Linux From Scratch

Version 12.1

Published March 1st, 2024

Created by Gerard Beekmans
Managing Editor: Bruce Dubbs

Linux From Scratch: Version 12.1: Published March 1st, 2024

by Created by Gerard Beekmans and Managing Editor: Bruce Dubbs

Table of Contents

Preface

Foreword

My journey to learn and better understand Linux began back in 1998. I had just installed my first Linux distribution and had quickly become intrigued with the whole concept and philosophy behind Linux.

There are always many ways to accomplish a single task. The same can be said about Linux distributions. A great many have existed over the years. Some still exist, some have morphed into something else, yet others have been relegated to our memories. They all do things differently to suit the needs of their target audience. Because so many different ways to accomplish the same end goal exist, I began to realize I no longer had to be limited by any one implementation. Prior to discovering Linux, we simply put up with issues in other Operating Systems as you had no choice. It was what it was, whether you liked it or not. With Linux, the concept of choice began to emerge. If you didn't like something, you were free, even encouraged, to change it.

I tried a number of distributions and could not decide on any one. They were great systems in their own right. It wasn't a matter of right and wrong anymore. It had become a matter of personal taste. With all that choice available, it became apparent that there would not be a single system that would be perfect for me. So I set out to create my own Linux system that would fully conform to my personal preferences.

To truly make it my own system, I resolved to compile everything from source code instead of using pre-compiled binary packages. This “perfect” Linux system would have the strengths of various systems without their perceived weaknesses. At first, the idea was rather daunting. I remained committed to the idea that such a system could be built.

After sorting through issues such as circular dependencies and compile-time errors, I finally built a custom-built Linux system. It was fully operational and perfectly usable like any of the other Linux systems out there at the time. But it was my own creation. It was very satisfying to have put together such a system myself. The only thing better would have been to create each piece of software myself. This was the next best thing.

As I shared my goals and experiences with other members of the Linux community, it became apparent that there was a sustained interest in these ideas. It quickly became plain that such custom-built Linux systems serve not only to meet user specific requirements, but also serve as an ideal learning opportunity for programmers and system administrators to enhance their (existing) Linux skills. Out of this broadened interest, the *Linux From Scratch Project* was born.

This Linux From Scratch book is the central core around that project. It provides the background and instructions necessary for you to design and build your own system. While this book provides a template that will result in a correctly working system, you are free to alter the instructions to suit yourself, which is, in part, an important part of this project. You remain in control; we just lend a helping hand to get you started on your own journey.

I sincerely hope you will have a great time working on your own Linux From Scratch system and enjoy the numerous benefits of having a system that is truly your own.

--
Gerard Beekmans
gerard@linuxfromscratch.org

Audience

There are many reasons why you would want to read this book. One of the questions 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?”

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

Another key benefit of LFS is that it gives you control of the system without relying on someone else's Linux implementation. With LFS, you are in the driver's seat. *You* dictate every aspect of your system.

LFS allows you to create very compact Linux systems. With other distributions you are often forced to install a great many programs you neither use nor understand. These programs waste resources. You may argue that with today's hard drives and CPUs, wasted resources are no longer a consideration. Sometimes, however, you are still constrained by the system's size, if nothing else. Think about bootable CDs, USB sticks, and embedded systems. Those are areas where LFS can be beneficial.

Another advantage of a custom built Linux system is security. By compiling the entire system from source code, you are empowered to audit everything and apply all the security patches you want. You don't have to wait for somebody else to compile binary packages that fix a security hole. Unless you examine the patch and implement it yourself, you have no guarantee that the new binary package was built correctly and adequately fixes the problem.

The goal of Linux From Scratch is to build a complete and usable foundation-level system. If you do not wish to build your own Linux system from scratch, you may nevertheless benefit from the information in this book.

There are too many good reasons to build your own LFS system to list them all here. In the end, education is by far the most important reason. As you continue your LFS experience, you will discover the power that information and knowledge can bring.

LFS Target Architectures

The primary target architectures of LFS are the AMD/Intel x86 (32-bit) and x86_64 (64-bit) CPUs. On the other hand, the instructions in this book are also known to work, with some modifications, with the Power PC and ARM CPUs. To build a system that utilizes one of these alternative CPUs, the main prerequisite, in addition to those on the next page, is an existing Linux system such as an earlier LFS installation, Ubuntu, Red Hat/Fedora, SuSE, or some other distribution that targets that architecture. (Note that a 32-bit distribution can be installed and used as a host system on a 64-bit AMD/Intel computer.)

The gain from building on a 64-bit system, as compared to a 32-bit system, is minimal. For example, in a test build of LFS-9.1 on a Core i7-4790 CPU based system, using 4 cores, the following statistics were measured:

```
Architecture Build Time     Build Size
32-bit       239.9 minutes  3.6 GB
64-bit       233.2 minutes  4.4 GB
```

As you can see, on the same hardware, the 64-bit build is only 3% faster (and 22% larger) than the 32-bit build. If you plan to use LFS as a LAMP server, or a firewall, a 32-bit CPU may be good enough. On the other hand, several packages in BLFS now need more than 4 GB of RAM to be built and/or to run; if you plan to use LFS as a desktop, the LFS authors recommend building a 64-bit system.

The default 64-bit build that results from LFS is a “pure” 64-bit system. That is, it supports 64-bit executables only. Building a “multi-lib” system requires compiling many applications twice, once for a 32-bit system and once for a 64-bit system. This is not directly supported in LFS because it would interfere with the educational objective of providing the minimal instructions needed for a basic Linux system. Some of the LFS/BLFS editors maintain a multilib fork of LFS, accessible at *https://www.linuxfromscratch.org/~thomas/multilib/index.html*. But that's an advanced topic.

Prerequisites

Building an LFS 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, you should already know how 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 you know how to use and install Linux software.

Because the LFS book assumes *at least* this basic level of skill, the various LFS support forums are unlikely to provide you with much assistance in these areas. You will find that your questions regarding such basic knowledge will likely go unanswered (or you will simply be referred to the LFS essential pre-reading list).

Before building an LFS system, we urge you to read these articles:

- Software-Building-HOWTO *https://tldp.org/HOWTO/Software-Building-HOWTO.html*

  This is a comprehensive guide to building and installing “generic” Unix software packages under Linux. Although it was written some time ago, it still provides a good summary of the basic techniques used to build and install software.
- Beginner's Guide to Installing from Source *https://moi.vonos.net/linux/beginners-installing-from-source/*

  This guide provides a good summary of the basic skills and techniques needed to build software from source code.

LFS and Standards

The structure of LFS follows Linux standards as closely as possible. The primary standards are:

- *POSIX.1-2008*.
- *Filesystem Hierarchy Standard (FHS) Version 3.0*
- *Linux Standard Base (LSB) Version 5.0 (2015)*

  The LSB has four separate specifications: Core, Desktop, Runtime Languages, and Imaging. Some parts of Core and Desktop specifications are architecture specific. There are also two trial specifications: Gtk3 and Graphics. LFS attempts to conform to the LSB specifications for the IA32 (32-bit x86) or AMD64 (x86_64) architectures discussed in the previous section.

Note

Many people do not agree with these requirements. The main purpose of the LSB is to ensure that proprietary software can be installed and run on a compliant system. Since LFS is source based, the user has complete control over what packages are desired; you may choose not to install some packages that are specified by the LSB.

While it is possible to create a complete system that will pass the LSB certification tests “from scratch,” this can't be done without many additional packages that are beyond the scope of the LFS book. Installation instructions for these additional packages can be found in BLFS.

Packages supplied by LFS needed to satisfy the LSB Requirements

LSB Core:	Bash, Bc, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux, Zlib

LSB Desktop:	None
LSB Runtime Languages:	Perl, Python
LSB Imaging:	None
LSB Gtk3 and LSB Graphics (Trial Use):	None

Packages supplied by BLFS needed to satisfy the LSB Requirements

LSB Core:	At, Batch (a part of At), Cpio, Ed, Fcrontab, LSB-Tools, NSPR, NSS, PAM, Pax, Sendmail (or Postfix or Exim), time
LSB Desktop:	Alsa, ATK, Cairo, Desktop-file-utils, Freetype, Fontconfig, Gdk-pixbuf, Glib2, GTK+2, Icon-naming-utils, Libjpeg-turbo, Libpng, Libtiff, Libxml2, MesaLib, Pango, Xdg-utils, Xorg
LSB Runtime Languages:	Libxml2, Libxslt
LSB Imaging:	CUPS, Cups-filters, Ghostscript, SANE
LSB Gtk3 and LSB Graphics (Trial Use):	GTK+3

Packages not supplied by LFS or BLFS needed to satisfy the LSB Requirements

LSB Core:	None
LSB Desktop:	Qt4 (but Qt5 is provided)
LSB Runtime Languages:	None
LSB Imaging:	None
LSB Gtk3 and LSB Graphics (Trial Use):	None

Rationale for Packages in the Book

The goal of LFS is to build a complete and usable foundation-level system—including all the packages needed to replicate itself—and providing a relatively minimal base from which to customize a more complete system based on the user's choices. This does not mean that LFS is the smallest system possible. Several important packages are included that are not, strictly speaking, required. The list below documents the reasons each package in the book has been included.

- Acl

  This package contains utilities to administer Access Control Lists, which are used to define fine-grained discretionary access rights for files and directories.

- Attr

  This package contains programs for managing extended attributes on file system objects.

- Autoconf

  This package supplies programs for producing shell scripts that can automatically configure source code from a developer's template. It is often needed to rebuild a package after the build procedure has been updated.

- Automake

  This package contains programs for generating Make files from a template. It is often needed to rebuild a package after the build procedure has been updated.

- Bash

This package satisfies an LSB core requirement to provide a Bourne Shell interface to the system. It was chosen over other shell packages because of its common usage and extensive capabilities.

- Bc

  This package provides an arbitrary precision numeric processing language. It satisfies a requirement for building the Linux kernel.

- Binutils

  This package supplies a linker, an assembler, and other tools for handling object files. The programs in this package are needed to compile most of the packages in an LFS system.

- Bison

  This package contains the GNU version of yacc (Yet Another Compiler Compiler) needed to build several of the LFS programs.

- Bzip2

  This package contains programs for compressing and decompressing files. It is required to decompress many LFS packages.

- Check

  This package provides a test harness for other programs.

- Coreutils

  This package contains a number of essential programs for viewing and manipulating files and directories. These programs are needed for command line file management, and are necessary for the installation procedures of every package in LFS.

- DejaGNU

  This package supplies a framework for testing other programs.

- Diffutils

  This package contains programs that show the differences between files or directories. These programs can be used to create patches, and are also used in many packages' build procedures.

- E2fsprogs

  This package supplies utilities for handling the ext2, ext3 and ext4 file systems. These are the most common and thoroughly tested file systems that Linux supports.

- Expat

  This package yields a relatively small XML parsing library. It is required by the XML::Parser Perl module.

- Expect

  This package contains a program for carrying out scripted dialogues with other interactive programs. It is commonly used for testing other packages.

- File

  This package contains a utility for determining the type of a given file or files. A few packages need it in their build scripts.

- Findutils

This package provides programs to find files in a file system. It is used in many packages' build scripts.

- Flex

  This package contains a utility for generating programs that recognize patterns in text. It is the GNU version of the lex (lexical analyzer) program. It is required to build several LFS packages.

- Gawk

  This package supplies programs for manipulating text files. It is the GNU version of awk (Aho-Weinberg-Kernighan). It is used in many other packages' build scripts.

- GCC

  This is the Gnu Compiler Collection. It contains the C and C++ compilers as well as several others not built by LFS.

- GDBM

  This package contains the GNU Database Manager library. It is used by one other LFS package, Man-DB.

- Gettext

  This package provides utilities and libraries for the internationalization and localization of many packages.

- Glibc

  This package contains the main C library. Linux programs will not run without it.

- GMP

  This package supplies math libraries that provide useful functions for arbitrary precision arithmetic. It is needed to build GCC.

- Gperf

  This package produces a program that generates a perfect hash function from a set of keys. It is required by Udev .

- Grep

  This package contains programs for searching through files. These programs are used by most packages' build scripts.

- Groff

  This package contributes programs for processing and formatting text. One important function of these programs is to format man pages.

- GRUB

  This is the Grand Unified Boot Loader. It is the most flexible of several boot loaders available.

- Gzip

  This package contains programs for compressing and decompressing files. It is needed to decompress many packages in LFS.

- Iana-etc

  This package provides data for network services and protocols. It is needed to enable proper networking capabilities.

- Inetutils

This package supplies programs for basic network administration.

- Intltool

  This package contributes tools for extracting translatable strings from source files.

- IProute2

  This package contains programs for basic and advanced IPv4 and IPv6 networking. It was chosen over the other common network tools package (net-tools) for its IPv6 capabilities.

- Kbd

  This package produces key-table files, keyboard utilities for non-US keyboards, and a number of console fonts.

- Kmod

  This package supplies programs needed to administer Linux kernel modules.

- Less

  This package contains a very nice text file viewer that allows scrolling up or down when viewing a file. Many packages use it for paging the output.

- Libcap

  This package implements the userspace interfaces to the POSIX 1003.1e capabilities available in Linux kernels.

- Libelf

  The elfutils project provides libraries and tools for ELF files and DWARF data. Most utilities in this package are available in other packages, but the library is needed to build the Linux kernel using the default (and most efficient) configuration.

- Libffi

  This package implements a portable, high level programming interface to various calling conventions. Some programs may not know at the time of compilation what arguments are to be passed to a function. For instance, an interpreter may be told at run-time about the number and types of arguments used to call a given function. Libffi can be used in such programs to provide a bridge from the interpreter program to compiled code.

- Libpipeline

  The Libpipeline package supplies a library for manipulating pipelines of subprocesses in a flexible and convenient way. It is required by the Man-DB package.

- Libtool

  This package contains the GNU generic library support script. It wraps the complexity of using shared libraries into a consistent, portable interface. It is needed by the test suites in other LFS packages.

- Libxcrypt

  This package provides the `libcrypt` library needed by various packages (notably, Shadow) for hashing passwords. It replaces the obsolete `libcrypt` implementation in Glibc.

- Linux Kernel

  This package is the Operating System. It is the Linux in the GNU/Linux environment.

- M4

  This package provides a general text macro processor useful as a build tool for other programs.

- Make

  This package contains a program for directing the building of packages. It is required by almost every package in LFS.

- Man-DB

  This package contains programs for finding and viewing man pages. It was chosen instead of the man package because of its superior internationalization capabilities. It supplies the man program.

- Man-pages

  This package provides the actual contents of the basic Linux man pages.

- Meson

  This package provides a software tool for automating the building of software. The main goal of Meson is to minimize the amount of time that software developers need to spend configuring a build system. It's required to build Systemd, as well as many BLFS packages.

- MPC

  This package supplies arithmetic functions for complex numbers. It is required by GCC.

- MPFR

  This package contains functions for multiple precision arithmetic. It is required by GCC.

- Ninja

  This package furnishes a small build system with a focus on speed. It is designed to have its input files generated by a higher-level build system, and to run builds as fast as possible. This package is required by Meson.

- Ncurses

  This package contains libraries for terminal-independent handling of character screens. It is often used to provide cursor control for a menuing system. It is needed by a number of the packages in LFS.

- Openssl

  This package provides management tools and libraries relating to cryptography. These supply cryptographic functions to other packages, including the Linux kernel.

- Patch

  This package contains a program for modifying or creating files by applying a *patch* file typically created by the diff program. It is needed by the build procedure for several LFS packages.

- Perl

  This package is an interpreter for the runtime language PERL. It is needed for the installation and test suites of several LFS packages.

- Pkgconf

  This package contains a program which helps to configure compiler and linker flags for development libraries. The program can be used as a drop-in replacement of **pkg-config**, which is needed by the building system of many packages. It's maintained more actively and slightly faster than the original Pkg-config package.

- Procps-NG

  This package contains programs for monitoring processes. These programs are useful for system administration, and are also used by the LFS Bootscripts.

- Psmisc

  This package produces programs for displaying information about running processes. These programs are useful for system administration.

- Python 3

  This package provides an interpreted language that has a design philosophy emphasizing code readability.

- Readline

  This package is a set of libraries that offer command-line editing and history capabilities. It is used by Bash.

