
```

If the `KEYMAP` or `FONT` variable is not set, the **console** initscript will not run the corresponding program.

In some keymaps, the Backspace and Delete keys send characters different from ones in the default keymap built into the kernel. This confuses some applications. For example, Emacs displays its help (instead of erasing the character before the cursor) when Backspace is pressed. To check if the keymap in use is affected (this works only for i386 keymaps):

```
zgrep '\W14\W' [/path/to/your/keymap]
```

If the keycode 14 is Backspace instead of Delete, create the following keymap snippet to fix this issue:

```
mkdir -pv /etc/kbd && cat > /etc/kbd/bs-sends-del <<"EOF"
                  keycode  14 = Delete Delete Delete Delete
              alt keycode  14 = Meta_Delete
       altgr alt keycode  14 = Meta_Delete
                  keycode 111 = Remove
   altgr control keycode 111 = Boot
```

```
      control alt keycode 111 = Boot
altgr control alt keycode 111 = Boot
EOF
```

Tell the **console** script to load this snippet after the main keymap:

```
cat >>/etc/sysconfig/console <<"EOF"
KEYMAP_CORRECTIONS="/etc/kbd/bs-sends-del"
EOF
```

To compile the keymap directly into the kernel instead of setting it every time from the **console** bootscript, follow the instructions given in Section 12.3, “Linux-2.6.17.13.” Doing this ensures that the keyboard will always work as expected, even when booting into maintenance mode (by passing `init=/bin/sh` to the kernel), because the **console** bootscript will not be run in that situation. Additionally, the kernel will not set the screen font automatically. This should not pose many problems because ASCII characters will be handled correctly, and it is unlikely that a user would need to rely on non-ASCII characters while in maintenance mode.

Since the kernel will set up the keymap, it is possible to omit the `KEYMAP` variable from the `/etc/sysconfig/console` configuration file. It can also be left in place, if desired, without consequence. Keeping it could be beneficial if running several different kernels where it is difficult to ensure that the keymap is compiled into every one of them.

11.8. 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.

11.9. Creating the /etc/inputrc File

The `/etc/inputrc` file deals with mapping the keyboard for specific situations. This file is the start-up file used by Readline — the input-related library — used by Bash and most other shells.

Most people do not need user-specific keyboard mappings so the command below creates a global `/etc/inputrc` used by everyone who logs in. If you later decide you need to override the defaults on a per-user basis, you can create a `.inputrc` file in the user's home directory with the modified mappings.

For more information on how to edit the `inputrc` file, see **info bash** under the *Readline Init File* section. **info readline** is also a good source of information.

Below is a generic global `inputrc` along with comments to explain what the various options do. Note that comments cannot be on the same line as commands. Create the file using the following command:

```
cat > /etc/inputrc << "EOF"
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>

# Allow the command prompt to wrap to the next line
set horizontal-scroll-mode Off

# Enable 8bit input
set meta-flag On
set input-meta On

# Turns off 8th bit stripping
set convert-meta Off

# Keep the 8th bit for display
set output-meta On

# none, visible or audible
set bell-style none

# All of the following map the escape sequence of the
# value contained inside the 1st argument to the
# readline specific functions

"\eOd": backward-word
"\eOc": forward-word

# for linux console
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert

# for xterm
"\eOH": beginning-of-line
"\eOF": end-of-line
```

```
# for Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line

# End /etc/inputrc
EOF
```

11.10. The Bash Shell Startup Files

The shell program **/bin/bash** (hereafter referred to as “the shell”) uses a collection of startup files to help create an environment to run in. Each file has a specific use and may affect login and interactive environments differently. The files in the `/etc` directory provide global settings. If an equivalent file exists in the home directory, it may override the global settings.

An interactive login shell is started after a successful login, using **/bin/login**, by reading the `/etc/passwd` file. An interactive non-login shell is started at the command-line (e.g., `[prompt]$` **/bin/bash**). A non-interactive shell is usually present when a shell script is running. It is non-interactive because it is processing a script and not waiting for user input between commands.

For more information, see **info bash** under the *Bash Startup Files and Interactive Shells* section.

The files `/etc/profile` and `~/.bash_profile` are read when the shell is invoked as an interactive login shell.

The base `/etc/profile` below sets some environment variables necessary for native language support. Setting them properly results in:

- The output of programs translated into the native language
- Correct classification of characters into letters, digits and other classes. This is necessary for **bash** to properly accept non-ASCII characters in command lines in non-English locales
- The correct alphabetical sorting order for the country
- Appropriate default paper size
- Correct formatting of monetary, time, and date values

This script also sets the `INPUTRC` environment variable that makes Bash and Readline use the `/etc/inputrc` file created earlier.

Replace *`[ll]`* below with the two-letter code for the desired language (e.g., “en”) and *`[CC]`* with the two-letter code for the appropriate country (e.g., “GB”). *`[charmap]`* should be replaced with the canonical charmap for your chosen locale.

The list of all locales supported by Glibc can be obtained by running the following command:

```
locale -a
```

Locales can have a number of synonyms, e.g. “ISO-8859-1” is also referred to as “iso8859-1” and “iso88591”. Some applications cannot handle the various synonyms correctly, so it is safest to choose the canonical name for a particular locale. To determine the canonical name, run the following command, where *`[locale name]`* is the output given by **locale -a** for your preferred locale (“en_GB.iso88591” in our example).

```
LC_ALL=[locale name] locale charmap
```

For the “en_GB.iso88591” locale, the above command will print:

```
ISO-8859-1
```

This results in a final locale setting of “en_GB.ISO-8859-1”. It is important that the locale found using the heuristic above is tested prior to it being added to the Bash startup files:

```
LC_ALL=[locale name] locale country
LC_ALL=[locale name] locale language
LC_ALL=[locale name] locale charmap
LC_ALL=[locale name] locale int_curr_symbol
LC_ALL=[locale name] locale int_prefix
```

The above commands should print the language name, the character encoding used by the locale, the local currency, and the prefix to dial before the telephone number in order to get into the country. If any of the commands above fail with a message similar to the one shown below, this means that your locale was either not installed in Chapter 10 or is not supported by the default installation of Glibc.