- Sed

  This package allows editing of text without opening it in a text editor. It is also needed by many LFS packages' configure scripts.

- Shadow

  This package contains programs for handling passwords securely.

- Sysklogd

  This package supplies programs for logging system messages, such as those emitted by the kernel or daemon processes when unusual events occur.

- Sysvinit

  This package provides the init program, the parent of all the other processes on a running Linux system.

- Udev

  This package is a device manager. It dynamically controls the ownership, permissions, names, and symbolic links of device nodes in the /dev directory when devices are added to or removed from the system.

- Tar

  This package provides archiving and extraction capabilities of virtually all the packages used in LFS.

- Tcl

  This package contains the Tool Command Language used in many test suites.

- Texinfo

  This package supplies programs for reading, writing, and converting info pages. It is used in the installation procedures of many LFS packages.

- Util-linux

  This package contains miscellaneous utility programs. Among them are utilities for handling file systems, consoles, partitions, and messages.

- Vim

  This package provides an editor. It was chosen because of its compatibility with the classic vi editor and its huge number of powerful capabilities. An editor is a very personal choice for many users. Any other editor can be substituted, if you wish.

- Wheel

  This package supplies a Python module that is the reference implementation of the Python wheel packaging standard.

- XML::Parser

  This package is a Perl module that interfaces with Expat.

- XZ Utils

  This package contains programs for compressing and decompressing files. It provides the highest compression generally available and is useful for decompressing packages in XZ or LZMA format.

- Zlib

  This package contains compression and decompression routines used by some programs.

- Zstd

  This package supplies compression and decompression routines used by some programs. It provides high compression ratios and a very wide range of compression / speed trade-offs.

Typography

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

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

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

In some cases, a logical line is extended to two or more physical lines with a backslash at the end of the line.

```
CC="gcc -B/usr/bin/" ../binutils-2.18/configure \
  --prefix=/tools --disable-nls --disable-werror
```

Note that the backslash must be followed by an immediate return. Other whitespace characters like spaces or tab characters will create incorrect results.

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

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

Note

Please configure your browser to display fixed-width text with a good monospace" font-size="9ptd font, with which you can distinguish the glyphs of `Il1` or `O0` clearly.

Emphasis

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

https://www.linuxfromscratch.org/

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

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

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

`<REPLACED TEXT>`

This format is used to encapsulate text that is not to be typed as seen or for copy-and-paste operations.

`[OPTIONAL TEXT]`

This format is used to encapsulate text that is optional.

passwd(5)

This format is used to refer to a specific manual (man) page. The number inside parentheses indicates a specific section inside the manuals. For example, **passwd** has two man pages. Per LFS installation instructions, those two man pages will be located at `/usr/share/man/man1/passwd.1` and `/usr/share/man/man5/passwd.5`. 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 page being specified. Note that most man pages do not have duplicate page names in different sections. Therefore, **man *`<program name>`*** is generally sufficient. In the LFS book these references to man pages are also hyperlinks, so clicking on such a reference will open the man page rendered in HTML from *Arch Linux manual pages*.

Structure

This book is divided into the following parts.

Part I - Introduction

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

Part II - Preparing for the Build

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

Part III - Building the LFS Cross Toolchain and Temporary Tools

Part III provides instructions for building the tools needed for constructing the final LFS system.

Part IV - Building the LFS System

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

Part V - Appendices

Part V provides information about the book itself including acronyms and terms, acknowledgments, package dependencies, a listing of LFS boot scripts, licenses for the distribution of the book, and a comprehensive index of packages, programs, libraries, and scripts.

Errata and Security Advisories

The software used to create an LFS system is constantly being updated and enhanced. Security warnings and bug fixes may become available after the LFS book has been released. To check whether the package versions or instructions in this release of LFS need any modifications—to repair security vulnerabilities or to fix other bugs—please visit *https://www.linuxfromscratch.org/lfs/errata/12.1/* before proceeding with your build. You should note any changes shown and apply them to the relevant sections of the book as you build the LFS system.

In addition, the Linux From Scratch editors maintain a list of security vulnerabilities discovered *after* a book has been released. To read the list, please visit *https://www.linuxfromscratch.org/lfs/advisories/* before proceeding with your build. You should apply the changes suggested by the advisories to the relevant sections of the book as you build the LFS system. And, if you will use the LFS system as a real desktop or server system, you should continue to consult the advisories and fix any security vulnerabilities, even when the LFS system has been completely constructed.

Part I. Introduction

Chapter 1. Introduction

1.1. How to Build an LFS System

The LFS system will be built by using an already installed Linux distribution (such as Debian, OpenMandriva, Fedora, or openSUSE). 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 include these tools.

Note

There are many ways to install a Linux distribution and the defaults are usually not optimal for building an LFS system. For suggestions on setting up a commercial distribution see: *https://www.linuxfromscratch.org/hints/downloads/files/partitioning-for-lfs.txt*.

As an alternative to installing a separate distribution on your machine, you may wish to use a LiveCD from a commercial distribution.

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

Chapter 5 explains the installation of the initial tool chain, (binutils, gcc, and glibc) using cross-compilation techniques to isolate the new tools from the host system.

Chapter 6 shows you how to cross-compile basic utilities using the just built cross-toolchain.

Chapter 7 then enters a "chroot" environment, where we use the new tools to build all the rest of the tools needed to create the LFS system.

This effort to isolate the new system from the host distribution may seem excessive. A full technical explanation as to why this is done is provided in Toolchain Technical Notes.

In Chapter 8 the full-blown LFS system is built. Another advantage provided by the chroot environment is that it allows you to continue using the host system while LFS is being built. While waiting for package compilations to complete, you can continue using your computer as usual.

To finish the installation, the basic system configuration is set up in Chapter 9, and the kernel and boot loader are created in Chapter 10. Chapter 11 contains information on continuing the LFS experience beyond this book. After the steps in this chapter have been implemented, the computer is ready to boot into the new LFS system.

This is the process in a nutshell. Detailed information on each step is presented in the following chapters. Items that seem complicated now will be clarified, and everything will fall into place as you commence your LFS adventure.

1.2. What's new since the last release

Here is a list of the packages updated since the previous release of LFS.

Upgraded to:

•

- Acl-2.3.2
- Attr-2.5.2
- Autoconf-2.72
- Bash-5.2.21
- Bc-6.7.5
- Binutils-2.42
- Coreutils-9.4
- Expat-2.6.0
- Gawk-5.3.0
- Gettext-0.22.4
- Glibc-2.39
- GRUB-2.12
- Gzip-1.13
- Iana-Etc-20240125
- Inetutils-2.5
- IPRoute2-6.7.0
- Jinja2-3.1.3
- Kbd-2.6.4
- Kmod-31
- Libelf from Elfutils-0.190
- Linux-6.7.4
- Man-DB-2.12.0
- Man-pages-6.06
- MarkupSafe-2.1.5
- Meson-1.3.2
- MPFR-4.2.1
- Ncurses-6.4-20230520
- OpenSSL-3.2.1
- Perl-5.38.2
- Pkgconf-2.1.1
- Procps-ng-4.0.4
- Python-3.12.2
- Setuptools-69.1.0

- Shadow-4.14.5
- SysVinit-3.08
- Texinfo-7.1
- Tzdata-2024a
- Util-linux-2.39.3
- Vim-9.1.0041
- Wheel-0.42.0
- XML::Parser-2.47
- Xz-5.4.6
- Zlib-1.3.1

Added:

-
- bash-5.2.21-upstream_fixes-1.patch
- readline-8.2-upstream_fixes-3.patch
- setuptools-69.1.0
- systemd-255-upstream_fixes-1.patch

Removed:

-
- glibc-2.38-memalign_fix-1.patch
- grub-2.06-upstream_fixes-1.patch
- readline-8.2-upstream_fix-1.patch

1.3. Changelog

This is version 12.1 of the Linux From Scratch book, dated March 1st, 2024. 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 *https://www.linuxfromscratch.org/mirrors.html*.

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

Changelog Entries:

- 2024-03-01
  - [bdubbs] - LFS-12.1 released.
- 2024-02-14
  - [bdubbs] - Update to meson-1.3.2. Fixes *#5442*.
- 2024-02-12

    - [bdubbs] - Update to shadow-4.14.5. Fixes *#5437*.
    - [bdubbs] - Update to setuptools-69.1.0 (Python module). Fixes *#5439*.
    - [bdubbs] - Update to python-3.12.2. Fixes *#5434*.
    - [bdubbs] - Update to pkgconf-2.1.1. Fixes *#5432*.
    - [bdubbs] - Update to MarkupSafe-2.1.5 (Python module). Fixes *#5431*.
    - [bdubbs] - Update to man-pages-6.06. Fixes *#5438*.
    - [bdubbs] - Update to expat-2.6.0. Fixes *#5435*.
    - [bdubbs] - Update to linux-6.7.4. Fixes *#5433*.
- 2024-02-02
    - [xry111] - Update to tzdata-2024a. Fixes *#5428*.
    - [xry111] - Update to glibc-2.39 (security fix). Fixes *#5426*.
    - [xry111] - Update to linux-6.7.3. Fixes *#5427*.
- 2024-02-01
    - [bdubbs] - Update to openssl-3.2.1 (security fix). Fixes *#5425*.
    - [bdubbs] - Update to zlib-1.3.1. Fixes *#5419*.
    - [bdubbs] - Update to xz-5.4.6. Fixes *#5423*.
    - [bdubbs] - Update to linux-6.7.2. Fixes *#5422*.
    - [bdubbs] - Update to iana-etc-20240125. Addresses *#5006*.
    - [bdubbs] - Update to binutils-2.42. Fixes *#5424*.
    - [bdubbs] - Update to acl-2.3.2. Fixes *#5421*.
    - [bdubbs] - Update upstream fixes for readline-8.2. Fixes *#5420*.
    - [bdubbs] - Apply upstream fix for bash-5.2.21. Fixes *#5420*.
- 2024-01-21
    - [xry111] - Apply upstream fix for pkgconf-2.1.0 regression. Fixes *#5414*.
    - [xry111] - Update to jinja2-3.1.3 (security fix). Fixes *#5411*.
    - [xry111] - Update to bc-6.7.5. Fixes *#5408*.
    - [xry111] - Update to attr-2.5.2. Fixes *#5412*.
    - [xry111] - Update to ncurses-6.4-20230520 (security fix). Fixes *#5416*.
    - [xry111] - Update to markupsafe-2.1.4. Fixes *#5418*.
    - [xry111] - Update to linux-6.7.1. Fixes *#5406*.
    - [xry111] - Update to iproute2-6.7.0. Fixes *#5410*.
    - [xry111] - Update to vim-9.1.0041. Addresses *#4500*.
    - [xry111] - Update to iana-etc-20240117. Addresses *#5006*.
    - [xry111] - Update to shadow-4.14.3. Fixes *#5413*.
    - [xry111] - Fix CVE-2024-0684 for coreutils-9.4. Fixes *#5417*.
- 2024-01-18

    - [xry111] - Edit a ncurses header to always use the wide-character ABI compatible with libncursesw.so because we are faking the 8-bit libncurses.so with it. Fixes *#5415*.
- 2024-01-09
    - [renodr] - Fix the definition of the C.UTF-8 locale. Fixes *#5409*.
- 2023-12-31
    - [xry111] - Add --enable-default-hash-style=gnu configuring binutils. Fixes *#5401*.
    - [xry111] - Fix CVE-2023-7008 for systemd-255. Fixes *#5405*.
    - [xry111] - Update to iana-etc-20231205. Addresses *#5006*.
    - [xry111] - Update to tzdata-2023d. Fixes *#5399*.
    - [xry111] - Update to linux-6.6.8. Fixes *#5397*.
    - [xry111] - Update to meson-1.3.1. Fixes *#5402*.
    - [xry111] - Update to grub-2.12. Fixes *#5396*.
    - [xry111] - Update to inetutils-2.5. Fixes *#5404*.
    - [xry111] - Update to setuptools-69.0.3. Fixes *#5400*.
    - [xry111] - Update to xml-parser-2.47. Fixes *#5403*.
    - [xry111] - Update to vim-9.0.2189. Addresses *#4500*.
    - [xry111] - Update to autoconf-2.72. Fixes *#5398*.
- 2023-12-16
    - [xry111] - Update to udev from systemd-255. Fixes *#5390*.
- 2023-12-14
    - [bdubbs] - Update to util-linux v2.39.3. Fixes *#5388*.
    - [bdubbs] - Update to python3-3.12.1. Fixes *#5392*.
    - [bdubbs] - Update to linux-6.6.7. Fixes *#5387*.
    - [bdubbs] - Update to kbd-2.6.4. Fixes *#5393*.
    - [bdubbs] - Update to bc-6.7.4. Fixes *#5389*.
    - [bdubbs] - Reformat util-linux configure parameters. Fixes *#5395*.
- 2023-12-04
    - [thomas] - Modify commands for install Python docs to avoid too restrictive permissions on the files and dirs.
- 2023-12-01
    - [xry111] - Restore NIC naming based on physical system characteristics. Fixes *#5386*.
- 2023-11-30
    - [bdubbs] - Update to vim-9.0.2136. Addresses *#4500*.
    - [bdubbs] - Update to iana-etc-20231117. Addresses *#5006*.
    - [bdubbs] - Update to bc-6.7.3. Fixes *#5385*.
    - [bdubbs] - Update to wheel-0.42.0 (Python Module). Fixes *#5384*.
    - [bdubbs] - Update to perl-5.38.2. Fixes *#5383*.

- [bdubbs] - Update to pkgconf-2.1.0. Fixes *#5382*.
- [bdubbs] - Update to readline patches 002 through 007. Fixes *#5381*.
- [bdubbs] - Update to openssl-3.2.0. Fixes *#5380*.
- [bdubbs] - Update to setuptools-69.0.2. Fixes *#5379*.
- [bdubbs] - Update to linux-6.6.3. Fixes *#5378*.
- [bdubbs] - Update to meson-1.3.0. Fixes *#5377*.
- [bdubbs] - Update to gettext-0.22.4. Fixes *#5376*.

- 2023-11-13
  - [xry111] - Update to elfutils-0.190. Fixes *#5373*.
  - [xry111] - Update to vim-9.0.2103. Addresses *#4500*.
  - [xry111] - Update to linux-6.6.1. Fixes *#5369*.
  - [xry111] - Update to xz-5.4.5. Fixes *#5371*.
  - [xry111] - Update to iana-etc-20231107. Addresses *#5006*.
  - [xry111] - Update to gawk-5.3.0. Fixes *#5372*.
  - [xry111] - Update to bash-5.2.21. Fixes *#5375*.
  - [xry111] - Update to iproute2-6.6.0. Fixes *#5374*.
- 2023-11-01
  - [bdubbs] - Update to iana-etc-20231019. Addresses *#5006*.
  - [bdubbs] - Update to wheel-0.41.3. Fixes *#5370*.
  - [bdubbs] - Update to shadow-4.14.2. Fixes *#5368*.
  - [bdubbs] - Update to openssl-3.1.4. Fixes *#5367*.
  - [bdubbs] - Update to texinfo-7.1. Fixes *#5364*.
  - [bdubbs] - Update to meson-1.2.3. Fixes *#5366*.
  - [bdubbs] - Update to bc-6.7.2. Fixes *#5363*.
  - [bdubbs] - Update to linux-6.5.9. Fixes *#5365*.
  - [bdubbs] - Update to Python-3.12.0. Fixes *#5357*.
  - [bdubbs] - Add setuptools-68.2.2. Fixes *#5358*.
- 2023-10-15
  - [bdubbs] - Update to linux-6.5.7. Fixes *#5362*.
  - [bdubbs] - Update to shadow-4.14.1. Fixes *#5361*.
  - [bdubbs] - Update to gettext-0.22.3. Fixes *#5359*.
- 2023-10-03
  - [xry111] - Update Glibc upstream fixes patch to fix CVE-2023-4911.
- 2023-10-01
  - [bdubbs] - Disable building nscd in glibc. Fixes *#5349*.
  - [bdubbs] - Update to iana-etc-20230929. Addresses *#5006*.

- [bdubbs] - Update to vim-9.0.1968. Addresses *#4500*.
- [bdubbs] - Update to openssl-3.1.3. Fixes *#5350*.
- [bdubbs] - Update to meson-1.2.2. Fixes *#5356*.
- [bdubbs] - Update to man-db-2.12.0. Fixes *#5354*.
- [bdubbs] - Update to linux-6.5.5. Fixes *#5352*.
- [bdubbs] - Update to kmod-31. Fixes *#5355*.
- [bdubbs] - Update to kbd-2.6.3. Fixes *#5361*.
- [bdubbs] - Update to gettext-0.22.2. Fixes *#5348*.
- [bdubbs] - Update to bc-6.7.0. Fixes *#5353*.

- 2023-09-24
  - [xry111] - Update Glibc upstream fixes patch to plug a memory leak introduced by the security fix.
- 2023-09-17
  - [xry111] - Update to linux-6.5.3. Fixes *#5343*.
  - [xry111] - Update to iana-etc-20230912. Addresses *#5006*.
  - [xry111] - Update to iproute2-6.5.0. Fixes *#5342*.
- 2023-09-13
  - [xry111] - Fix CVE-2023-4806 for Glibc-2.38. Fixes *#5347*.
- 2023-09-12
  - [xry111] - Fix CVE-2023-4527 for Glibc-2.38. Fixes *#5346*.
- 2023-09-07
  - [xry111] - Fix an issue in pkgconf-2.0.3 causing BLFS packages fail to build. Fixes *#5341*.
- 2023-09-05
  - [xry111] - Move pkgconf before binutils for binutils building system to detect zstd properly. Fixes *#5340*.
  - [xry111] - Update to linux-6.5.1. Fixes *#5332*.
  - [xry111] - Update to pkgconf-2.0.3. Fixes *#5339*.
  - [xry111] - Update to dbus-1.14.10. Fixes *#5337*.
- 2023-09-04
  - [bdubbs] - Move caution regarding building by mixing different version of LFS to General Compilation Instructions. Fixes *#5338*.
- 2023-09-02
  - [xry111] - Add --no-cache-dir option for pip3 wheel commands. Addresses *BLFS #18466*.
  - [bdubbs] - Update to vim-9.0.1837. Addresses *#4500*.
  - [bdubbs] - Update to zlib-1.3. Fixes *#5324*.
  - [bdubbs] - Update to wheel-0.41.2 (Python Module). Fixes *#5328*.
  - [bdubbs] - Update to util-linux-2.39.2. Fixes *#5322*.
  - [bdubbs] - Update to sysvinit-3.08. Fixes *#5321*.

- [bdubbs] - Update to shadow-4.14.0. Fixes *#5319*.
- [bdubbs] - Update to Python-3.11.5. Fixes *#5330*.
- [bdubbs] - Update to procps-ng-4.0.4 (security fix for 32-bit systems). Fixes *#5335*.
- [bdubbs] - Update to pkgconf-2.0.2. Fixes *#5323*.
- [bdubbs] - Update to mpfr-4.2.1. Fixes *#5326*.
- [bdubbs] - Update to kbd-2.6.2. Fixes *#5318*.
- [bdubbs] - Update to gzip-1.13. Fixes *#5325*.
- [bdubbs] - Update to coreutils-9.4. Fixes *#5334*.
- [bdubbs] - Remove unused usb group. Fixes *#5331*.

- 2023-09-01
  - [bdubbs] - LFS-12.0 released.

1.4. Resources

1.4.1. FAQ

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

1.4.2. Mailing Lists

The `linuxfromscratch.org` server hosts a number of mailing lists used for the development of the LFS project. These lists include the main development and support lists, among others. If you cannot find an answer to your problem on the FAQ page, the next step would be to search the mailing lists at *https://www.linuxfromscratch.org/search.html*.

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

1.4.3. IRC

Several members of the LFS community offer assistance via Internet Relay Chat (IRC). Before using this support, please make sure your question is not already answered in the LFS FAQ or the mailing list archives. You can find the IRC network at `irc.libera.chat`. The support channel is named #lfs-support.

1.4.4. Mirror Sites

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

1.4.5. Contact Information

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

1.5. Help

Note

In case you've hit an issue building one package with the LFS instruction, we strongly discourage posting the issue directly onto the upstream support channel before discussing via a LFS support channel listed in Section 1.4, “Resources”. Doing so is often quite inefficient because the upstream maintainers are rarely familiar with LFS building procedure. Even if you've really hit an upstream issue, the LFS community can still help to isolate the information wanted by the upstream maintainers and make a proper report.

If you must ask a question directly via an upstream support channel, you shall at least note that many upstream projects have the support channels separated from the bug tracker. The “bug” reports for asking questions are considered invalid and may annoy upstream developers for these projects.

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

If you cannot find your problem listed in the FAQ, search the mailing lists at *https://www.linuxfromscratch.org/search.html*.

We also have a wonderful LFS community that is willing to offer assistance through the mailing lists and IRC (see the Section 1.4, “Resources” section of this book). However, we get several support questions every day, and many of them could have been easily answered by going to the FAQ or by searching the mailing lists first. So, for us to offer the best assistance possible, you should first do some research on your own. That allows us to focus on the more unusual support needs. If your searches do not produce a solution, please include all the relevant information (mentioned below) in your request for help.

1.5.1. Things to Mention

Apart from a brief explanation of the problem being experienced, any request for help should include these essential things:

- The version of the book being used (in this case 12.1)
- The host distribution and version being used to create LFS
- The output from the Host System Requirements script
- The package or section the problem was encountered in
- The exact error message, or a clear description of the problem
- Note whether you have deviated from the book at all

Note

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

1.5.2. Configure Script Problems

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

1.5.3. 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 all of the relevant information. Here is an example of the type of information to include from the **make** screen output.

```
gcc -DALIASPATH=\"/mnt/lfs/usr/share/locale:.\"
-DLOCALEDIR=\"/mnt/lfs/usr/share/locale\"
-DLIBDIR=\"/mnt/lfs/usr/lib\"
-DINCLUDEDIR=\"/mnt/lfs/usr/include\" -DHAVE_CONFIG_H -I. -I.
-g -O2 -c getopt1.c
gcc -g -O2 -static -o make ar.o arscan.o commands.o dir.o
expand.o file.o function.o getopt.o implicit.o job.o main.o
misc.o read.o remake.o rule.o signame.o variable.o vpath.o
default.o remote-stub.o version.o opt1.o
-lutil job.o: In function `load_too_high':
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference
to `getloadavg'
collect2: ld returned 1 exit status
make[2]: *** [make] Error 1
make[2]: Leaving directory `/lfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/lfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2
```

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

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

This is not enough information to 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 all the associated error messages.

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

Part II. Preparing for the Build

Chapter 2. Preparing the Host System

2.1. Introduction

In this chapter, the host tools needed for building LFS are checked and, if necessary, installed. Then a partition which will host the LFS system is prepared. We will create the partition itself, create a file system on it, and mount it.

2.2. Host System Requirements

2.2.1. Hardware

The LFS editors recommend that the system CPU have at least four cores and that the system have at least 8 GB of memory. Older systems that do not meet these requirements will still work, but the time to build packages will be significantly longer than documented.

2.2.2. Software

Your host system should have the following software with the minimum versions indicated. This should not be an issue for most modern Linux distributions. 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.

Earlier versions of the listed software packages may work, but have not been tested.

- **Bash-3.2** (/bin/sh should be a symbolic or hard link to bash)
- **Binutils-2.13.1** (Versions greater than 2.42 are not recommended as they have not been tested)
- **Bison-2.7** (/usr/bin/yacc should be a link to bison or a small script that executes bison)
- **Coreutils-8.1**
- **Diffutils-2.8.1**
- **Findutils-4.2.31**
- **Gawk-4.0.1** (/usr/bin/awk should be a link to gawk)
- **GCC-5.2** including the C++ compiler, **g++** (Versions greater than 13.2.0 are not recommended as they have not been tested). C and C++ standard libraries (with headers) must also be present so the C++ compiler can build hosted programs
- **Grep-2.5.1a**
- **Gzip-1.3.12**
- **Linux Kernel-4.19**

  The reason for the kernel version requirement is that we specify that version when building glibc in Chapter 5 and Chapter 8, so the workarounds for older kernels are not enabled and the compiled glibc is slightly faster and smaller. As at Feb 2024, 4.19 is the oldest kernel release still supported by the kernel developers. Some kernel releases older than 4.19 may be still supported by third-party teams, but they are not considered official upstream kernel releases; read *https://kernel.org/category/releases.html* for the details.

  If the host kernel is earlier than 4.19 you will need to replace the kernel with a more up-to-date version. There are two ways you can go about this. First, see if your Linux vendor provides a 4.19 or later kernel package. If so, you may wish to install it. If your vendor doesn't offer an acceptable kernel package, or you would prefer not to install it, you can compile a kernel yourself. Instructions for compiling the kernel and configuring the boot loader (assuming the host uses GRUB) are located in Chapter 10.

We require the host kernel to support UNIX 98 pseudo terminal (PTY). It should be enabled on all desktop or server distros shipping Linux 4.19 or a newer kernel. If you are building a custom host kernel, ensure `CONFIG_UNIX98_PTYS` is set to `y` in the kernel configuration.

- **M4-1.4.10**
- **Make-4.0**
- **Patch-2.5.4**
- **Perl-5.8.8**
- **Python-3.4**
- **Sed-4.1.5**
- **Tar-1.22**
- **Texinfo-5.0**
- **Xz-5.0.0**

Important

Note that the symlinks mentioned above are required to build an LFS system using the instructions contained within this book. Symlinks that point to other software (such as dash, mawk, etc.) may work, but are not tested or supported by the LFS development team, and may require either deviation from the instructions or additional patches to some packages.

To see whether your host system has all the appropriate versions, and the ability to compile programs, run the following commands:

```
cat > version-check.sh << "EOF"
#!/bin/bash
# A script to list version numbers of critical development tools

# If you have tools installed in other directories, adjust PATH here AND
# in ~lfs/.bashrc (section 4.4) as well.

LC_ALL=C
PATH=/usr/bin:/bin

bail() { echo "FATAL: $1"; exit 1; }
grep --version > /dev/null 2> /dev/null || bail "grep does not work"
sed '' /dev/null || bail "sed does not work"
sort   /dev/null || bail "sort does not work"

ver_check()
{
   if ! type -p $2 &>/dev/null
   then
     echo "ERROR: Cannot find $2 ($1)"; return 1;
   fi
   v=$($2 --version 2>&1 | grep -E -o '[0-9]+\.[0-9\.]+[a-z]*' | head -n1)
   if printf '%s\n' $3 $v | sort --version-sort --check &>/dev/null
   then
     printf "OK:    %-9s %-6s >= $3\n" "$1" "$v"; return 0;
   else
     printf "ERROR: %-9s is TOO OLD ($3 or later required)\n" "$1";
     return 1;
   fi
}

ver_kernel()
{
```

```
   kver=$(uname -r | grep -E -o '^[0-9\.]+')
   if printf '%s\n' $1 $kver | sort --version-sort --check &>/dev/null
   then
     printf "OK:    Linux Kernel $kver >= $1\n"; return 0;
   else
     printf "ERROR: Linux Kernel ($kver) is TOO OLD ($1 or later required)\n" "$kver";
     return 1;
   fi
}

# Coreutils first because --version-sort needs Coreutils >= 7.0
ver_check Coreutils      sort     8.1 || bail "Coreutils too old, stop"
ver_check Bash           bash     3.2
ver_check Binutils       ld       2.13.1
ver_check Bison          bison    2.7
ver_check Diffutils      diff     2.8.1
ver_check Findutils      find     4.2.31
ver_check Gawk           gawk     4.0.1
ver_check GCC            gcc      5.2
ver_check "GCC (C++)"    g++      5.2
ver_check Grep           grep     2.5.1a
ver_check Gzip           gzip     1.3.12
ver_check M4             m4       1.4.10
ver_check Make           make     4.0
ver_check Patch          patch    2.5.4
ver_check Perl           perl     5.8.8
ver_check Python         python3  3.4
ver_check Sed            sed      4.1.5
ver_check Tar            tar      1.22
ver_check Texinfo        texi2any 5.0
ver_check Xz             xz       5.0.0
ver_kernel 4.19

if mount | grep -q 'devpts on /dev/pts' && [ -e /dev/ptmx ]
then echo "OK:    Linux Kernel supports UNIX 98 PTY";
else echo "ERROR: Linux Kernel does NOT support UNIX 98 PTY"; fi

alias_check() {
   if $1 --version 2>&1 | grep -qi $2
   then printf "OK:    %-4s is $2\n" "$1";
   else printf "ERROR: %-4s is NOT $2\n" "$1"; fi
}
echo "Aliases:"
alias_check awk GNU
alias_check yacc Bison
alias_check sh Bash

echo "Compiler check:"
if printf "int main(){}" | g++ -x c++ -
then echo "OK:    g++ works";
else echo "ERROR: g++ does NOT work"; fi
rm -f a.out

if [ "$(nproc)" = "" ]; then
   echo "ERROR: nproc is not available or it produces empty output"
else
   echo "OK: nproc reports $(nproc) logical cores are available"
fi
EOF

bash version-check.sh
```

2.3. Building LFS in Stages

LFS is designed to be built in one session. That is, the instructions assume that the system will not be shut down during the process. This does not mean that the system has to be built in one sitting. The issue is that certain procedures must be repeated after a reboot when resuming LFS at different points.

2.3.1. Chapters 1–4

These chapters run commands on the host system. When restarting, be certain of one thing:

- Procedures performed as the `root` user after Section 2.4 must have the LFS environment variable set *FOR THE ROOT USER*.

2.3.2. Chapters 5–6

- The /mnt/lfs partition must be mounted.
- These two chapters *must* be done as user `lfs`. A **su - lfs** command must be issued before performing any task in these chapters. If you don't do that, you are at risk of installing packages to the host, and potentially rendering it unusable.
- The procedures in General Compilation Instructions are critical. If there is any doubt a package has been installed correctly, ensure the previously expanded tarball has been removed, then re-extract the package, and complete all the instructions in that section.

2.3.3. Chapters 7–10

- The /mnt/lfs partition must be mounted.
- A few operations, from “Changing Ownership” to “Entering the Chroot Environment,” must be done as the `root` user, with the LFS environment variable set for the `root` user.
- When entering chroot, the LFS environment variable must be set for `root`. The LFS variable is not used after the chroot environment has been entered.
- The virtual file systems must be mounted. This can be done before or after entering chroot by changing to a host virtual terminal and, as `root`, running the commands in Section 7.3.1, “Mounting and Populating /dev” and Section 7.3.2, “Mounting Virtual Kernel File Systems”.

2.4. Creating a New Partition

Like most other operating systems, LFS is usually installed on a dedicated partition. The recommended approach to building an LFS system is to use an available empty partition or, if you have enough unpartitioned space, to create one.

A minimal system requires a partition of around 10 gigabytes (GB). This is enough to store all the source tarballs and compile the packages. However, if the LFS system is intended to be the primary Linux system, additional software will probably be installed which will require additional space. A 30 GB partition is a reasonable size to provide for growth. The LFS system itself will not take up this much room. A large portion of this requirement is to provide sufficient free temporary storage as well as for adding additional capabilities after LFS is complete. Additionally, 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 LFS 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/sda` for the primary disk drive. 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.

Note

For experienced users, other partitioning schemes are possible. The new LFS system can be on a software *RAID* array or an *LVM* logical volume. However, some of these options require an *initramfs*, which is an advanced topic. These partitioning methodologies are not recommended for first time LFS users.

Remember the designation of the new partition (e.g., `sda5`). This book will refer to this as the LFS partition. Also remember the designation of the `swap` partition. These names will be needed later for the `/etc/fstab` file.

2.4.1. Other Partition Issues

Requests for advice on system partitioning are often posted on the LFS mailing lists. This is a highly subjective topic. The default for most distributions is to use the entire drive with the exception of one small swap partition. This is not optimal for LFS for several reasons. It reduces flexibility, makes sharing of data across multiple distributions or LFS builds more difficult, makes backups more time consuming, and can waste disk space through inefficient allocation of file system structures.