```
locale: Cannot set LC_* to default locale: No such file or directory
```

If this happens, you should either install the desired locale using the **localedef** command, or consider choosing a different locale. Further instructions assume that there are no such error messages from Glibc.

Some packages beyond CLFS may also lack support for your chosen locale. One example is the X library (part of the X Window System), which outputs the following error message:

```
Warning: locale not supported by Xlib, locale set to C
```

Sometimes it is possible to fix this by removing the charmap part of the locale specification, as long as that does not change the character map that Glibc associates with the locale (this can be checked by running the **locale charmap** command in both locales). For example, one would have to change "de_DE.ISO-8859-15@euro" to "de_DE@euro" in order to get this locale recognized by Xlib.

Other packages can also function incorrectly (but may not necessarily display any error messages) if the locale name does not meet their expectations. In those cases, investigating how other Linux distributions support your locale might provide some useful information.

Once the proper locale settings have been determined, create the `/etc/profile` file:

```
cat > /etc/profile << "EOF"
# Begin /etc/profile

export LANG=[ll]_[CC].[charmap]
export INPUTRC=/etc/inputrc

# End /etc/profile
EOF
```

Note

The “C” (default) and “en_US” (the recommended one for United States English users) locales are different.

Setting the keyboard layout, screen font, and locale-related environment variables are the only internationalization steps needed to support locales that use ordinary single-byte encodings and left-to-right writing direction. More complex cases (including UTF-8 based locales) require additional steps and additional

patches because many applications tend to not work properly under such conditions. These steps and patches are not included in the CLFS book and such locales are not yet supported by CLFS.

11.11. Configuring the localnet Script

Part of the job of the **localnet** script is setting the system's hostname. This needs to be configured in the `/etc/sysconfig/network` file.

Create the `/etc/sysconfig/network` file and enter a hostname by running:

```
echo "HOSTNAME=[clfs]" > /etc/sysconfig/network
```

`[clfs]` needs to be replaced with the name given to the computer. Do not enter the Fully Qualified Domain Name (FQDN) here. That information will be put in the `/etc/hosts` file in the next section.

11.12. Customizing the /etc/hosts File

If a network card is to be configured, decide on the IP address, FQDN, and possible aliases for use in the `/etc/hosts` file. The syntax is:

```
<IP address> myhost.example.org aliases
```

Unless the computer is to be visible to the Internet (i.e., there is a registered domain and a valid block of assigned IP addresses—most users do not have this), make sure that the IP address is in the private network IP address range. Valid ranges are:

```
  Class Networks
  A     10.0.0.0
  B     172.16.0.0 through 172.31.0.255
  C     192.168.0.0 through 192.168.255.255
```

A valid IP address could be 192.168.1.1. A valid FQDN for this IP could be `www.linuxfromscratch.org` (not recommended because this is a valid registered domain address and could cause domain name server issues).

Even if not using a network card, an FQDN is still required. This is necessary for certain programs to operate correctly.

Create the `/etc/hosts` file by running:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
[192.168.1.1] [<HOSTNAME>.example.org] [HOSTNAME]

# End /etc/hosts (network card version)
EOF
```

The *`[192.168.1.1]`* and *`[<HOSTNAME>.example.org]`* values need to be changed for specific users or requirements (if assigned an IP address by a network/system administrator and the machine will be connected to an existing network).

If a network card is not going to be configured, create the `/etc/hosts` file by running:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)

127.0.0.1 [<HOSTNAME>.example.org] [HOSTNAME] localhost

# End /etc/hosts (no network card version)
EOF
```

11.13. Creating custom symlinks to devices

11.13.1. CD-ROM symlinks

Some software that you may want to install later (e.g., various media players) expect the /dev/cdrom and /dev/dvd symlinks to exist. Also, it may be convenient to put references to those symlinks into `/etc/fstab`. For each of your CD-ROM devices, find the corresponding directory under `/sys` (e.g., this can be `/sys/block/hdd`) and run a command similar to the following:

```
udevtest /block/hdd
```

Look at the lines containing the output of various *_id programs.

There are two approaches to creating symlinks. The first one is to use the model name and the serial number, the second one is based on the location of the device on the bus. If you are going to use the first approach, create a file similar to the following:

```
cat >/etc/udev/rules.d/82-cdrom.rules << EOF

# Custom CD-ROM symlinks
SUBSYSTEM=="block", ENV{ID_MODEL}=="SAMSUNG_CD-ROM_SC-148F", \
    ENV{ID_REVISION}=="PS05", SYMLINK+="cdrom"
SUBSYSTEM=="block", ENV{ID_MODEL}=="PHILIPS_CDD5301", \
    ENV{ID_SERIAL}=="5VO1306DM00190", SYMLINK+="cdrom1 dvd"

EOF
```

Note

Although the examples in this book work properly, be aware that Udev does not recognize the backslash for line continuation. If modifying Udev rules with an editor, be sure to leave each rule on one physical line.

This way, the symlinks will stay correct even if you move the drives to different positions on the IDE bus, but the `/dev/cdrom` symlink won't be created if you replace the old SAMSUNG CD-ROM with a new drive.

The SUBSYSTEM=="block" key is needed in order to avoid matching SCSI generic devices. Without it, in the case with SCSI CD-ROMs, the symlinks will sometimes point to the correct `/dev/srX` devices, and sometimes to `/dev/sgX`, which is wrong.

The second approach yields:

```
cat >/etc/udev/rules.d/82-cdrom.rules << EOF

# Custom CD-ROM symlinks
SUBSYSTEM=="block", ENV{ID_TYPE}=="cd", \
    ENV{ID_PATH}=="pci-0000:00:07.1-ide-0:1", SYMLINK+="cdrom"
SUBSYSTEM=="block", ENV{ID_TYPE}=="cd", \
    ENV{ID_PATH}=="pci-0000:00:07.1-ide-1:1", SYMLINK+="cdrom1 dvd"

EOF
```

This way, the symlinks will stay correct even if you replace drives with different models, but place them to the old positions on the IDE bus. The ENV{ID_TYPE}=="cd" key makes sure that the symlink disappears if you put something other than a CD-ROM in that position on the bus.

Of course, it is possible to mix the two approaches.

11.13.2. Dealing with duplicate devices

As explained in Section 11.5, “Device and Module Handling on a CLFS System”, the order in which devices with the same function appear in `/dev` is essentially random. E.g., if you have a USB web camera and a TV tuner, sometimes `/dev/video0` refers to the camera and `/dev/video1` refers to the tuner, and sometimes after a reboot the order changes to the opposite one. For all classes of hardware except sound cards and network cards, this is fixable by creating udev rules for custom persistent symlinks. The case of network cards is covered separately in Section 11.14, “Configuring the network Script”, and sound card configuration can be found in *BLFS*.

For each of your devices that is likely to have this problem (even if the problem doesn't exist in your current Linux distribution), find the corresponding directory under `/sys/class` or `/sys/block`. For video devices, this may be `/sys/class/video4linux/video`*X*. Figure out the attributes that identify the device uniquely (usually, vendor and product IDs and/or serial numbers work):