2.4.1.1. The Root Partition

A root LFS partition (not to be confused with the `/root` directory) of twenty gigabytes is a good compromise for most systems. It provides enough space to build LFS and most of BLFS, but is small enough so that multiple partitions can be easily created for experimentation.

2.4.1.2. The Swap Partition

Most distributions automatically create a swap partition. Generally the recommended size of the swap partition is about twice the amount of physical RAM, however this is rarely needed. If disk space is limited, hold the swap partition to two gigabytes and monitor the amount of disk swapping.

If you want to use the hibernation feature (suspend-to-disk) of Linux, it writes out the contents of RAM to the swap partition before turning off the machine. In this case the size of the swap partition should be at least as large as the system's installed RAM.

Swapping is never good. For mechanical hard drives you can generally tell if a system is swapping by just listening to disk activity and observing how the system reacts to commands. With an SSD you will not be able to hear swapping, but you can tell how much swap space is being used by running the **top** or **free** programs. Use of an SSD for a swap partition should be avoided if possible. The first reaction to swapping should be to check for an unreasonable command such as trying to edit a five gigabyte file. If swapping becomes a normal occurrence, the best solution is to purchase more RAM for your system.

2.4.1.3. The Grub Bios Partition

If the *boot disk* has been partitioned with a GUID Partition Table (GPT), then a small, typically 1 MB, partition must be created if it does not already exist. This partition is not formatted, but must be available for GRUB to use during installation of the boot loader. This partition will normally be labeled 'BIOS Boot' if using **fdisk** or have a code of *EF02* if using the **gdisk** command.

Note

The Grub Bios partition must be on the drive that the BIOS uses to boot the system. This is not necessarily the drive that holds the LFS root partition. The disks on a system may use different partition table types. The necessity of the Grub Bios partition depends only on the partition table type of the boot disk.

2.4.1.4. Convenience Partitions

There are several other partitions that are not required, but should be considered when designing a disk layout. The following list is not comprehensive, but is meant as a guide.

- /boot – Highly recommended. Use this partition to store kernels and other booting information. To minimize potential boot problems with larger disks, make this the first physical partition on your first disk drive. A partition size of 200 megabytes is adequate.
- /boot/efi – The EFI System Partition, which is needed for booting the system with UEFI. Read *the BLFS page* for details.
- /home – Highly recommended. Share your home directory and user customization across multiple distributions or LFS builds. The size is generally fairly large and depends on available disk space.
- /usr – In LFS, `/bin`, `/lib`, and `/sbin` are symlinks to their counterparts in `/usr`. So `/usr` contains all the binaries needed for the system to run. For LFS a separate partition for `/usr` is normally not needed. If you create it anyway, you should make a partition large enough to fit all the programs and libraries in the system. The root partition can be very small (maybe just one gigabyte) in this configuration, so it's suitable for a thin client or diskless workstation (where `/usr` is mounted from a remote server). However, you should be aware that an initramfs (not covered by LFS) will be needed to boot a system with a separate `/usr` partition.
- /opt – This directory is most useful for BLFS, where multiple large packages like KDE or Texlive can be installed without embedding the files in the /usr hierarchy. If used, 5 to 10 gigabytes is generally adequate.
- /tmp – A separate /tmp partition is rare, but useful if configuring a thin client. This partition, if used, will usually not need to exceed a couple of gigabytes. If you have enough RAM, you can mount a `tmpfs` on /tmp to make access to temporary files faster.
- /usr/src – This partition is very useful for providing a location to store BLFS source files and share them across LFS builds. It can also be used as a location for building BLFS packages. A reasonably large partition of 30-50 gigabytes provides plenty of room.

Any separate partition that you want automatically mounted when the system starts must be specified in the `/etc/fstab` file. Details about how to specify partitions will be discussed in Section 10.2, “Creating the /etc/fstab File”.

2.5. Creating a File System on the Partition

A partition is just a range of sectors on a disk drive, delimited by boundaries set in a partition table. Before the operating system can use a partition to store any files, the partition must be formatted to contain a file system, typically consisting of a label, directory blocks, data blocks, and an indexing scheme to locate a particular file on demand. The file system also helps the OS keep track of free space on the partition, reserve the needed sectors when a new file is created or an existing file is extended, and recycle the free data segments created when files are deleted. It may also provide support for data redundancy, and for error recovery.

LFS can use any file system recognized by the Linux kernel, but the most common types are ext3 and ext4. The choice of the right file system can be complex; it depends on the characteristics of the files and the size of the partition. For example:

ext2
: is suitable for small partitions that are updated infrequently such as /boot.

ext3
: is an upgrade to ext2 that includes a journal to help recover the partition's status in the case of an unclean shutdown. It is commonly used as a general purpose file system.

ext4
: is the latest version of the ext family of file systems. It provides several new capabilities including nano-second timestamps, creation and use of very large files (up to 16 TB), and speed improvements.

Other file systems, including FAT32, NTFS, ReiserFS, JFS, and XFS are useful for specialized purposes. More information about these file systems, and many others, can be found at *https://en.wikipedia.org/wiki/Comparison_of_file_systems*.

LFS assumes that the root file system (/) is of type ext4. To create an `ext4` file system on the LFS partition, issue the following command:

```
mkfs -v -t ext4 /dev/<xxx>
```

Replace *<xxx>* with the name of the LFS partition.

If you are using an existing `swap` partition, there is no need to format it. If a new `swap` partition was created, it will need to be initialized with this command:

```
mkswap /dev/<yyy>
```

Replace *<yyy>* with the name of the `swap` partition.

2.6. Setting The $LFS Variable

Throughout this book, the environment variable `LFS` will be used several times. You should ensure that this variable is always defined throughout the LFS build process. It should be set to the name of the directory where you will be building your LFS system - we will use `/mnt/lfs` as an example, but you may choose any directory name you want. If you are building LFS on a separate partition, this directory will be the mount point for the partition. Choose a directory location and set the variable with the following command:

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

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

Caution

Do not forget to check that `LFS` is set whenever you leave and reenter the current working environment (such as when doing a **su** to `root` or another user). Check that the `LFS` variable is set up properly with:

```
echo $LFS
```

Make sure the output shows the path to your LFS system's build location, which is `/mnt/lfs` if the provided example was followed. If the output is incorrect, use the command given earlier on this page to set `$LFS` to the correct directory name.

Note

One way to ensure that the LFS variable is always set is to edit the .bash_profile file in both your personal home directory and in /root/.bash_profile and enter the export command above. In addition, the shell specified in the /etc/passwd file for all users that need the LFS variable must be bash to ensure that the /root/.bash_profile file is incorporated as a part of the login process.

Another consideration is the method that is used to log into the host system. If logging in through a graphical display manager, the user's .bash_profile is not normally used when a virtual terminal is started. In this case, add the export command to the .bashrc file for the user and root. In addition, some distributions use an "if" test, and do not run the remaining .bashrc instructions for a non-interactive bash invocation. Be sure to place the export command ahead of the test for non-interactive use.

2.7. Mounting the New Partition

Now that a file system has been created, the partition must be mounted so the host system can access it. This book assumes that the file system is mounted at the directory specified by the LFS environment variable described in the previous section.

Strictly speaking, one cannot “mount a partition.” One mounts the *file system* embedded in that partition. But since a single partition can't contain more than one file system, people often speak of the partition and the associated file system as if they were one and the same.

Create the mount point and mount the LFS file system with these commands:

```
mkdir -pv $LFS
mount -v -t ext4 /dev/<xxx> $LFS
```

Replace <*xxx*> with the name of the LFS partition.

If you are using multiple partitions for LFS (e.g., one for / and another for /home), mount them like this:

```
mkdir -pv $LFS
mount -v -t ext4 /dev/<xxx> $LFS
mkdir -v $LFS/home
mount -v -t ext4 /dev/<yyy> $LFS/home
```

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 or nodev options). Run the **mount** command without any parameters to see what options are set for the mounted LFS partition. If nosuid and/or nodev are set, the partition must be remounted.

Warning

The above instructions assume that you will not restart your computer throughout the LFS process. If you shut down your system, you will either need to remount the LFS partition each time you restart the build process, or modify the host system's /etc/fstab file to automatically remount it when you reboot. For example, you might add this line to your /etc/fstab file:

```
/dev/<xxx>  /mnt/lfs ext4    defaults      1     1
```

If you use additional optional partitions, be sure to add them also.

If you are using a swap partition, ensure that it is enabled using the **swapon** command:

```
/sbin/swapon -v /dev/<zzz>
```

Replace <zzz> with the name of the swap partition.

Now that the new LFS partition is open for business, it's 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 in order to build 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 against using different versions, because the build commands for one version may not work with a different version, unless the different version is specified by an LFS erratum or security advisory. 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.

For some packages, the release tarball and the (Git or SVN) repository snapshot tarball for that release may be published with similar file names. A release tarball contains generated files (for example, a **configure** script generated by **autoconf**), in addition to the contents of the corresponding repository snapshot. The book uses release tarballs whenever possible. Using a repository snapshot instead of a release tarball specified by the book will cause problems.

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

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

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

```
mkdir -v $LFS/sources
```

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

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

There are several ways to obtain all the necessary packages and patches to build LFS:

- The files can be downloaded individually as described in the next two sections.
- For stable versions of the book, a tarball of all the needed files can be downloaded from one of the mirror sites listed at *https://www.linuxfromscratch.org/mirrors.html#files*.
- The files can be downloaded using **wget** and a wget-list as described below.

To download all of the packages and patches by using *wget-list-sysv* as an input to the **wget** command, use:

```
wget --input-file=wget-list-sysv --continue --directory-prefix=$LFS/sources
```

Additionally, starting with LFS-7.0, there is a separate file, *md5sums*, which can be used to verify that all the correct packages are available before proceeding. Place that file in `$LFS/sources` and run:

```
pushd $LFS/sources
  md5sum -c md5sums
popd
```

This check can be used after retrieving the needed files with any of the methods listed above.

If the packages and patches are downloaded as a non-`root` user, these files will be owned by the user. The file system records the owner by its UID, and the UID of a normal user in the host distro is not assigned in LFS. So the files will be left owned by an unnamed UID in the final LFS system. If you won't assign the same UID for your user in the LFS system, change the owners of these files to `root` now to avoid this issue:

```
chown root:root $LFS/sources/*
```

3.2. All Packages

Note

Read the *security advisories* before downloading packages to figure out if a newer version of any package should be used to avoid security vulnerabilities.

The upstream sources may remove old releases, especially when those releases contain a security vulnerability. If one URL below is not reachable, you should read the security advisories first to figure out if a newer version (with the vulnerability fixed) should be used. If not, try to download the removed package from a mirror. Although it's possible to download an old release from a mirror even if this release has been removed because of a vulnerability, it's not a good idea to use a release known to be vulnerable when building your system.

Download or otherwise obtain the following packages:

• Acl (2.3.2) - 363 KB:
Home page: *https://savannah.nongnu.org/projects/acl*
Download: *https://download.savannah.gnu.org/releases/acl/acl-2.3.2.tar.xz*
MD5 sum: `590765dee95907dbc3c856f7255bd669`

• Attr (2.5.2) - 484 KB:
Home page: *https://savannah.nongnu.org/projects/attr*
Download: *https://download.savannah.gnu.org/releases/attr/attr-2.5.2.tar.gz*
MD5 sum: `227043ec2f6ca03c0948df5517f9c927`

• Autoconf (2.72) - 1,360 KB:
Home page: *https://www.gnu.org/software/autoconf/*
Download: *https://ftp.gnu.org/gnu/autoconf/autoconf-2.72.tar.xz*
MD5 sum: `1be79f7106ab6767f18391c5e22be701`

• Automake (1.16.5) - 1,565 KB:
Home page: *https://www.gnu.org/software/automake/*
Download: *https://ftp.gnu.org/gnu/automake/automake-1.16.5.tar.xz*
MD5 sum: `4017e96f89fca45ca946f1c5db6be714`

• Bash (5.2.21) - 10,696 KB:
Home page: *https://www.gnu.org/software/bash/*
Download: *https://ftp.gnu.org/gnu/bash/bash-5.2.21.tar.gz*
MD5 sum: `ad5b38410e3bf0e9bcc20e2765f5e3f9`

• Bc (6.7.5) - 460 KB:
Home page: *https://git.gavinhoward.com/gavin/bc*
Download: *https://github.com/gavinhoward/bc/releases/download/6.7.5/bc-6.7.5.tar.xz*
MD5 sum: `e249b1f86f886d6fb71c15f72b65dd3d`

• Binutils (2.42) - 26,922 KB:
Home page: *https://www.gnu.org/software/binutils/*
Download: *https://sourceware.org/pub/binutils/releases/binutils-2.42.tar.xz*
MD5 sum: `a075178a9646551379bfb64040487715`

• Bison (3.8.2) - 2,752 KB:
Home page: *https://www.gnu.org/software/bison/*
Download: *https://ftp.gnu.org/gnu/bison/bison-3.8.2.tar.xz*
MD5 sum: `c28f119f405a2304ff0a7ccdcc629713`

• Bzip2 (1.0.8) - 792 KB:
Download: *https://www.sourceware.org/pub/bzip2/bzip2-1.0.8.tar.gz*
MD5 sum: `67e051268d0c475ea773822f7500d0e5`

• Check (0.15.2) - 760 KB:
Home page: *https://libcheck.github.io/check*
Download: *https://github.com/libcheck/check/releases/download/0.15.2/check-0.15.2.tar.gz*
MD5 sum: `50fcafcecde5a380415b12e9c574e0b2`

• Coreutils (9.4) - 5,840 KB:
Home page: *https://www.gnu.org/software/coreutils/*
Download: *https://ftp.gnu.org/gnu/coreutils/coreutils-9.4.tar.xz*
MD5 sum: `459e9546074db2834eefe5421f250025`

• DejaGNU (1.6.3) - 608 KB:
Home page: *https://www.gnu.org/software/dejagnu/*
Download: *https://ftp.gnu.org/gnu/dejagnu/dejagnu-1.6.3.tar.gz*
MD5 sum: `68c5208c58236eba447d7d6d1326b821`

• Diffutils (3.10) - 1,587 KB:
Home page: *https://www.gnu.org/software/diffutils/*
Download: *https://ftp.gnu.org/gnu/diffutils/diffutils-3.10.tar.xz*
MD5 sum: `2745c50f6f4e395e7b7d52f902d075bf`

• E2fsprogs (1.47.0) - 9,412 KB:
Home page: *http://e2fsprogs.sourceforge.net/*
Download: *https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.47.0/e2fsprogs-1.47.0.tar.gz*
MD5 sum: `6b4f18a33873623041857b4963641ee9`

• Elfutils (0.190) - 8,949 KB:
Home page: *https://sourceware.org/elfutils/*
Download: *https://sourceware.org/ftp/elfutils/0.190/elfutils-0.190.tar.bz2*
MD5 sum: `79ad698e61a052bea79e77df6a08bc4b`

• Expat (2.6.0) - 473 KB:
Home page: *https://libexpat.github.io/*
Download: *https://prdownloads.sourceforge.net/expat/expat-2.6.0.tar.xz*
MD5 sum: `bd169cb11f4b9bdfddadf9e88a5c4d4b`

• Expect (5.45.4) - 618 KB:
Home page: *https://core.tcl.tk/expect/*
Download: *https://prdownloads.sourceforge.net/expect/expect5.45.4.tar.gz*
MD5 sum: `00fce8de158422f5ccd2666512329bd2`

• File (5.45) - 1,218 KB:
Home page: *https://www.darwinsys.com/file/*
Download: *https://astron.com/pub/file/file-5.45.tar.gz*
MD5 sum: `26b2a96d4e3a8938827a1e572afd527a`

• Findutils (4.9.0) - 1,999 KB:
Home page: *https://www.gnu.org/software/findutils/*
Download: *https://ftp.gnu.org/gnu/findutils/findutils-4.9.0.tar.xz*
MD5 sum: `4a4a547e888a944b2f3af31d789a1137`

• Flex (2.6.4) - 1,386 KB:
Home page: *https://github.com/westes/flex*
Download: *https://github.com/westes/flex/releases/download/v2.6.4/flex-2.6.4.tar.gz*
MD5 sum: `2882e3179748cc9f9c23ec593d6adc8d`

• Flit-core (3.9.0) - 41 KB:
Home page: *https://pypi.org/project/flit-core/*
Download: *https://pypi.org/packages/source/f/flit-core/flit_core-3.9.0.tar.gz*
MD5 sum: `3bc52f1952b9a78361114147da63c35b`

• Gawk (5.3.0) - 3,356 KB:
Home page: *https://www.gnu.org/software/gawk/*
Download: *https://ftp.gnu.org/gnu/gawk/gawk-5.3.0.tar.xz*
MD5 sum: `97c5a7d83f91a7e1b2035ebbe6ac7abd`

• GCC (13.2.0) - 85,800 KB:
Home page: *https://gcc.gnu.org/*
Download: *https://ftp.gnu.org/gnu/gcc/gcc-13.2.0/gcc-13.2.0.tar.xz*
MD5 sum: `e0e48554cc6e4f261d55ddee9ab69075`

• GDBM (1.23) - 1,092 KB:
Home page: *https://www.gnu.org/software/gdbm/*
Download: *https://ftp.gnu.org/gnu/gdbm/gdbm-1.23.tar.gz*
MD5 sum: `8551961e36bf8c70b7500d255d3658ec`

• Gettext (0.22.4) - 10,016 KB:
Home page: *https://www.gnu.org/software/gettext/*
Download: *https://ftp.gnu.org/gnu/gettext/gettext-0.22.4.tar.xz*
MD5 sum: `2d8507d003ef3ddd1c172707ffa97ed8`

• Glibc (2.39) - 18,092 KB:
Home page: *https://www.gnu.org/software/libc/*
Download: *https://ftp.gnu.org/gnu/glibc/glibc-2.39.tar.xz*
MD5 sum: `be81e87f72b5ea2c0ffe2bedfeb680c6`

Note

The Glibc developers maintain a *Git branch* containing patches considered worthy for Glibc-2.39 but unfortunately developed after Glibc-2.39 release. The LFS editors will issue a security advisory if any security fix is added into the branch, but no actions will be taken for other newly added patches. You may review the patches yourself and incorporate some patches if you consider them important.

• GMP (6.3.0) - 2,046 KB:
Home page: *https://www.gnu.org/software/gmp/*
Download: *https://ftp.gnu.org/gnu/gmp/gmp-6.3.0.tar.xz*
MD5 sum: `956dc04e864001a9c22429f761f2c283`

• Gperf (3.1) - 1,188 KB:
Home page: *https://www.gnu.org/software/gperf/*
Download: *https://ftp.gnu.org/gnu/gperf/gperf-3.1.tar.gz*
MD5 sum: `9e251c0a618ad0824b51117d5d9db87e`

• Grep (3.11) - 1,664 KB:
Home page: *https://www.gnu.org/software/grep/*
Download: *https://ftp.gnu.org/gnu/grep/grep-3.11.tar.xz*
MD5 sum: `7c9bbd74492131245f7cdb291fa142c0`

• Groff (1.23.0) - 7,259 KB:
Home page: *https://www.gnu.org/software/groff/*
Download: *https://ftp.gnu.org/gnu/groff/groff-1.23.0.tar.gz*
MD5 sum: `5e4f40315a22bb8a158748e7d5094c7d`

• GRUB (2.12) - 6,524 KB:
Home page: *https://www.gnu.org/software/grub/*
Download: *https://ftp.gnu.org/gnu/grub/grub-2.12.tar.xz*
MD5 sum: `60c564b1bdc39d8e43b3aab4bc0fb140`

• Gzip (1.13) - 819 KB:
Home page: *https://www.gnu.org/software/gzip/*
Download: *https://ftp.gnu.org/gnu/gzip/gzip-1.13.tar.xz*
MD5 sum: `d5c9fc9441288817a4a0be2da0249e29`

• Iana-Etc (20240125) - 589 KB:
Home page: *https://www.iana.org/protocols*
Download: *https://github.com/Mic92/iana-etc/releases/download/20240125/iana-etc-20240125.tar.gz*
MD5 sum: `aed66d04de615d76c70890233081e584`

• Inetutils (2.5) - 1,632 KB:
Home page: *https://www.gnu.org/software/inetutils/*
Download: *https://ftp.gnu.org/gnu/inetutils/inetutils-2.5.tar.xz*
MD5 sum: `9e5a6dfd2d794dc056a770e8ad4a9263`

• Intltool (0.51.0) - 159 KB:
Home page: *https://freedesktop.org/wiki/Software/intltool*
Download: *https://launchpad.net/intltool/trunk/0.51.0/+download/intltool-0.51.0.tar.gz*
MD5 sum: `12e517cac2b57a0121cda351570f1e63`

• IPRoute2 (6.7.0) - 900 KB:
Home page: *https://www.kernel.org/pub/linux/utils/net/iproute2/*
Download: *https://www.kernel.org/pub/linux/utils/net/iproute2/iproute2-6.7.0.tar.xz*
MD5 sum: `35d8277d1469596b7edc07a51470a033`

• Jinja2 (3.1.3) - 264 KB:
Home page: *https://jinja.palletsprojects.com/en/3.1.x/*
Download: *https://pypi.org/packages/source/J/Jinja2/Jinja2-3.1.3.tar.gz*
MD5 sum: `caf5418c851eac59e70a78d9730d4cea`

• Kbd (2.6.4) - 1,470 KB:
Home page: *https://kbd-project.org/*
Download: *https://www.kernel.org/pub/linux/utils/kbd/kbd-2.6.4.tar.xz*
MD5 sum: `e2fd7adccf6b1e98eb1ae8d5a1ce5762`

• Kmod (31) - 558 KB:
Home page: *https://github.com/kmod-project/kmod*
Download: *https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-31.tar.xz*
MD5 sum: `6165867e1836d51795a11ea4762ff66a`

• Less (643) - 579 KB:
Home page: *https://www.greenwoodsoftware.com/less/*
Download: *https://www.greenwoodsoftware.com/less/less-643.tar.gz*
MD5 sum: `cf05e2546a3729492b944b4874dd43dd`

• LFS-Bootscripts (20230728) - 33 KB:
Download: *https://www.linuxfromscratch.org/lfs/downloads/12.1/lfs-bootscripts-20230728.tar.xz*
MD5 sum: `a236eaa9a1f699bc3fb6ab2acd7e7b6c`

• Libcap (2.69) - 185 KB:
Home page: *https://sites.google.com/site/fullycapable/*
Download: *https://www.kernel.org/pub/linux/libs/security/linux-privs/libcap2/libcap-2.69.tar.xz*
MD5 sum: `4667bacb837f9ac4adb4a1a0266f4b65`

• Libffi (3.4.4) - 1,331 KB:
Home page: *https://sourceware.org/libffi/*
Download: *https://github.com/libffi/libffi/releases/download/v3.4.4/libffi-3.4.4.tar.gz*
MD5 sum: `0da1a5ed7786ac12dcbaf0d499d8a049`

• Libpipeline (1.5.7) - 956 KB:
Home page: *https://libpipeline.nongnu.org/*
Download: *https://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.7.tar.gz*
MD5 sum: `1a48b5771b9f6c790fb4efdb1ac71342`

• Libtool (2.4.7) - 996 KB:
Home page: *https://www.gnu.org/software/libtool/*
Download: *https://ftp.gnu.org/gnu/libtool/libtool-2.4.7.tar.xz*
MD5 sum: `2fc0b6ddcd66a89ed6e45db28fa44232`

• Libxcrypt (4.4.36) - 610 KB:
Home page: *https://github.com/besser82/libxcrypt/*
Download: *https://github.com/besser82/libxcrypt/releases/download/v4.4.36/libxcrypt-4.4.36.tar.xz*
MD5 sum: `b84cd4104e08c975063ec6c4d0372446`

• Linux (6.7.4) - 138,130 KB:
Home page: *https://www.kernel.org/*
Download: *https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.7.4.tar.xz*
MD5 sum: `370e1b6155ae63133380e421146619e0`

Note

The Linux kernel is updated quite frequently, many times due to discoveries of security vulnerabilities. The latest available stable kernel version may be used, unless the errata page says otherwise.
For users with limited speed or expensive bandwidth who wish to update the Linux kernel, a baseline version of the package and patches can be downloaded separately. This may save some time or cost for a subsequent patch level upgrade within a minor release.

• M4 (1.4.19) - 1,617 KB:
Home page: *https://www.gnu.org/software/m4/*
Download: *https://ftp.gnu.org/gnu/m4/m4-1.4.19.tar.xz*
MD5 sum: `0d90823e1426f1da2fd872df0311298d`

• Make (4.4.1) - 2,300 KB:
Home page: *https://www.gnu.org/software/make/*
Download: *https://ftp.gnu.org/gnu/make/make-4.4.1.tar.gz*
MD5 sum: `c8469a3713cbbe04d955d4ae4be23eeb`

• Man-DB (2.12.0) - 1,941 KB:
Home page: *https://www.nongnu.org/man-db/*
Download: *https://download.savannah.gnu.org/releases/man-db/man-db-2.12.0.tar.xz*
MD5 sum: `67e0052fa200901b314fad7b68c9db27`

• Man-pages (6.06) - 2,116 KB:
Home page: *https://www.kernel.org/doc/man-pages/*
Download: *https://www.kernel.org/pub/linux/docs/man-pages/man-pages-6.06.tar.xz*
MD5 sum: `26b39e38248144156d437e1e10cb20bf`

• MarkupSafe (2.1.5) - 19 KB:
Home page: *https://palletsprojects.com/p/markupsafe/*
Download: *https://pypi.org/packages/source/M/MarkupSafe/MarkupSafe-2.1.5.tar.gz*
MD5 sum: `8fe7227653f2fb9b1ffe7f9f2058998a`

• Meson (1.3.2) - 2,172 KB:
Home page: *https://mesonbuild.com*
Download: *https://github.com/mesonbuild/meson/releases/download/1.3.2/meson-1.3.2.tar.gz*
MD5 sum: `2d0ebd3a24249617b1c4d30026380cf8`

• MPC (1.3.1) - 756 KB:
Home page: *https://www.multiprecision.org/*
Download: *https://ftp.gnu.org/gnu/mpc/mpc-1.3.1.tar.gz*
MD5 sum: `5c9bc658c9fd0f940e8e3e0f09530c62`

• MPFR (4.2.1) - 1,459 KB:
Home page: *https://www.mpfr.org/*
Download: *https://ftp.gnu.org/gnu/mpfr/mpfr-4.2.1.tar.xz*
MD5 sum: `523c50c6318dde6f9dc523bc0244690a`

• Ncurses (6.4-20230520) - 2,156 KB:
Home page: *https://www.gnu.org/software/ncurses/*
Download: *https://anduin.linuxfromscratch.org/LFS/ncurses-6.4-20230520.tar.xz*
MD5 sum: `c5367e829b6d9f3f97b280bb3e6bfbc3`

• Ninja (1.11.1) - 225 KB:
Home page: *https://ninja-build.org/*
Download: *https://github.com/ninja-build/ninja/archive/v1.11.1/ninja-1.11.1.tar.gz*
MD5 sum: `32151c08211d7ca3c1d832064f6939b0`

• OpenSSL (3.2.1) - 17,318 KB:
Home page: *https://www.openssl.org/*
Download: *https://www.openssl.org/source/openssl-3.2.1.tar.gz*
MD5 sum: `c239213887804ba00654884918b37441`

• Patch (2.7.6) - 766 KB:
Home page: *https://savannah.gnu.org/projects/patch/*
Download: *https://ftp.gnu.org/gnu/patch/patch-2.7.6.tar.xz*
MD5 sum: `78ad9937e4caadcba1526ef1853730d5`

• Perl (5.38.2) - 13,359 KB:
Home page: *https://www.perl.org/*
Download: *https://www.cpan.org/src/5.0/perl-5.38.2.tar.xz*
MD5 sum: `d3957d75042918a23ec0abac4a2b7e0a`

• Pkgconf (2.1.1) - 305 KB:
Home page: *http://pkgconf.org/*
Download: *https://distfiles.ariadne.space/pkgconf/pkgconf-2.1.1.tar.xz*
MD5 sum: `bc29d74c2483197deb9f1f3b414b7918`

• Procps (4.0.4) - 1,369 KB:
Home page: *https://gitlab.com/procps-ng/procps/*
Download: *https://sourceforge.net/projects/procps-ng/files/Production/procps-ng-4.0.4.tar.xz*
MD5 sum: `2f747fc7df8ccf402d03e375c565cf96`

• Psmisc (23.6) - 415 KB:
Home page: *https://gitlab.com/psmisc/psmisc*
Download: *https://sourceforge.net/projects/psmisc/files/psmisc/psmisc-23.6.tar.xz*
MD5 sum: `ed3206da1184ce9e82d607dc56c52633`

• Python (3.12.2) - 20,109 KB:
Home page: *https://www.python.org/*
Download: *https://www.python.org/ftp/python/3.12.2/Python-3.12.2.tar.xz*
MD5 sum: `e7c178b97bf8f7ccd677b94d614f7b3c`

• Python Documentation (3.12.2) - 8,065 KB:
Download: *https://www.python.org/ftp/python/doc/3.12.2/python-3.12.2-docs-html.tar.bz2*
MD5 sum: `8a6310f6288e7f60c3565277ec3b5279`

• Readline (8.2) - 2,973 KB:
Home page: *https://tiswww.case.edu/php/chet/readline/rltop.html*
Download: *https://ftp.gnu.org/gnu/readline/readline-8.2.tar.gz*
MD5 sum: `4aa1b31be779e6b84f9a96cb66bc50f6`

• Sed (4.9) - 1,365 KB:
Home page: *https://www.gnu.org/software/sed/*
Download: *https://ftp.gnu.org/gnu/sed/sed-4.9.tar.xz*
MD5 sum: `6aac9b2dbafcd5b7a67a8a9bcb8036c3`

• Setuptools (69.1.0) - 2,168 KB:
Home page: *https://pypi.org/project/setuptools/*
Download: *https://pypi.org/packages/source/s/setuptools/setuptools-69.1.0.tar.gz*
MD5 sum: `6f6eb780ce12c90d81ce243747ed7ab0`

• Shadow (4.14.5) - 1,765 KB:
Home page: *https://github.com/shadow-maint/shadow/*
Download: *https://github.com/shadow-maint/shadow/releases/download/4.14.5/shadow-4.14.5.tar.xz*
MD5 sum: `452b0e59f08bf618482228ba3732d0ae`

• Sysklogd (1.5.1) - 88 KB:
Home page: *https://www.infodrom.org/projects/sysklogd/*
Download: *https://www.infodrom.org/projects/sysklogd/download/sysklogd-1.5.1.tar.gz*
MD5 sum: `c70599ab0d037fde724f7210c2c8d7f8`

• Systemd (255) - 14,516 KB:
Home page: *https://www.freedesktop.org/wiki/Software/systemd/*
Download: *https://github.com/systemd/systemd/archive/v255/systemd-255.tar.gz*
MD5 sum: `521cda27409a9edf0370c128fae3e690`

• Systemd Man Pages(255) - 652 KB:
Home page: *https://www.freedesktop.org/wiki/Software/systemd/*
Download: *https://anduin.linuxfromscratch.org/LFS/systemd-man-pages-255.tar.xz*
MD5 sum: `1ebe54d7a80f9abf8f2d14ddfeb2432d`

Note

The Linux From Scratch team generates its own tarball of the man pages using the systemd source. This is done in order to avoid unnecessary dependencies.