```
udevinfo -a -p /sys/class/video4linux/video0
```

Then write rules that create the symlinks, e.g.:

```
cat >/etc/udev/rules.d/83-duplicate_devs.rules << EOF

# Persistent symlinks for webcam and tuner
KERNEL=="video*", SYSFS{idProduct}=="1910", SYSFS{idVendor}=="0d81", \
    SYMLINK+="webcam"
KERNEL=="video*", SYSFS{device}=="0x036f", SYSFS{vendor}=="0x109e", \
    SYMLINK+="tvtuner"

EOF
```

The result is that `/dev/video0` and `/dev/video1` devices still refer randomly to the tuner and the web camera (and thus should never be used directly), but there are symlinks `/dev/tvtuner` and `/dev/webcam` that always point to the correct device.

More information on writing Udev rules can be found in `/usr/share/doc/udev-096/index.html`.

11.14. Configuring the network Script

This section only applies if a network card is to be configured.

If a network card will not be used, there is likely no need to create any configuration files relating to network cards. If that is the case, remove the `network` symlinks from all run-level directories (`/etc/rc.d/rc*.d`).

11.14.1. Creating stable names for network interfaces

Instructions in this section are optional if you have only one network card.

With Udev and modular network drivers, the network interface numbering is not persistent across reboots by default, because the drivers are loaded in parallel and, thus, in random order. For example, on a computer having two network cards made by Intel and Realtek, the network card manufactured by Intel may become `eth0` and the Realtek card becomes `eth1`. In some cases, after a reboot the cards get renumbered the other way around. To avoid this, create Udev rules that assign stable names to network cards based on their MAC addresses or bus positions.

If you are going to use MAC addresses to identify your network cards, find the addresses with the following command:

```
grep -H . /sys/class/net/*/address
```

For each network card (but not for the loopback interface), invent a descriptive name, such as “realtek”, and create Udev rules similar to the following:

```
cat > /etc/udev/rules.d/26-network.rules << EOF
ACTION=="add", SUBSYSTEM=="net", SYSFS{address}=="00:e0:4c:12:34:56", \
    NAME="realtek"
ACTION=="add", SUBSYSTEM=="net", SYSFS{address}=="00:a0:c9:78:9a:bc", \
    NAME="intel"
EOF
```

Note

Although the examples in this book work properly, be aware that Udev does not recognize the backslash for line continuation. If modifying Udev rules with an editor, be sure to leave each rule on one physical line.

If you are going to use the bus position as a key, create Udev rules similar to the following:

```
cat > /etc/udev/rules.d/26-network.rules << EOF
ACTION=="add", SUBSYSTEM=="net", BUS=="pci", ID=="0000:00:0c.0", \
    NAME="realtek"
ACTION=="add", SUBSYSTEM=="net", BUS=="pci", ID=="0000:00:0d.0", \
    NAME="intel"
EOF
```

These rules will always rename the network cards to “realtek” and “intel”, independently of the original numbering provided by the kernel (i.e.: the original “eth0” and “eth1” interfaces will no longer exist, unless you put such “descriptive” names in the NAME key). Use the descriptive names from the Udev rules instead of

“eth0” in the network interface configuration files below.

Note that the rules above don't work for every setup. For example, MAC-based rules break when bridges or VLANs are used, because bridges and VLANs have the same MAC address as the network card. One wants to rename only the network card interface, not the bridge or VLAN interface, but the example rule matches both. If you use such virtual interfaces, you have two potential solutions. One is to add the DRIVER=="?*" key after SUBSYSTEM=="net" in MAC-based rules which will stop matching the virtual interfaces. This is known to fail with some older Ethernet cards because they don't have the DRIVER variable in the uevent and thus the rule does not match with such cards. Another solution is to switch to rules that use the bus position as a key.

The second known non-working case is with wireless cards using the MadWifi or HostAP drivers, because they create at least two interfaces with the same MAC address and bus position. For example, the Madwifi driver creates both an athX and a wifiX interface where X is a digit. To differentiate these interfaces, add an appropriate KERNEL parameter such as KERNEL=="ath*" after SUBSYSTEM=="net".

There may be other cases where the rules above don't work. Currently, bugs on this topic are still being reported to Linux distributions, and no solution that covers every case is available.

11.14.2. Creating Network Interface Configuration Files

Which interfaces are brought up and down by the network script depends on the files and directories in the `/etc/sysconfig/network-devices` hierarchy. This directory should contain a sub-directory for each interface to be configured, such as `ifconfig.xyz`, where “xyz” is a network interface name. Inside this directory would be files defining the attributes to this interface, such as its IP address(es), subnet masks, and so forth.

The following command creates a sample `ipv4` file for the *eth0* device:

```
cd /etc/sysconfig/network-devices &&
mkdir -v ifconfig.eth0 &&
cat > ifconfig.eth0/ipv4 << "EOF"
ONBOOT=yes
SERVICE=ipv4-static
IP=192.168.1.1
GATEWAY=192.168.1.2
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

The values of these variables must be changed in every file to match the proper setup. If the `ONBOOT` variable is set to “yes” the network script will bring up the Network Interface Card (NIC) during booting of the system. If set to anything but “yes” the NIC will be ignored by the network script and not be brought up.

The `SERVICE` variable defines the method used for obtaining the IP address. The CLFS-Bootscripts package has a modular IP assignment format, and creating additional files in the `/etc/sysconfig/network-devices/services` directory allows other IP assignment methods. This is commonly used for Dynamic Host Configuration Protocol (DHCP), which is addressed in the BLFS book.

The `GATEWAY` variable should contain the default gateway IP address, if one is present. If not, then comment out the variable entirely.

The `PREFIX` variable needs to contain the number of bits used in the subnet. Each octet in an IP address is 8 bits. If the subnet's netmask is 255.255.255.0, then it is using the first three octets (24 bits) to specify the

network number. If the netmask is 255.255.255.240, it would be using the first 28 bits. Prefixes longer than 24 bits are commonly used by DSL and cable-based Internet Service Providers (ISPs). In this example (PREFIX=24), the netmask is 255.255.255.0. Adjust the PREFIX variable according to your specific subnet.