• Sysvinit (3.08) - 263 KB:
Home page: *https://savannah.nongnu.org/projects/sysvinit*
Download: *https://github.com/slicer69/sysvinit/releases/download/3.08/sysvinit-3.08.tar.xz*
MD5 sum: `81a05f28d7b67533cfc778fcadea168c`

• Tar (1.35) - 2,263 KB:
Home page: *https://www.gnu.org/software/tar/*
Download: *https://ftp.gnu.org/gnu/tar/tar-1.35.tar.xz*
MD5 sum: `a2d8042658cfd8ea939e6d911eaf4152`

• Tcl (8.6.13) - 10,581 KB:
Home page: *http://tcl.sourceforge.net/*
Download: *https://downloads.sourceforge.net/tcl/tcl8.6.13-src.tar.gz*
MD5 sum: `0e4358aade2f5db8a8b6f2f6d9481ec2`

• Tcl Documentation (8.6.13) - 1,165 KB:
Download: *https://downloads.sourceforge.net/tcl/tcl8.6.13-html.tar.gz*
MD5 sum: `4452f2f6d557f5598cca17b786d6eb68`

• Texinfo (7.1) - 5,416 KB:
Home page: *https://www.gnu.org/software/texinfo/*
Download: *https://ftp.gnu.org/gnu/texinfo/texinfo-7.1.tar.xz*
MD5 sum: `edd9928b4a3f82674bcc3551616eef3b`

• Time Zone Data (2024a) - 444 KB:
Home page: *https://www.iana.org/time-zones*
Download: *https://www.iana.org/time-zones/repository/releases/tzdata2024a.tar.gz*
MD5 sum: `2349edd8335245525cc082f2755d5bf4`

• Udev-lfs Tarball (udev-lfs-20230818) - 10 KB:
Download: *https://anduin.linuxfromscratch.org/LFS/udev-lfs-20230818.tar.xz*
MD5 sum: `acd4360d8a5c3ef320b9db88d275dae6`

• Util-linux (2.39.3) - 8,327 KB:
Home page: *https://git.kernel.org/pub/scm/utils/util-linux/util-linux.git/*
Download: *https://www.kernel.org/pub/linux/utils/util-linux/v2.39/util-linux-2.39.3.tar.xz*
MD5 sum: `f3591e6970c017bb4bcd24ae762a98f5`

• Vim (9.1.0041) - 17,224 KB:
Home page: *https://www.vim.org*
Download: *https://github.com/vim/vim/archive/v9.1.0041/vim-9.1.0041.tar.gz*
MD5 sum: `79dfe62be5d347b1325cbd5ce2a1f9b3`

Note
The version of vim changes daily. To get the latest version, go to *https://github.com/vim/vim/tags*.

• Wheel (0.42.0) - 97 KB:
Home page: *https://pypi.org/project/wheel/*
Download: *https://pypi.org/packages/source/w/wheel/wheel-0.42.0.tar.gz*
MD5 sum: `802ad6e5f9336fcb1c76b7593f0cd22d`

• XML::Parser (2.47) - 276 KB:
Home page: *https://github.com/chorny/XML-Parser*
Download: *https://cpan.metacpan.org/authors/id/T/TO/TODDR/XML-Parser-2.47.tar.gz*
MD5 sum: `89a8e82cfd2ad948b349c0a69c494463`

• Xz Utils (5.4.6) - 1,645 KB:
Home page: *https://tukaani.org/xz*
Download: *https://github.com/tukaani-project/xz/releases/download/v5.4.6/xz-5.4.6.tar.xz*
MD5 sum: `7ade7bd1181a731328f875bec62a9377`

• Zlib (1.3.1) - 1,478 KB:
Home page: *https://zlib.net/*
Download: *https://zlib.net/fossils/zlib-1.3.1.tar.gz*
MD5 sum: `9855b6d802d7fe5b7bd5b196a2271655`

• Zstd (1.5.5) - 2,314 KB:
Home page: *https://facebook.github.io/zstd/*
Download: *https://github.com/facebook/zstd/releases/download/v1.5.5/zstd-1.5.5.tar.gz*
MD5 sum: `63251602329a106220e0a5ad26ba656f`

Total size of these packages: about 504 MB

3.3. Needed Patches

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

• Bash Upstream Fixes Patch - 5.9 KB:
Download: *https://www.linuxfromscratch.org/patches/lfs/12.1/bash-5.2.21-upstream_fixes-1.patch*
MD5 sum: `2d1691a629c558e894dbb78ee6bf34ef`

• Bzip2 Documentation Patch - 1.6 KB:
Download: *https://www.linuxfromscratch.org/patches/lfs/12.1/bzip2-1.0.8-install_docs-1.patch*
MD5 sum: `6a5ac7e89b791aae556de0f745916f7f`

• Coreutils Internationalization Fixes Patch - 166 KB:
Download: *https://www.linuxfromscratch.org/patches/lfs/12.1/coreutils-9.4-i18n-1.patch*
MD5 sum: `cca7dc8c73147444e77bc45d210229bb`

• Glibc FHS Patch - 2.8 KB:
Download: *https://www.linuxfromscratch.org/patches/lfs/12.1/glibc-2.39-fhs-1.patch*
MD5 sum: `9a5997c3452909b1769918c759eff8a2`

• Kbd Backspace/Delete Fix Patch - 12 KB:
Download: *https://www.linuxfromscratch.org/patches/lfs/12.1/kbd-2.6.4-backspace-1.patch*
MD5 sum: `f75cca16a38da6caa7d52151f7136895`

• Readline Upstream Fix Patch - 13 KB:
Download: *https://www.linuxfromscratch.org/patches/lfs/12.1/readline-8.2-upstream_fixes-3.patch*
MD5 sum: `9ed497b6cb8adcb8dbda9dee9ebce791`

• Sysvinit Consolidated Patch - 2.5 KB:
Download: *https://www.linuxfromscratch.org/patches/lfs/12.1/sysvinit-3.08-consolidated-1.patch*
MD5 sum: `17ffccbb8e18c39e8cedc32046f3a475`

Total size of these patches: about 203.8 KB

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

Chapter 4. Final Preparations

4.1. Introduction

In this chapter, we will perform a few additional tasks to prepare for building the temporary system. We will create a set of directories in `$LFS` (in which we will install the temporary tools), add an unprivileged user, and create an appropriate build environment for that user. We will also explain the units of time ("SBUs") we use to measure how long it takes to build LFS packages, and provide some information about package test suites.

4.2. Creating a Limited Directory Layout in the LFS Filesystem

In this section, we begin populating the LFS filesystem with the pieces that will constitute the final Linux system. The first step is to create a limited directory hierarchy, so that the programs compiled in Chapter 6 (as well as glibc and libstdc++ in Chapter 5) can be installed in their final location. We do this so those temporary programs will be overwritten when the final versions are built in Chapter 8.

Create the required directory layout by issuing the following commands as `root`:

```
mkdir -pv $LFS/{etc,var} $LFS/usr/{bin,lib,sbin}

for i in bin lib sbin; do
  ln -sv usr/$i $LFS/$i
done

case $(uname -m) in
  x86_64) mkdir -pv $LFS/lib64 ;;
esac
```

Programs in Chapter 6 will be compiled with a cross-compiler (more details can be found in section Toolchain Technical Notes). This cross-compiler will be installed in a special directory, to separate it from the other programs. Still acting as `root`, create that directory with this command:

```
mkdir -pv $LFS/tools
```

Note

The LFS editors have deliberately decided not to use a `/usr/lib64` directory. Several steps are taken to be sure the toolchain will not use it. If for any reason this directory appears (either because you made an error in following the instructions, or because you installed a binary package that created it after finishing LFS), it may break your system. You should always be sure this directory does not exist.

4.3. Adding the LFS User

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

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

This is what the command line options mean:

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

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

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

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

lfs
This is the name of the new user.

If you want to log in as `lfs` or switch to `lfs` from a non-`root` user (as opposed to switching to user `lfs` when logged in as `root`, which does not require the `lfs` user to have a password), you need to set a password for `lfs`. Issue the following command as the `root` user to set the password:

```
passwd lfs
```

Grant `lfs` full access to all the directories under `$LFS` by making `lfs` the owner:

```
chown -v lfs $LFS/{usr{,/*},lib,var,etc,bin,sbin,tools}
case $(uname -m) in
  x86_64) chown -v lfs $LFS/lib64 ;;
esac
```

Note

In some host systems, the following **su** command does not complete properly and suspends the login for the `lfs` user to the background. If the prompt "lfs:~$" does not appear immediately, entering the **fg** command will fix the issue.

Next, start a shell running as user `lfs`. This can be done by logging in as `lfs` on a virtual console, or with the following substitute/switch user command:

```
su - lfs
```

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

4.4. Setting Up the Environment

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

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

When logged on as user `lfs`, or when switched to the `lfs` user using an **su** command with the "`-`" option, the initial shell is 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 new instance of the shell is a *non-login* shell, which does not read, and execute, the contents of the `/etc/profile` or `.bash_profile` files, but rather reads, and executes, the `.bashrc` file instead. Create the `.bashrc` file now:

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
LFS_TGT=$(uname -m)-lfs-linux-gnu
PATH=/usr/bin
if [ ! -L /bin ]; then PATH=/bin:$PATH; fi
PATH=$LFS/tools/bin:$PATH
CONFIG_SITE=$LFS/usr/share/config.site
export LFS LC_ALL LFS_TGT PATH CONFIG_SITE
EOF
```

The meaning of the settings in `.bashrc`

`set +h`

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 to 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. Switching off the hash function forces the shell to search the `PATH` whenever a program is to be run. As such, the shell will find the newly compiled tools in `$LFS/tools/bin` as soon as they are available without remembering a previous version of the same program provided by the host distro, in `/usr/bin` or `/bin`.

`umask 022`

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).

`LFS=/mnt/lfs`

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

`LC_ALL=POSIX`

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 in the cross-compilation environment.

`LFS_TGT=$(uname -m)-lfs-linux-gnu`

The `LFS_TGT` variable sets a non-default, but compatible machine description for use when building our cross-compiler and linker and when cross-compiling our temporary toolchain. More information is provided by Toolchain Technical Notes.

`PATH=/usr/bin`

Many modern Linux distributions have merged `/bin` and `/usr/bin`. When this is the case, the standard `PATH` variable should be set to `/usr/bin/` for the Chapter 6 environment. When this is not the case, the following line adds `/bin` to the path.

`if [ ! -L /bin ]; then PATH=/bin:$PATH; fi`

If `/bin` is not a symbolic link, it must be added to the `PATH` variable.

`PATH=$LFS/tools/bin:$PATH`

By putting `$LFS/tools/bin` ahead of the standard `PATH`, the cross-compiler installed at the beginning of Chapter 5 is picked up by the shell immediately after its installation. This, combined with turning off hashing, limits the risk that the compiler from the host is used instead of the cross-compiler.

CONFIG_SITE=$LFS/usr/share/config.site

In Chapter 5 and Chapter 6, if this variable is not set, **configure** scripts may attempt to load configuration items specific to some distributions from `/usr/share/config.site` on the host system. Override it to prevent potential contamination from the host.

export ...

While the preceding commands have set some variables, in order to make them visible within any sub-shells, we export them.

Important

Several commercial distributions add an undocumented instantiation of `/etc/bash.bashrc` to the initialization of **bash**. This file has the potential to modify the `lfs` user's environment in ways that can affect the building of critical LFS packages. To make sure the `lfs` user's environment is clean, check for the presence of `/etc/bash.bashrc` and, if present, move it out of the way. As the `root` user, run:

```
[ ! -e /etc/bash.bashrc ] || mv -v /etc/bash.bashrc /etc/bash.bashrc.NOUSE
```

When the `lfs` user is no longer needed (at the beginning of Chapter 7), you may safely restore `/etc/bash.bashrc` (if desired).

Note that the LFS Bash package we will build in Section 8.35, “Bash-5.2.21” is not configured to load or execute `/etc/bash.bashrc`, so this file is useless on a completed LFS system.

For many modern systems with multiple processors (or cores) the compilation time for a package can be reduced by performing a "parallel make" by telling the make program how many processors are available via a command line option or an environment variable. For instance, an Intel Core i9-13900K processor has 8 P (performance) cores and 16 E (efficiency) cores, and a P core can simultaneously run two threads so each P core are modeled as two logical cores by the Linux kernel. As the result there are 32 logical cores in total. One obvious way to use all these logical cores is allowing **make** to spawn up to 32 build jobs. This can be done by passing the *-j32* option to **make**:

```
make -j32
```

Or set the `MAKEFLAGS` environment variable and its content will be automatically used by **make** as command line options:

```
export MAKEFLAGS=-j32
```

Important

Never pass a *-j* option without a number to **make** or set such an option in `MAKEFLAGS`. Doing so will allow **make** to spawn infinite build jobs and cause system stability problems.

To use all logical cores available for building packages in Chapter 5 and Chapter 6, set `MAKEFLAGS` now in `.bashrc`:

```
cat >> ~/.bashrc << "EOF"
export MAKEFLAGS=-j$(nproc)
EOF
```

Replace *$(nproc)* with the number of logical cores you want to use if you don't want to use all the logical cores.

Finally, to ensure the environment is fully prepared for building the temporary tools, force the **bash** shell to read the new user profile:

```
source ~/.bash_profile
```

4.5. About SBUs

Many people would like to know beforehand approximately how long it takes to compile and install each package. Because Linux From Scratch can be built on many different systems, it is impossible to provide absolute time estimates. The biggest package (gcc) will take approximately 5 minutes on the fastest systems, but could take days on slower systems! Instead of providing actual times, the Standard Build Unit (SBU) measure will be used instead.

The SBU measure works as follows. The first package to be compiled is binutils in Chapter 5. The time it takes to compile using one core is what we will refer to as the Standard Build Unit or SBU. All other compile times will be expressed in terms of this unit of time.

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

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

Note

When multiple processors are used in this way, the SBU units in the book will vary even more than they normally would. In some cases, the make step will simply fail. Analyzing the output of the build process will also be more difficult because the lines from different processes will be interleaved. If you run into a problem with a build step, revert to a single processor build to properly analyze the error messages.

The times presented here are based upon using four cores (-j4). The times in Chapter 8 also include the time to run the regression tests for the package unless specified otherwise.

4.6. About the Test Suites

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

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

Note

Running the test suites in Chapter 5 and Chapter 6 is pointless; since the test programs are compiled with a cross-compiler, they probably can't run on the build host.

A common issue with running the test suites for binutils and GCC is running out of pseudo terminals (PTYs). This can result in a large number of failing tests. This may happen for several reasons, but the most likely cause is that the host system does not have the `devpts` file system set up correctly. This issue is discussed in greater detail at *https://www.linuxfromscratch.org/lfs/faq.html#no-ptys*.

Sometimes package test suites will fail for reasons which the developers are aware of and have deemed non-critical. Consult the logs located at *https://www.linuxfromscratch.org/lfs/build-logs/12.1/* to verify whether or not these failures are expected. This site is valid for all test suites throughout this book.

Part III. Building the LFS Cross Toolchain and Temporary Tools

Important Preliminary Material

Introduction

This part is divided into three stages: first, building a cross compiler and its associated libraries; second, using this cross toolchain to build several utilities in a way that isolates them from the host distribution; and third, entering the chroot environment (which further improves host isolation) and constructing the remaining tools needed to build the final system.

Important

This is where the real work of building a new system begins. Be very careful to follow the instructions exactly as the book shows them. You should try to understand what each command does, and no matter how eager you are to finish your build, you should refrain from blindly typing the commands as shown. Read the documentation when there is something you do not understand. Also, keep track of your typing and of the output of commands, by using the **tee** utility to send the terminal output to a file. This makes debugging easier if something goes wrong.

The next section is a technical introduction to the build process, while the following one presents **very important** general instructions.

Toolchain Technical Notes

This section explains some of the rationale and technical details behind the overall build method. Don't try to immediately understand everything in this section. Most of this information will be clearer after performing an actual build. Come back and re-read this chapter at any time during the build process.

The overall goal of Chapter 5 and Chapter 6 is to produce a temporary area containing a set of tools that are known to be good, and that are isolated from the host system. By using the **chroot** command, the compilations in the remaining chapters will be isolated within that environment, ensuring a clean, trouble-free build of the target LFS system. The build process has been designed to minimize the risks for new readers, and to provide the most educational value at the same time.

This build process is based on *cross-compilation*. Cross-compilation is normally used to build a compiler and its associated toolchain for a machine different from the one that is used for the build. This is not strictly necessary for LFS, since the machine where the new system will run is the same as the one used for the build. But cross-compilation has one great advantage: anything that is cross-compiled cannot depend on the host environment.

About Cross-Compilation

Note

The LFS book is not (and does not contain) a general tutorial to build a cross- (or native) toolchain. Don't use the commands in the book for a cross-toolchain for some purpose other than building LFS, unless you really understand what you are doing.

Cross-compilation involves some concepts that deserve a section of their own. Although this section may be omitted on a first reading, coming back to it later will help you gain a fuller understanding of the process.

Let us first define some terms used in this context.

The build
: is the machine where we build programs. Note that this machine is also referred to as the “host.”

The host
: is the machine/system where the built programs will run. Note that this use of “host” is not the same as in other sections.

The target
: is only used for compilers. It is the machine the compiler produces code for. It may be different from both the build and the host.

As an example, let us imagine the following scenario (sometimes referred to as “Canadian Cross”). We have a compiler on a slow machine only, let's call it machine A, and the compiler ccA. We also have a fast machine (B), but no compiler for (B), and we want to produce code for a third, slow machine (C). We will build a compiler for machine C in three stages.

Stage	Build	Host	Target	Action
1	A	A	B	Build cross-compiler cc1 using ccA on machine A.
2	A	B	C	Build cross-compiler cc2 using cc1 on machine A.
3	B	C	C	Build compiler ccC using cc2 on machine B.

Then, all the programs needed by machine C can be compiled using cc2 on the fast machine B. Note that unless B can run programs produced for C, there is no way to test the newly built programs until machine C itself is running. For example, to run a test suite on ccC, we may want to add a fourth stage:

Stage	Build	Host	Target	Action
4	C	C	C	Rebuild and test ccC using ccC on machine C.

In the example above, only cc1 and cc2 are cross-compilers, that is, they produce code for a machine different from the one they are run on. The other compilers ccA and ccC produce code for the machine they are run on. Such compilers are called *native* compilers.

Implementation of Cross-Compilation for LFS

Note

All the cross-compiled packages in this book use an autoconf-based building system. The autoconf-based building system accepts system types in the form cpu-vendor-kernel-os, referred to as the system triplet. Since the vendor field is often irrelevant, autoconf lets you omit it.

An astute reader may wonder why a “triplet” refers to a four component name. The kernel field and the os field began as a single “system” field. Such a three-field form is still valid today for some systems, for example, `x86_64-unknown-freebsd`. But two systems can share the same kernel and still be too different to use the same triplet to describe them. For example, Android running on a mobile phone is completely different from Ubuntu running on an ARM64 server, even though they are both running on the same type of CPU (ARM64) and using the same kernel (Linux).

Without an emulation layer, you cannot run an executable for a server on a mobile phone or vice versa. So the “system” field has been divided into kernel and os fields, to designate these systems unambiguously. In our example, the Android system is designated `aarch64-unknown-linux-android`, and the Ubuntu system is designated `aarch64-unknown-linux-gnu`.

The word “triplet” remains embedded in the lexicon. A simple way to determine your system triplet is to run the **config.guess** script that comes with the source for many packages. Unpack the binutils sources, run the script **`./config.guess`**, and note the output. For example, for a 32-bit Intel processor the output will be *i686-pc-linux-gnu*. On a 64-bit system it will be *x86_64-pc-linux-gnu*. On most Linux systems the even simpler **gcc -dumpmachine** command will give you similar information.

You should also be aware of the name of the platform's dynamic linker, often referred to as the dynamic loader (not to be confused with the standard linker **ld** that is part of binutils). The dynamic linker provided by package glibc finds and loads the shared libraries needed by a program, prepares the program to run, and then runs it. The name of the dynamic linker for a 32-bit Intel machine is `ld-linux.so.2`; it's `ld-linux-x86-64.so.2` on 64-bit systems. A sure-fire way to determine the name of the dynamic linker is to inspect a random binary from the host system by running: **`readelf -l <name of binary> | grep interpreter`** and noting the output. The authoritative reference covering all platforms is in the `shlib-versions` file in the root of the glibc source tree.

In order to fake a cross-compilation in LFS, the name of the host triplet is slightly adjusted by changing the "vendor" field in the `LFS_TGT` variable so it says "lfs". We also use the *`--with-sysroot`* option when building the cross-linker and cross-compiler, to tell them where to find the needed host files. This ensures that none of the other programs built in Chapter 6 can link to libraries on the build machine. Only two stages are mandatory, plus one more for tests.

Stage	Build	Host	Target	Action
1	pc	pc	lfs	Build cross-compiler cc1 using cc-pc on pc.

Stage	Build	Host	Target	Action
2	pc	lfs	lfs	Build compiler cc-lfs using cc1 on pc.
3	lfs	lfs	lfs	Rebuild and test cc-lfs using cc-lfs on lfs.

In the preceding table, “on pc” means the commands are run on a machine using the already installed distribution. “On lfs” means the commands are run in a chrooted environment.

This is not yet the end of the story. The C language is not merely a compiler; it also defines a standard library. In this book, the GNU C library, named glibc, is used (there is an alternative, "musl"). This library must be compiled for the LFS machine; that is, using the cross-compiler cc1. But the compiler itself uses an internal library providing complex subroutines for functions not available in the assembler instruction set. This internal library is named libgcc, and it must be linked to the glibc library to be fully functional. Furthermore, the standard library for C++ (libstdc++) must also be linked with glibc. The solution to this chicken and egg problem is first to build a degraded cc1-based libgcc, lacking some functionalities such as threads and exception handling, and then to build glibc using this degraded compiler (glibc itself is not degraded), and also to build libstdc++. This last library will lack some of the functionality of libgcc.

The upshot of the preceding paragraph is that cc1 is unable to build a fully functional libstdc++ with the degraded libgcc, but cc1 is the only compiler available for building the C/C++ libraries during stage 2. There are two reasons we don't immediately use the compiler built in stage 2, cc-lfs, to build those libraries.

- Generally speaking, cc-lfs cannot run on pc (the host system). Even though the triplets for pc and lfs are compatible with each other, an executable for lfs must depend on glibc-2.39; the host distro may utilize either a different implementation of libc (for example, musl), or a previous release of glibc (for example, glibc-2.13).
- Even if cc-lfs can run on pc, using it on pc would create a risk of linking to the pc libraries, since cc-lfs is a native compiler.

So when we build gcc stage 2, we instruct the building system to rebuild libgcc and libstdc++ with cc1, but we link libstdc++ to the newly rebuilt libgcc instead of the old, degraded build. This makes the rebuilt libstdc++ fully functional.

In Chapter 8 (or “stage 3”), all the packages needed for the LFS system are built. Even if a package has already been installed into the LFS system in a previous chapter, we still rebuild the package. The main reason for rebuilding these packages is to make them stable: if we reinstall an LFS package on a completed LFS system, the reinstalled content of the package should be the same as the content of the same package when first installed in Chapter 8. The temporary packages installed in Chapter 6 or Chapter 7 cannot satisfy this requirement, because some of them are built without optional dependencies, and autoconf cannot perform some feature checks in Chapter 6 because of cross-compilation, causing the temporary packages to lack optional features, or use suboptimal code routines. Additionally, a minor reason for rebuilding the packages is to run the test suites.

Other Procedural Details

The cross-compiler will be installed in a separate `$LFS/tools` directory, since it will not be part of the final system.

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

Binutils installs its assembler and linker in two locations, `$LFS/tools/bin` and `$LFS/tools/$LFS_TGT/bin`. The tools in one location are hard linked to the other. An important facet of the linker is its library search order. Detailed information can be obtained from **ld** by passing it the *`--verbose`* flag. For example, **$LFS_TGT-ld --verbose | grep SEARCH** will illustrate the current search paths and their order. (Note that this example can be run as shown only while logged in as user `lfs`. If you come back to this page later, replace **$LFS_TGT-ld** with **ld**).

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

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

This is important for the reasons mentioned above. It also demonstrates that gcc's configure script does not search the PATH directories to find which tools to use. However, during the actual operation of **gcc** itself, the same search paths are not necessarily used. To find out which standard linker **gcc** will use, run: **$LFS_TGT-gcc -print-prog-name=ld**. (Again, remove the **$LFS_TGT-** prefix if coming back to this later.)

Detailed information can be obtained from **gcc** by passing it the *`-v`* command line option while compiling a program. For example, **$LFS_TGT-gcc -v *`example.c`*** (or without **$LFS_TGT-** if coming back later) will show detailed information about the preprocessor, compilation, and assembly stages, including **gcc**'s search paths for included headers and their order.

Next up: sanitized Linux API headers. These allow the standard C library (glibc) to interface with features that the Linux kernel will provide.

Next comes glibc. The most important considerations for building glibc are the compiler, binary tools, and kernel headers. The compiler is generally not an issue since glibc will always use the compiler relating to the *`--host`* parameter passed to its configure script; e.g., in our case, the compiler will be **$LFS_TGT-gcc**. The binary tools and kernel headers can be a bit more complicated. Therefore, we take no risks and use the available configure switches to enforce the correct selections. After the run of **configure**, check the contents of the `config.make` file in the `build` directory for all important details. Note the use of *`CC="$LFS_TGT-gcc"`* (with `$LFS_TGT` expanded) to control which binary tools are used and the use of the *`-nostdinc`* and *`-isystem`* flags to control the compiler's include search path. These items highlight an important aspect of the glibc package—it is very self-sufficient in terms of its build machinery, and generally does not rely on toolchain defaults.

As mentioned above, the standard C++ library is compiled next, followed in Chapter 6 by other programs that must be cross-compiled to break circular dependencies at build time. The install step of all those packages uses the `DESTDIR` variable to force installation in the LFS filesystem.

At the end of Chapter 6 the native LFS compiler is installed. First binutils-pass2 is built, in the same `DESTDIR` directory as the other programs, then the second pass of gcc is constructed, omitting some non-critical libraries. Due to some weird logic in gcc's configure script, `CC_FOR_TARGET` ends up as **cc** when the host is the same as the target, but different from the build system. This is why *`CC_FOR_TARGET=$LFS_TGT-gcc`* is declared explicitly as one of the configuration options.

Upon entering the chroot environment in Chapter 7, the temporary installations of programs needed for the proper operation of the toolchain are performed. From this point onwards, the core toolchain is self-contained and self-hosted. In Chapter 8, final versions of all the packages needed for a fully functional system are built, tested, and installed.

General Compilation Instructions

Caution

During a development cycle of LFS, the instructions in the book are often modified to adapt for a package update or take the advantage of new features from updated packages. Mixing up the instructions of different versions of the LFS book can cause subtle breakages. This kind of issue is generally a result from reusing some script created for a prior LFS release. Such a reuse is strongly discouraged. If you are reusing scripts for a prior LFS release for any reason, you'll need to be very careful to update the scripts to match current version of the LFS book.

Here are some things you should know about building each package:

- Several packages are patched before compilation, but only when the patch is needed to circumvent a problem. A patch is often needed in both the current and the following chapters, but sometimes, when the same package is built more than once, the patch is not needed right away. 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; the patch was still successfully applied.
- During the compilation of most packages, some warnings will scroll by on the screen. These are normal and can safely be ignored. These warnings are usually about deprecated, but not invalid, use of the C or C++ syntax. C standards change fairly often, and some packages have not yet been updated. This is not a serious problem, but it does cause the warnings to appear.
- Check one last time that the `LFS` environment variable is set up properly:

```
echo $LFS
```

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

- Finally, two important items must be emphasized:

Important

The build instructions assume that the Host System Requirements, including symbolic links, have been set properly:

- **bash** is the shell in use.
- **sh** is a symbolic link to **bash**.
- **/usr/bin/awk** is a symbolic link to **gawk**.
- **/usr/bin/yacc** is a symbolic link to **bison**, or to a small script that executes bison.

Important

Here is a synopsis of the build process.

1. Place all the sources and patches in a directory that will be accessible from the chroot environment, such as `/mnt/lfs/sources/`.
2. Change to the `/mnt/lfs/sources/` directory.
3. For each package:
   a. Using the **tar** program, extract the package to be built. In Chapter 5 and Chapter 6, ensure you are the *lfs* user when extracting the package.

      Do not use any method except the **tar** command to extract the source code. Notably, using the **cp -R** command to copy the source code tree somewhere else can destroy links and timestamps in the source tree, and cause the build to fail.
   b. Change to the directory created when the package was extracted.
   c. Follow the instructions for building the package.
   d. Change back to the sources directory when the build is complete.
   e. Delete the extracted source directory unless instructed otherwise.

Chapter 5. Compiling a Cross-Toolchain

5.1. Introduction

This chapter shows how to build a cross-compiler and its associated tools. Although here cross-compilation is faked, the principles are the same as for a real cross-toolchain.

The programs compiled in this chapter will be installed under the `$LFS/tools` directory to keep them separate from the files installed in the following chapters. The libraries, on the other hand, are installed into their final place, since they pertain to the system we want to build.

5.2. Binutils-2.42 - Pass 1

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

Approximate build time: 1 SBU
Required disk space: 663 MB

5.2.1. Installation of Cross Binutils

Note

Go back and re-read the notes in the section titled General Compilation Instructions. Understanding the notes labeled important can save you a lot of problems later.

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.

The Binutils documentation recommends building Binutils in a dedicated build directory:

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

Note

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

Now prepare Binutils for compilation:

```
../configure --prefix=$LFS/tools \
             --with-sysroot=$LFS \
             --target=$LFS_TGT   \
             --disable-nls       \
             --enable-gprofng=no \
             --disable-werror    \
             --enable-default-hash-style=gnu
```

The meaning of the configure options:

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

`--with-sysroot=$LFS`
For cross compilation, this tells the build system to look in $LFS for the target system libraries as needed.

`--target=$LFS_TGT`
Because the machine description in the `LFS_TGT` variable is slightly different than the value returned by the **config.guess** script, this switch will tell the **configure** script to adjust binutil's build system for building a cross linker.

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

`--enable-gprofng=no`
This disables building gprofng which is not needed for the temporary tools.

--disable-werror

This prevents the build from stopping in the event that there are warnings from the host's compiler.

--enable-default-hash-style=gnu

By default, the linker would generate both the GNU-style hash table and the classic ELF hash table for shared libraries and dynamically linked executables. The hash tables are only intended for a dynamic linker to perform symbol lookup. On LFS the dynamic linker (provided by the Glibc package) will always use the GNU-style hash table which is faster to query. So the classic ELF hash table is completely useless. This makes the linker only generate the GNU-style hash table by default, so we can avoid wasting time to generate the classic ELF hash table when we build the packages, or wasting disk space to store it.

Continue with compiling the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 8.19.2, “Contents of Binutils.”

5.3. GCC-13.2.0 - Pass 1

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

Approximate build time: 3.8 SBU
Required disk space: 4.1 GB

5.3.1. Installation of Cross GCC

GCC requires the GMP, MPFR and MPC packages. As these packages may not be included in your host distribution, they will be built with GCC. Unpack each package into the GCC source directory and rename the resulting directories so the GCC build procedures will automatically use them:

Note

There are frequent misunderstandings about this chapter. The procedures are the same as every other chapter, as explained earlier (Package build instructions). First, extract the gcc-13.2.0 tarball from the sources directory, and then change to the directory created. Only then should you proceed with the instructions below.

```
tar -xf ../mpfr-4.2.1.tar.xz
mv -v mpfr-4.2.1 mpfr
tar -xf ../gmp-6.3.0.tar.xz
mv -v gmp-6.3.0 gmp
tar -xf ../mpc-1.3.1.tar.gz
mv -v mpc-1.3.1 mpc
```

On x86_64 hosts, set the default directory name for 64-bit libraries to “lib”:

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' \
        -i.orig gcc/config/i386/t-linux64
 ;;
esac
```

The GCC documentation recommends building GCC in a dedicated build directory:

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

Prepare GCC for compilation:

```
../configure                   \
    --target=$LFS_TGT          \
    --prefix=$LFS/tools        \
    --with-glibc-version=2.39 \
    --with-sysroot=$LFS        \
    --with-newlib              \
    --without-headers          \
    --enable-default-pie       \
    --enable-default-ssp       \
    --disable-nls              \
    --disable-shared           \
    --disable-multilib         \
    --disable-threads          \
    --disable-libatomic        \
    --disable-libgomp          \
    --disable-libquadmath      \
    --disable-libssp           \
    --disable-libvtv           \
    --disable-libstdcxx        \
    --enable-languages=c,c++
```

The meaning of the configure options:

`--with-glibc-version=2.39`

This option specifies the version of Glibc which will be used on the target. It is not relevant to the libc of the host distro because everything compiled by pass1 GCC will run in the chroot environment, which is isolated from libc of the host distro.