11.14.3. Creating the /etc/resolv.conf File

If the system is going to be connected to the Internet, it will need some means of Domain Name Service (DNS) name resolution to resolve Internet domain names to IP addresses, and vice versa. This is best achieved by placing the IP address of the DNS server, available from the ISP or network administrator, into /etc/resolv.conf. Create the file by running the following:

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

domain [Your Domain Name]
nameserver [IP address of your primary nameserver]
nameserver [IP address of your secondary nameserver]

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

Replace *[IP address of the nameserver]* with the IP address of the DNS most appropriate for the setup. There will often be more than one entry (requirements demand secondary servers for fallback capability). If you only need or want one DNS server, remove the second *nameserver* line from the file. The IP address may also be a router on the local network.

Chapter 12. Making the CLFS System Bootable

12.1. Introduction

It is time to make the CLFS system bootable. This chapter discusses creating an `fstab` file, building a kernel for the new CLFS system, and installing the boot loader so that the CLFS system can be selected for booting at startup.

12.2. Creating the /etc/fstab File

The `/etc/fstab` file is used by some programs to determine where file systems are to be mounted by default, in which order, and which must be checked (for integrity errors) prior to mounting. Create a new file systems table like this:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab

# file system  mount-point  type   options          dump  fsck
#                                                        order

/dev/[xxx]     /            [fff]  defaults         1     1
/dev/[yyy]     swap         swap   pri=1            0     0
proc           /proc        proc   defaults         0     0
sysfs          /sys         sysfs  defaults         0     0
devpts         /dev/pts     devpts gid=4,mode=620   0     0
shm            /dev/shm     tmpfs  defaults         0     0
# End /etc/fstab
EOF
```

Replace *[xxx]*, *[yyy]*, and *[fff]* with the values appropriate for the system, for example, `hda2`, `hda5`, and `ext2`. For details on the six fields in this file, see **man 5 fstab**.

The `/dev/shm` mount point for `tmpfs` is included to allow enabling POSIX-shared memory. The kernel must have the required support built into it for this to work (more about this is in the next section). Please note that very little software currently uses POSIX-shared memory. Therefore, consider the `/dev/shm` mount point optional. For more information, see `Documentation/filesystems/tmpfs.txt` in the kernel source tree.

12.3. Linux-2.6.17.13

The Linux package contains the Linux kernel.

12.3.1. Installation of the kernel

Building the kernel involves a few steps—configuration, compilation, and installation. Read the `README` file in the kernel source tree for alternative methods to the way this book configures the kernel.

The following patch fixes on intialization issue with the tulip network driver:

```
patch -Np1 -i ../linux-2.6.17.13-tulip-1.patch
```

Prepare for compilation by running the following command:

```
make mrproper
```

This ensures that the kernel tree is absolutely clean. The kernel team recommends that this command be issued prior to each kernel compilation. Do not rely on the source tree being clean after un-tarring.

If, in Section 11.7, “Configuring the Linux Console,” it was decided to compile the keymap into the kernel, issue the command below:

```
loadkeys -m /lib/kbd/keymaps/[path to  keymap] > \
    drivers/char/defkeymap.c
```

For example, if using a Dutch keyboard, use `/lib/kbd/keymaps/i386/qwerty/nl.map.gz`.

Configure the kernel via a menu-driven interface. Please note that the udev bootscript requires "rtc" and "tmpfs" to be enabled and built into the kernel, not as modules. BLFS has some information regarding particular kernel configuration requirements of packages outside of CLFS at *http://www.linuxfromscratch.org/blfs/view/svn/longindex.html#kernel-config-index*:

```
make menuconfig
```

Alternatively, **make oldconfig** may be more appropriate in some situations. See the `README` file for more information.

If desired, skip kernel configuration by copying the kernel config file, `.config`, from the host system (assuming it is available) to the root directory of the unpacked kernel sources. However, we do not recommend this option. It is often better to explore all the configuration menus and create the kernel configuration from scratch.

Compile the kernel image and modules:

```
make
```

If using kernel modules, an `/etc/modprobe.conf` file may be needed. Information pertaining to modules and kernel configuration is located in the kernel documentation in the `Documentation` directory of the kernel sources tree. Also, `modprobe.conf(5)` may be of interest.

Be very careful when reading other documentation relating to kernel modules because it usually applies to 2.4.x

kernels only. As far as we know, kernel configuration issues specific to Hotplug and Udev are not documented. The problem is that Udev will create a device node only if Hotplug or a user-written script inserts the corresponding module into the kernel, and not all modules are detectable by Hotplug. Note that statements like the one below in the `/etc/modprobe.conf` file do not work with Udev:

```
alias char-major-XXX some-module
```

Because of the complications with Udev and modules, we strongly recommend starting with a completely non-modular kernel configuration, especially if this is the first time using Udev.

Install the modules, if the kernel configuration uses them:

```
make modules_install
```

After kernel compilation is complete, additional steps are required to complete the installation. Some files need to be copied to the `/boot` directory.

Issue the following command to install the kernel:

```
cp arch/x86_64/boot/bzImage /boot/clfskernel-2.6.17.13
```

`System.map` is a symbol file for the kernel. It maps the function entry points of every function in the kernel API, as well as the addresses of the kernel data structures for the running kernel. Issue the following command to install the map file:

```
cp System.map /boot/System.map-2.6.17.13
```

The kernel configuration file `.config` produced by the **make menuconfig** step above contains all the configuration selections for the kernel that was just compiled. It is a good idea to keep this file for future reference:

```
cp .config /boot/config-2.6.17.13
```

It is important to note that the files in the kernel source directory are not owned by `root`. Whenever a package is unpacked as user `root` (like we did inside chroot), the files have the user and group IDs of whatever they were on the packager's computer. This is usually not a problem for any other package to be installed because the source tree is removed after the installation. However, the Linux source tree is often retained for a long time. Because of this, there is a chance that whatever user ID the packager used will be assigned to somebody on the machine. That person would then have write access to the kernel source.

If the kernel source tree is going to retained, run **chown -R 0:0** on the `linux-2.6.17.13` directory to ensure all files are owned by user `root`.

Warning

Some kernel documentation recommends creating a symlink from `/usr/src/linux` pointing to the kernel source directory. This is specific to kernels prior to the 2.6 series and *must not* be created on a CLFS system as it can cause problems for packages you may wish to build once your base CLFS system is complete.

Also, the headers in the system's `include` directory should *always* be the ones against which Glibc was compiled (from the Linux-Headers package) and should *never* be replaced by the kernel

headers.

12.3.2. Contents of Linux

Installed files: config-[linux-version], clfskernel-[linux-version], and System.map-[linux-version]

Short Descriptions

`config-[linux-version]`	Contains all the configuration selections for the kernel
`clfskernel-[linux-version]`	The engine of the Linux system. When turning on the computer, the kernel is the first part of the operating system that gets loaded. It detects and initializes all components of the computer's hardware, then makes these components available as a tree of files to the software and turns a single CPU into a multitasking machine capable of running scores of programs seemingly at the same time.
`System.map-[linux-version]`	A list of addresses and symbols; it maps the entry points and addresses of all the functions and data structures in the kernel

12.4. Making the CLFS System Bootable

Your shiny new CLFS system is almost complete. One of the last things to do is to ensure that the system can be properly booted. The instructions below apply only to computers using Lilo, which in the context of this book means x86_64 Pure64 systems. Information on “boot loading” for other architectures should be available in the usual resource-specific locations for those architectures.

Boot loading can be a complex area, so a few cautionary words are in order. Be familiar with the current boot loader and any other operating systems present on the hard drive(s) that need to be bootable. Make sure that an emergency boot disk is ready to “rescue” the computer if the computer becomes unusable (un-bootable).

If you have multiple systems on your machine using a different bootloader such as GRUB, you may prefer to use that instead - consult the appropriate documentation. The rest of this section assumes you are going to use Lilo.

Earlier, we compiled and installed the Lilo boot loader software in preparation for this step. The procedure involves writing a boot image to a specific location on the hard drive. We highly recommend using mkrescue to create a Lilo boot CD (using e.g. dvdrecord from dvdrtools) as a backup (this requires loopback block device support in the kernel).

Normally, you interact with Lilo by using the cursor and `enter` keys to select from the available option(s), but sometimes it is necessary to add other boot options, such as e.g. 'init=/bin/bash' to debug boot failures. The more your keyboard layout differs from the US qwerty layout, the harder it becomes to type boot options unless Lilo knows about your keyboard layout. So, we will create a key table for Lilo (.ktl) file - at one point in the documentation these are referred to as .klt files, which may be a typo, but has been followed by some distros. The name, and location, are not important but it is conventional to put these in /boot with the name representing the key layout. For a British keyboard layout, the following command will achieve this:

```
keytab-lilo.pl uk >/boot/uk.ktl
```

The argument to the command is the name of the keymap, or if necessary you can specify the full path to the keymap. Use whatever is appropriate for your keyboard.

When the x86 CLFS book used to include Lilo, it advised against running it from chroot in case the MBR became corrupted. Provided you have /proc mounted and have device special files for the disks, it seems to be safe to run recent versions of Lilo in chroot, although it is always possible that an updated bootloader, or defective configuration file, may render the system unbootable.

The next step is to create /etc/lilo.conf:

```
cat > /etc/lilo.conf << "EOF"
# Begin /etc/lilo.conf
# lilo.conf
#
# global options
boot=/dev/<bootdisk>
keytable=/boot/<keytable>
lba32
map=/boot/map
prompt

# set the name of the default image to boot
```

```
default=clfs

# define an image
image=/boot/clfskernel
    label=clfs
    root=/dev/<partition>
    read-only
# optionally add parameters to pass, e.g.
#   append="video=radeonfb:1024x768-16@70"

# repeat for any other kernel images

# optionally, add legacy operating systems
# see man lilo.conf for examples
EOF
```

Replace <bootdisk> with the name of the disk (or partition) on which the boot sector is to be written, e.g. sda. Replace <keytable> with the name of the keytable file you created, and <partition> with the name of the root partition for the new system.

Warning

The following command will overwrite the current boot loader. Do not run the command if this is not desired.

Run Lilo:

```
/sbin/lilo -v
```

Note

People who have been used to GRUB need to be aware that Lilo works differently - in particular, you cannot edit the available choices as you can in the **grub** shell, and Lilo records the block addresses of the kernels into the boot blocks each time /sbin/lilo is run. This means that when you compile a new kernel, you have to add it to /etc/lilo.conf and rerun /sbin/lilo. It also means that if you recompile an existing kernel and save it to the same name you still have to rerun /sbin/lilo in case it now occupies different blocks on the filesystem.

If you are running multiple systems on this box and using Lilo, it is a good idea to ensure that each system is running the same version of Lilo, otherwise an old version may not be able to overwrite the bootloader from a newer version. You will also need to ensure that the copies of /etc/lilo.conf on each system are kept synchronised.

Chapter 13. The End

13.1. The End

Well done! The new CLFS system is installed! We wish you much success with your shiny new custom-built Linux system.

It may be a good idea to create an `/etc/clfs-release` file. By having this file, it is very easy for you (and for us if you need to ask for help at some point) to find out which CLFS version is installed on the system. Create this file by running:

```
echo 1.0.0rc6 > /etc/clfs-release
```

13.2. Get Counted

Now that you have finished the book, do you want to be counted as a CLFS user? Head over to *http://www.linuxfromscratch.org/cgi-bin/lfscounter.cgi* and register as a CLFS user by entering your name and the first CLFS version you have used.

13.3. Rebooting the System

If you built your final system using the boot method, just run **shutdown -r now** to reboot again, using your newly-built kernel instead of the miminal one currently in use. If you chrooted, there are a few more steps.

The system you have created in this book is quite minimal, and most likely will not have the functionality you would need to be able to continue forward. By installing a few extra packages from the BLFS book while still in our current chroot environment, you can leave yourself in a much better position to continue on once you reboot into your new CLFS installation. Installing a text mode web browser, such as Lynx, you can easily view the BLFS book in one virtual terminal, while building packages in another. The GPM package will also allow you to perform copy/paste actions in your virtual terminals. Lastly, if you are in a situation where static IP configuration does not meet your networking requirements, installing packages such as Dhcpcd or PPP at this point might also be useful.

Now that we have said that, lets move on to booting our shiny new CLFS installation for the first time! First exit from the chroot environment:

```
logout
```

Then unmount the virtual file systems:

```
umount ${CLFS}/dev/pts
umount ${CLFS}/dev/shm
umount ${CLFS}/dev
umount ${CLFS}/proc
umount ${CLFS}/sys
```

Unmount the CLFS file system itself:

```
umount ${CLFS}
```

If multiple partitions were created, unmount the other partitions before unmounting the main one, like this:

```
umount ${CLFS}/usr
umount ${CLFS}/home
umount ${CLFS}
```

Now, reboot the system with:

```
shutdown -r now
```

Assuming the boot loader was set up as outlined earlier, *CLFS 1.0.0rc6* will boot automatically.

When the reboot is complete, the CLFS system is ready for use and more software may be added to suit your needs.

13.4. What Now?

Thank you for reading this CLFS book. We hope that you have found this book helpful and have learned more about the system creation process.

Now that the CLFS system is installed, you may be wondering “What next?” To answer that question, we have compiled a list of resources for you.

- Maintenance

  Bugs and security notices are reported regularly for all software. Since a CLFS system is compiled from source, it is up to you to keep abreast of such reports. There are several online resources that track such reports, some of which are shown below:

  - Freshmeat.net (*http://freshmeat.net/*)

    Freshmeat can notify you (via email) of new versions of packages installed on your system.

  - *CERT* (Computer Emergency Response Team)

    CERT has a mailing list that publishes security alerts concerning various operating systems and applications. Subscription information is available at *http://www.us-cert.gov/cas/signup.html*.

  - Bugtraq

    Bugtraq is a full-disclosure computer security mailing list. It publishes newly discovered security issues, and occasionally potential fixes for them. Subscription information is available at *http://www.securityfocus.com/archive*.

- Beyond Linux From Scratch

  The Beyond Linux From Scratch book covers installation procedures for a wide range of software beyond the scope of the CLFS Book. The BLFS project is located at *http://www.linuxfromscratch.org/blfs/*.

- LFS Hints

  The LFS Hints are a collection of educational documents submitted by volunteers in the LFS community. The hints are available at *http://www.linuxfromscratch.org/hints/list.html*.

- Mailing lists

  There are several CLFS mailing lists you may subscribe to if you are in need of help, want to stay current with the latest developments, want to contribute to the project, and more. See Chapter 1 - Mailing Lists for more information.

- The Linux Documentation Project

  The goal of The Linux Documentation Project (TLDP) is to collaborate on all of the issues of Linux documentation. The TLDP features a large collection of HOWTOs, guides, and man pages. It is located at *http://www.tldp.org/*.

Part VI. Appendices

Appendix A. Acronyms and Terms

ABI Application Binary Interface

ALFS Automated Linux From Scratch

ALSA Advanced Linux Sound Architecture

API Application Programming Interface

ASCII American Standard Code for Information Interchange

ATA Advanced Technology Attachment (see IDE)

BIOS Basic Input/Output System

bless manipulate a filesystem so that OF will boot from it

BLFS Beyond Linux From Scratch

BSD Berkeley Software Distribution

chroot change root

CLFS Cross-Compiled Linux From Scratch

CMOS Complementary Metal Oxide Semiconductor

COS Class Of Service

CPU Central Processing Unit

CRC Cyclic Redundancy Check

DHCP Dynamic Host Configuration Protocol

DNS Domain Name Service

EGA Enhanced Graphics Adapter

ELF Executable and Linkable Format

EOF End of File

EQN equation

EVMS Enterprise Volume Management System

ext2 second extended file system

FAQ Frequently Asked Questions

FHS Filesystem Hierarchy Standard

FIFO First-In, First Out

FQDN Fully Qualified Domain Name

FTP File Transfer Protocol

GB Gibabytes
GCC GNU Compiler Collection
GID Group Identifier
GMT Greenwich Mean Time
HTML Hypertext Markup Language
IDE Integrated Drive Electronics
IEEE Institute of Electrical and Electronic Engineers
IO Input/Output
IP Internet Protocol
IPC Inter-Process Communication
IRC Internet Relay Chat
ISO International Organization for Standardization
ISP Internet Service Provider
KB Kilobytes
LED Light Emitting Diode
LFS Linux From Scratch
LSB Linux Standard Base
MB Megabytes
MBR Master Boot Record
MD5 Message Digest 5
NIC Network Interface Card
NLS Native Language Support
NNTP Network News Transport Protocol
NPTL Native POSIX Threading Library
OF Open Firmware
OSS Open Sound System
PCH Pre-Compiled Headers
PCRE Perl Compatible Regular Expression
PID Process Identifier
PTY pseudo terminal
QA Quality Assurance

QOS Quality Of Service

RAM Random Access Memory

RPC Remote Procedure Call

RTC Real Time Clock

SCO The Santa Cruz Operation

SATA Serial ATA

SGR Select Graphic Rendition

SHA1 Secure-Hash Algorithm 1

SMP Symmetric Multi-Processor

TLDP The Linux Documentation Project

TFTP Trivial File Transfer Protocol

TLS Thread-Local Storage

UID User Identifier

umask user file-creation mask

USB Universal Serial Bus

UTC Coordinated Universal Time

UUID Universally Unique Identifier

VC Virtual Console

VGA Video Graphics Array

VT Virtual Terminal

Appendix B. Acknowledgments

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

- *Gerard Beekmans* <gerard@linuxfromscratch.org> – LFS Creator, LFS Project Leader
- *Matthew Burgess* <matthew@linuxfromscratch.org> – LFS Project Leader, LFS Technical Writer/Editor, LFS Release Manager
- *Archaic* <archaic@linuxfromscratch.org> – LFS Technical Writer/Editor, HLFS Project Leader, BLFS Editor, Hints and Patches Project Maintainer
- *Nathan Coulson* <nathan@linuxfromscratch.org> – LFS-Bootscripts Maintainer
- *Bruce Dubbs* <bdubbs@linuxfromscratch.org> – BLFS Project Leader
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – LFS/BLFS/HLFS XML and XSL Maintainer
- *Jim Gifford* <jim@linuxfromscratch.org> – LFS Technical Writer, Patches Project Leader
- *Jeremy Huntwork* <jhuntwork@linuxfromscratch.org> – LFS Technical Writer, LFS LiveCD Maintainer, ALFS Project Leader
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Website Backend-Scripts Maintainer
- *Ryan Oliver* <ryan@linuxfromscratch.org> – LFS Toolchain Maintainer
- *James Robertson* <jwrober@linuxfromscratch.org> – Bugzilla Maintainer
- *Tushar Teredesai* <tushar@linuxfromscratch.org> – BLFS Book Editor, Hints and Patches Project Leader
- Countless other people on the various LFS and BLFS mailing lists who helped make this book possible by giving their suggestions, testing the book, and submitting bug reports, instructions, and their experiences with installing various packages.

Translators

- *Manuel Canales Esparcia* <macana@lfs-es.com> – Spanish LFS translation project
- *Johan Lenglet* <johan@linuxfromscratch.org> – French LFS translation project
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Portuguese LFS translation project
- *Thomas Reitelbach* <tr@erdfunkstelle.de> – German LFS translation project

Mirror Maintainers

North American Mirrors

- *Scott Kveton* <scott@osuosl.org> – lfs.oregonstate.edu mirror
- *Mikhail Pastukhov* <miha@xuy.biz> – lfs.130th.net mirror

- *William Astle* <lost@l-w.net> – ca.linuxfromscratch.org mirror
- *Jeremy Polen* <jpolen@rackspace.com> – us2.linuxfromscratch.org mirror
- *Tim Jackson* <tim@idge.net> – linuxfromscratch.idge.net mirror
- *Jeremy Utley* <jeremy@linux-phreak.net> – lfs.linux-phreak.net mirror

South American Mirrors

- *Andres Meggiotto* <sysop@mesi.com.ar> – lfs.mesi.com.ar mirror
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.info mirror
- *Eduardo B. Fonseca* <ebf@aedsolucoes.com.br> – br.linuxfromscratch.org mirror

European Mirrors

- *Barna Koczka* <barna@siker.hu> – hu.linuxfromscratch.org mirror
- *UK Mirror Service* – linuxfromscratch.mirror.ac.uk mirror
- *Martin Voss* <Martin.Voss@ada.de> – lfs.linux-matrix.net mirror
- *Guido Passet* <guido@primerelay.net> – nl.linuxfromscratch.org mirror
- *Bastiaan Jacques* <baafie@planet.nl> – lfs.pagefault.net mirror
- *Roel Neefs* <lfs-mirror@linuxfromscratch.rave.org> – linuxfromscratch.rave.org mirror
- *Justin Knierim* <justin@jrknierim.de> – www.lfs-matrix.de mirror
- *Stephan Brendel* <stevie@stevie20.de> – lfs.netservice-neuss.de mirror
- *Antonin Sprinzl* <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org mirror
- *Fredrik Danerklint* <fredan-lfs@fredan.org> – se.linuxfromscratch.org mirror
- *Parisian sysadmins* <archive@doc.cs.univ-paris8.fr> – www2.fr.linuxfromscratch.org mirror
- *Alexander Velin* <velin@zadnik.org> – bg.linuxfromscratch.org mirror
- *Dirk Webster* <dirk@securewebservices.co.uk> – lfs.securewebservices.co.uk mirror
- *Thomas Skyt* <thomas@sofagang.dk> – dk.linuxfromscratch.org mirror
- *Simon Nicoll* <sime@dot-sime.com> – uk.linuxfromscratch.org mirror

Asian Mirrors

- *Pui Yong* <pyng@spam.averse.net> – sg.linuxfromscratch.org mirror
- *Stuart Harris* <stuart@althalus.me.uk> – lfs.mirror.intermedia.com.sg mirror

Australian Mirrors

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

Former Project Team Members

- *Christine Barczak* <theladyskye@linuxfromscratch.org> – LFS Book Editor
- Timothy Bauscher
- Robert Briggs
- Ian Chilton
- *Jeroen Coumans* <jeroen@linuxfromscratch.org> – Website Developer, FAQ Maintainer
- Alex Groenewoud – LFS Technical Writer
- Marc Heerdink
- Mark Hymers
- Seth W. Klein – FAQ maintainer
- *Nicholas Leippe* <nicholas@linuxfromscratch.org> – Wiki Maintainer
- Simon Perreault
- *Scot Mc Pherson* <scot@linuxfromscratch.org> – LFS NNTP Gateway Maintainer
- *Alexander Patrakov* <semzx@newmail.ru> – LFS Technical Writer
- *Greg Schafer* <gschafer@zip.com.au> – LFS Technical Writer
- Jesse Tie-Ten-Quee – LFS Technical Writer
- *Jeremy Utley* <jeremy@linuxfromscratch.org> – LFS Technical Writer, Bugzilla Maintainer, LFS-Bootscripts Maintainer
- *Zack Winkles* <zwinkles@gmail.com> – LFS Technical Writer

A very special thank you to our donators

- *Dean Benson* <dean@vipersoft.co.uk> for several monetary contributions
- *Hagen Herrschaft* <hrx@hrxnet.de> for donating a 2.2 GHz P4 system, now running under the name of Lorien
- *VA Software* who, on behalf of *Linux.com*, donated a VA Linux 420 (former StartX SP2) workstation
- Mark Stone for donating Belgarath, the linuxfromscratch.org server

Appendix C. Dependencies

Every package built in CLFS relies on one or more other packages in order to build and install properly. Some packages even participate in circular dependencies, that is, the first package depends on the second which in turn depends on the first. Because of these dependencies, the order in which packages are built in CLFS is very important. The purpose of this page is to document the dependencies of each package built in CLFS.

For each package we build, we have listed three types of dependencies. The first lists what other packages need to be available in order to compile and install the package in question. The second lists what packages, in addition to those on the first list, need to be available in order to run the testsuites. The last list of dependencies are packages that require this package to be built and installed in its final location before they are built and installed. In most cases, this is because these packages will hardcode paths to binaries within their scripts. If not built in a certain order, this could result in paths of /tools/bin/[binary] being placed inside scripts installed to the final system. This is obviously not desirable.

Autoconf

Installation depends on: Bash, Coreutils, Grep, M4, Make, Perl, Sed and Texinfo

Test suite depends on: Automake, Diffutils, Findutils, GCC and Libtool

Must be installed before: Automake

Automake

Installation depends on: Autoconf, Bash, Coreutils, Gettext, Grep, M4, Make, Perl, Sed and Texinfo

Test suite depends on: Binutils, Bison, Bzip2, DejaGNU, Diffutils, Expect, Findutils, Flex, GCC, Gettext, Gzip, Libtool and Tar. Can also use several other packages that are not installed in CLFS.

Must be installed before: None

Bash

Installation depends on: Bash, Bison, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Readline, Sed and Texinfo

Test suite depends on: Diffutils and Gawk

Must be installed before: None

Binutils

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

Test suite depends on: DejaGNU and Expect

Must be installed before: None

Bison

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make and Sed

Test suite depends on: Diffutils and Findutils

Must be installed before: Flex, Kbd and Tar

Bzip2

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make and Patch

Test suite depends on: None

Must be installed before: None

CLFS-Bootscripts

Installation depends on: Bash, Coreutils, Make and Sed

Test suite depends on: None

Must be installed before: None

Coreutils

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

Test suite depends on: Diffutils

Must be installed before: Bash, Diffutils, Findutils, Man-DB and Udev

DejaGNU

Installation depends on: Bash, Coreutils, Diffutils, GCC, Grep, Make and Sed

Test suite depends on: None

Must be installed before: None

Diffutils

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

Test suite depends on: No testsuite available

Must be installed before: None

Expect

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

Test suite depends on: None

Must be installed before: None

E2fsprogs

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Gettext, Glibc, Grep, Gzip, Make, Sed and Texinfo

Test suite depends on: Diffutils

Must be installed before: Util-Linux

File

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed and Zlib

Test suite depends on: No testsuite available

Must be installed before: None

Findutils

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

Test suite depends on: DejaGNU, Diffutils and Expect

Must be installed before: None

Flex

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Patch, Sed and Texinfo

Test suite depends on: Bison and Gawk

Must be installed before: IPRoute2, Kbd and Man-DB

Gawk

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed and Texinfo

Test suite depends on: Diffutils

Must be installed before: None

Gcc

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

Test suite depends on: DejaGNU and Expect

Must be installed before: None

Gettext

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed and Texinfo

Test suite depends on: Diffutils, Perl and Tcl

Must be installed before: Automake

Glibc

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

Test suite depends on: None

Must be installed before: None

Grep

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Patch, Sed and Texinfo

Test suite depends on: Diffutils and Gawk

Must be installed before: Man-DB

Groff

Installation depends on: Bash, Binutils, Bison, Coreutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed and Texinfo

Test suite depends on: No testsuite available

Must be installed before: Man and Perl

Gzip

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

Test suite depends on: No testsuite available

Must be installed before: Man-DB

Iana-Etc

Installation depends on: Coreutils, Gawk and Make

Test suite depends on: No testsuite available

Must be installed before: Perl

Inetutils

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed and Texinfo

Test suite depends on: No testsuite available

Must be installed before: Tar

IProute2

Installation depends on: Bash, Bison, Coreutils, Flex, GCC, Glibc, Make and Linux-Headers

Test suite depends on: No testsuite available

Must be installed before: None

Kbd

Installation depends on: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch and Sed

Test suite depends on: No testsuite available

Must be installed before: None

Less

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

Test suite depends on: No testsuite available

Must be installed before: None

Libtool

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

Test suite depends on: Findutils

Must be installed before: None

Linux-Headers

Installation depends on: Coreutils and Findutils

Test suite depends on: No testsuite available

Must be installed before: None

Linux Kernel

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Mktemp, Module-Init-Tools, Ncurses and Sed

Test suite depends on: No testsuite available

Must be installed before: None

M4

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make and Sed

Test suite depends on: Diffutils

Must be installed before: Autoconf and Bison

Man-DB

Installation depends on: Bash, Binutils, Bzip2, Coreutils, Flex, GCC, Gettext, Glibc, Grep, Groff, Gzip, Less, Make and Sed

Test suite depends on: No testsuite available

Must be installed before: None

Make

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

Test suite depends on: Perl

Must be installed before: None

Mktemp

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Patch and Sed

Test suite depends on: No testsuite available

Must be installed before: None

Module-Init-Tools

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed and Zlib

Test suite depends on: File, Findutils and Gawk

Must be installed before: None

Ncurses

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

Test suite depends on: No testsuite available

Must be installed before: Bash, GRUB, Inetutils, Less, Procps, Psmisc, Readline, Texinfo, Util-Linux and Vim

Patch

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make and Sed

Test suite depends on: No testsuite available

Must be installed before: None

Perl

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Groff, Make and Sed

Test suite depends on: Iana-Etc and Procps

Must be installed before: Autoconf

Procps

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make and Ncurses

Test suite depends on: No testsuite available

Must be installed before: None

Psmisc

Installation depends on: Bash, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses and Sed

Test suite depends on: No testsuite available

Must be installed before: None

Readline

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed and Texinfo

Test suite depends on: No testsuite available

Must be installed before: Bash

Sed

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

Test suite depends on: Diffutils and Gawk

Must be installed before: E2fsprogs, File, Libtool and Shadow

Shadow

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

Test suite depends on: No testsuite available

Must be installed before: None

Sysklogd

Installation depends on: Binutils, Coreutils, GCC, Glibc, Make and Patch

Test suite depends on: No testsuite available

Must be installed before: None

Sysvinit

Installation depends on: Binutils, Coreutils, GCC, Glibc, Make and Sed

Test suite depends on: No testsuite available

Must be installed before: None

Tar

Installation depends on: Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Patch, Sed and Texinfo

Test suite depends on: Diffutils, Findutils and Gawk

Must be installed before: None

Tcl

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

Test suite depends on: None

Must be installed before: None

Texinfo

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch and Sed

Test suite depends on: None

Must be installed before: None

Tree

Installation depends on: Coreutils, GCC and Make

Test suite depends on: None

Must be installed before: None

Udev

Installation depends on: Binutils, Coreutils, GCC, Glibc and Make

Test suite depends on: Findutils, Perl and Sed

Must be installed before: None

Udev Rules

Installation depends on: Bash, Coreutils, Make and Sed

Test suite depends on: None

Must be installed before: None

Util-Linux

Installation depends on: Bash, Binutils, Coreutils, E2fprogs, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch, Sed and Zlib

Test suite depends on: No testsuite available

Must be installed before: None

Vim

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

Test suite depends on: None

Must be installed before: None

Zlib

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make and Sed

Test suite depends on: None

Must be installed before: File, Module-Init-Tools and Util-Linux

Appendix D. x86 Dependencies

This page contains dependency information for packages specific to x86 Pure64.

Bin86

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make and Patch

Test suite depends on: None

Must be installed before: Lilo

Lilo

Installation depends on: Bash, Bin86, Binutils, Coreutils, GCC, Glibc, Make and Sed

Test suite depends on: None

Must be installed before: None

Index

Packages

Programs

Libraries

Scripts

Others