`--with-newlib`

Since a working C library is not yet available, this ensures that the inhibit_libc constant is defined when building libgcc. This prevents the compiling of any code that requires libc support.

`--without-headers`

When creating a complete cross-compiler, GCC requires standard headers compatible with the target system. For our purposes these headers will not be needed. This switch prevents GCC from looking for them.

`--enable-default-pie and --enable-default-ssp`

Those switches allow GCC to compile programs with some hardening security features (more information on those in the note on PIE and SSP in chapter 8) by default. The are not strictly needed at this stage, since the compiler will only produce temporary executables. But it is cleaner to have the temporary packages be as close as possible to the final ones.

`--disable-shared`

This switch forces GCC to link its internal libraries statically. We need this because the shared libraries require Glibc, which is not yet installed on the target system.

`--disable-multilib`

On x86_64, LFS does not support a multilib configuration. This switch is harmless for x86.

`--disable-threads, --disable-libatomic, --disable-libgomp, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdcxx`

These switches disable support for threading, libatomic, libgomp, libquadmath, libssp, libvtv, and the C++ standard library respectively. These features may fail to compile when building a cross-compiler and are not necessary for the task of cross-compiling the temporary libc.

`--enable-languages=c,c++`

This option ensures that only the C and C++ compilers are built. These are the only languages needed now.

Compile GCC by running:

```
make
```

Install the package:

```
make install
```

This build of GCC has installed a couple of internal system headers. Normally one of them, `limits.h`, would in turn include the corresponding system `limits.h` header, in this case, `$LFS/usr/include/limits.h`. However, at the time of this build of GCC `$LFS/usr/include/limits.h` does not exist, so the internal header that has just been installed is a partial, self-contained file and does not include the extended features of the system header. This is adequate for building Glibc, but the full internal header will be needed later. Create a full version of the internal header using a command that is identical to what the GCC build system does in normal circumstances:

Note

The command below shows an example of nested command substitution using two methods: backquotes and a `$()` construct. It could be rewritten using the same method for both substitutions, but is shown this way to demonstrate how they can be mixed. Generally the `$()` method is preferred.

```
cd ..
cat gcc/limitx.h gcc/glimits.h gcc/limity.h > \
  `dirname $($LFS_TGT-gcc -print-libgcc-file-name)`/include/limits.h
```

Details on this package are located in Section 8.28.2, “Contents of GCC.”

5.4. Linux-6.7.4 API Headers

The Linux API Headers (in linux-6.7.4.tar.xz) expose the kernel's API for use by Glibc.

Approximate build time: less than 0.1 SBU
Required disk space: 1.5 GB

5.4.1. Installation of Linux API Headers

The Linux kernel needs to expose an Application Programming Interface (API) for the system's C library (Glibc in LFS) to use. This is done by way of sanitizing various C header files that are shipped in the Linux kernel source tarball.

Make sure there are no stale files embedded in the package:

```
make mrproper
```

Now extract the user-visible kernel headers from the source. The recommended make target “headers_install” cannot be used, because it requires rsync, which may not be available. The headers are first placed in `./usr`, then copied to the needed location.

```
make headers
find usr/include -type f ! -name '*.h' -delete
cp -rv usr/include $LFS/usr
```

5.4.2. Contents of Linux API Headers

Installed headers: /usr/include/asm/*.h, /usr/include/asm-generic/*.h, /usr/include/drm/*.h, /usr/include/linux/*.h, /usr/include/misc/*.h, /usr/include/mtd/*.h, /usr/include/rdma/*.h, /usr/include/scsi/*.h, /usr/include/sound/*.h, /usr/include/video/*.h, and /usr/include/xen/*.h
Installed directories: /usr/include/asm, /usr/include/asm-generic, /usr/include/drm, /usr/include/linux, /usr/include/misc, /usr/include/mtd, /usr/include/rdma, /usr/include/scsi, /usr/include/sound, /usr/include/video, and /usr/include/xen

Short Descriptions

`/usr/include/asm/*.h`	The Linux API ASM Headers
`/usr/include/asm-generic/*.h`	The Linux API ASM Generic Headers
`/usr/include/drm/*.h`	The Linux API DRM Headers
`/usr/include/linux/*.h`	The Linux API Linux Headers
`/usr/include/misc/*.h`	The Linux API Miscellaneous Headers
`/usr/include/mtd/*.h`	The Linux API MTD Headers
`/usr/include/rdma/*.h`	The Linux API RDMA Headers
`/usr/include/scsi/*.h`	The Linux API SCSI Headers
`/usr/include/sound/*.h`	The Linux API Sound Headers
`/usr/include/video/*.h`	The Linux API Video Headers
`/usr/include/xen/*.h`	The Linux API Xen Headers

5.5. Glibc-2.39

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

Approximate build time: 1.5 SBU
Required disk space: 846 MB

5.5.1. Installation of Glibc

First, create a symbolic link for LSB compliance. Additionally, for x86_64, create a compatibility symbolic link required for proper operation of the dynamic library loader:

```
case $(uname -m) in
    i?86)   ln -sfv ld-linux.so.2 $LFS/lib/ld-lsb.so.3
    ;;
    x86_64) ln -sfv ../lib/ld-linux-x86-64.so.2 $LFS/lib64
            ln -sfv ../lib/ld-linux-x86-64.so.2 $LFS/lib64/ld-lsb-x86-64.so.3
    ;;
esac
```

Note

The above command is correct. The **ln** command has several syntactic versions, so be sure to check **info coreutils ln** and *ln(1)* before reporting what may appear to be an error.

Some of the Glibc programs use the non-FHS-compliant `/var/db` directory to store their runtime data. Apply the following patch to make such programs store their runtime data in the FHS-compliant locations:

```
patch -Np1 -i ../glibc-2.39-fhs-1.patch
```

The Glibc documentation recommends building Glibc in a dedicated build directory:

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

Ensure that the **ldconfig** and **sln** utilities are installed into `/usr/sbin`:

```
echo "rootsbindir=/usr/sbin" > configparms
```

Next, prepare Glibc for compilation:

```
../configure                             \
      --prefix=/usr                      \
      --host=$LFS_TGT                    \
      --build=$(../scripts/config.guess) \
      --enable-kernel=4.19               \
      --with-headers=$LFS/usr/include    \
      --disable-nscd                     \
      libc_cv_slibdir=/usr/lib
```

The meaning of the configure options:

`--host=$LFS_TGT, --build=$(../scripts/config.guess)`

The combined effect of these switches is that Glibc's build system configures itself to be cross-compiled, using the cross-linker and cross-compiler in `$LFS/tools`.

`--enable-kernel=4.19`
This tells Glibc to compile the library with support for 4.19 and later Linux kernels. Workarounds for older kernels are not enabled.

`--with-headers=$LFS/usr/include`
This tells Glibc to compile itself against the headers recently installed to the $LFS/usr/include directory, so that it knows exactly what features the kernel has and can optimize itself accordingly.

`libc_cv_slibdir=/usr/lib`
This ensures that the library is installed in /usr/lib instead of the default /lib64 on 64-bit machines.

`--disable-nscd`
Do not build the name service cache daemon which is no longer used.

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.

Note

There have been reports that this package may fail when building as a “parallel make.” If that occurs, rerun the make command with the `-j1` option.

Compile the package:

```
make
```

Install the package:

Warning

If `LFS` is not properly set, and despite the recommendations, you are building as `root`, the next command will install the newly built Glibc to your host system, which will almost certainly render it unusable. So double-check that the environment is correctly set, and that you are not `root`, before running the following command.

```
make DESTDIR=$LFS install
```

The meaning of the make install option:

`DESTDIR=$LFS`
The `DESTDIR` make variable is used by almost all packages to define the location where the package should be installed. If it is not set, it defaults to the root (`/`) directory. Here we specify that the package is installed in `$LFS` , which will become the root directory in Section 7.4, “Entering the Chroot Environment”.

Fix a hard coded path to the executable loader in the **ldd** script:

```
sed '/RTLDLIST=/s@/usr@@g' -i $LFS/usr/bin/ldd
```

Caution

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

```
echo 'int main(){}' | $LFS_TGT-gcc -xc -
readelf -l a.out | grep ld-linux
```

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

```
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

Note that for 32-bit machines, the interpreter name will be `/lib/ld-linux.so.2`.

If the output is not as shown above, or there is no output at all, then something is wrong. Investigate and retrace the steps to find out where the problem is and correct it. This issue must be resolved before continuing.

Once all is well, clean up the test file:

```
rm -v a.out
```

Note

Building the packages in the next chapter will serve as an additional check that the toolchain has been built properly. If some package, especially Binutils-pass2 or GCC-pass2, fails to build, it is an indication that something has gone wrong with the preceding Binutils, GCC, or Glibc installations.

Details on this package are located in Section 8.5.3, “Contents of Glibc.”

5.6. Libstdc++ from GCC-13.2.0

Libstdc++ is the standard C++ library. It is needed to compile C++ code (part of GCC is written in C++), but we had to defer its installation when we built gcc-pass1 because Libstdc++ depends on Glibc, which was not yet available in the target directory.

Approximate build time: 0.2 SBU
Required disk space: 1.1 GB

5.6.1. Installation of Target Libstdc++

Note

Libstdc++ is part of the GCC sources. You should first unpack the GCC tarball and change to the `gcc-13.2.0` directory.

Create a separate build directory for Libstdc++ and enter it:

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

Prepare Libstdc++ for compilation:

```
../libstdc++-v3/configure           \
    --host=$LFS_TGT                 \
    --build=$(../config.guess)      \
    --prefix=/usr                   \
    --disable-multilib              \
    --disable-nls                   \
    --disable-libstdcxx-pch         \
    --with-gxx-include-dir=/tools/$LFS_TGT/include/c++/13.2.0
```

The meaning of the configure options:

`--host=...`
Specifies that the cross-compiler we have just built should be used instead of the one in `/usr/bin`.

`--disable-libstdcxx-pch`
This switch prevents the installation of precompiled include files, which are not needed at this stage.

`--with-gxx-include-dir=/tools/$LFS_TGT/include/c++/13.2.0`
This specifies the installation directory for include files. Because Libstdc++ is the standard C++ library for LFS, this directory should match the location where the C++ compiler (**$LFS_TGT-g++**) would search for the standard C++ include files. In a normal build, this information is automatically passed to the Libstdc++ **configure** options from the top level directory. In our case, this information must be explicitly given. The C++ compiler will prepend the sysroot path `$LFS` (specified when building GCC-pass1) to the include file search path, so it will actually search in `$LFS/tools/$LFS_TGT/include/c++/13.2.0`. The combination of the *`DESTDIR`* variable (in the **make install** command below) and this switch causes the headers to be installed there.

Compile Libstdc++ by running:

```
make
```

Install the library:

```
make DESTDIR=$LFS install
```

Remove the libtool archive files because they are harmful for cross-compilation:

```
rm -v $LFS/usr/lib/lib{stdc++{,exp,fs},supc++}.la
```

Details on this package are located in Section 8.28.2, “Contents of GCC.”

Chapter 6. Cross Compiling Temporary Tools

6.1. Introduction

This chapter shows how to cross-compile basic utilities using the just built cross-toolchain. Those utilities are installed into their final location, but cannot be used yet. Basic tasks still rely on the host's tools. Nevertheless, the installed libraries are used when linking.

Using the utilities will be possible in the next chapter after entering the “chroot” environment. But all the packages built in the present chapter need to be built before we do that. Therefore we cannot be independent of the host system yet.

Once again, let us recall that improper setting of `LFS` together with building as `root`, may render your computer unusable. This whole chapter must be done as user `lfs`, with the environment as described in Section 4.4, “Setting Up the Environment”.

6.2. M4-1.4.19

The M4 package contains a macro processor.

Approximate build time: 0.1 SBU
Required disk space: 31 MB

6.2.1. Installation of M4

Prepare M4 for compilation:

```
./configure --prefix=/usr   \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.12.2, “Contents of M4.”

6.3. Ncurses-6.4-20230520

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

Approximate build time: 0.3 SBU
Required disk space: 51 MB

6.3.1. Installation of Ncurses

First, ensure that **gawk** is found first during configuration:

```
sed -i s/mawk// configure
```

Then, run the following commands to build the “tic” program on the build host:

```
mkdir build
pushd build
  ../configure
  make -C include
  make -C progs tic
popd
```

Prepare Ncurses for compilation:

```
./configure --prefix=/usr                \
            --host=$LFS_TGT              \
            --build=$(./config.guess)    \
            --mandir=/usr/share/man      \
            --with-manpage-format=normal \
            --with-shared                \
            --without-normal             \
            --with-cxx-shared            \
            --without-debug              \
            --without-ada                \
            --disable-stripping          \
            --enable-widec
```

The meaning of the new configure options:

`--with-manpage-format=normal`
This prevents Ncurses from installing compressed manual pages, which may happen if the host distribution itself has compressed manual pages.

`--with-shared`
This makes Ncurses build and install shared C libraries.

`--without-normal`
This prevents Ncurses from building and installing static C libraries.

`--without-debug`
This prevents Ncurses from building and installing debug libraries.

`--with-cxx-shared`
This makes Ncurses build and install shared C++ bindings. It also prevents it building and installing static C++ bindings.

`--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 once we enter the **chroot** environment.

`--disable-stripping`
This switch prevents the building system from using the **strip** program from the host. Using host tools on cross-compiled programs can cause failure.

`--enable-widec`
This switch causes wide-character libraries (e.g., `libncursesw.so.6.4-20230520`) to be built instead of normal ones (e.g., `libncurses.so.6.4-20230520`). These wide-character libraries are usable in both multibyte and traditional 8-bit locales, while normal libraries work properly only in 8-bit locales. Wide-character and normal libraries are source-compatible, but not binary-compatible.

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS TIC_PATH=$(pwd)/build/progs/tic install
ln -sv libncursesw.so $LFS/usr/lib/libncurses.so
sed -e 's/^#if.*XOPEN.*$/#if 1/' \
    -i $LFS/usr/include/curses.h
```

The meaning of the install options:

`TIC_PATH=$(pwd)/build/progs/tic`
We need to pass the path of the newly built **tic** program that runs on the building machine, so the terminal database can be created without errors.

ln -sv libncursesw.so $LFS/usr/lib/libncurses.so
The `libncurses.so` library is needed by a few packages we will build soon. We create this symlink to use `libncursesw.so` as a replacement.

sed -e 's/^#if.*XOPEN.*$/#if 1/' ...
The header file `curses.h` contains the definition of various Ncurses data structures. With different preprocessor macro definitions two different sets of the data structure definition may be used: the 8-bit definition is compatible with `libncurses.so` and the wide-character definition is compatible with `libncursesw.so`. Since we are using `libncursesw.so` as a replacement of `libncurses.so`, edit the header file so it will always use the wide-character data structure definition compatible with `libncursesw.so`.

Details on this package are located in Section 8.29.2, “Contents of Ncurses.”

6.4. Bash-5.2.21

The Bash package contains the Bourne-Again Shell.

Approximate build time: 0.2 SBU
Required disk space: 67 MB

6.4.1. Installation of Bash

Prepare Bash for compilation:

```
./configure --prefix=/usr                      \
            --build=$(sh support/config.guess) \
            --host=$LFS_TGT                    \
            --without-bash-malloc
```

The meaning of the configure options:

`--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 DESTDIR=$LFS install
```

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

```
ln -sv bash $LFS/bin/sh
```

Details on this package are located in Section 8.35.2, “Contents of Bash.”

6.5. Coreutils-9.4

The Coreutils package contains the basic utility programs needed by every operating system.

Approximate build time: 0.3 SBU
Required disk space: 173 MB

6.5.1. Installation of Coreutils

Prepare Coreutils for compilation:

```
./configure --prefix=/usr                     \
            --host=$LFS_TGT                   \
            --build=$(build-aux/config.guess) \
            --enable-install-program=hostname \
            --enable-no-install-program=kill,uptime
```

The meaning of the configure options:

`--enable-install-program=hostname`
This enables the **hostname** binary to be built and installed – it is disabled by default but is required by the Perl test suite.

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Move programs to their final expected locations. Although this is not necessary in this temporary environment, we must do so because some programs hardcode executable locations:

```
mv -v $LFS/usr/bin/chroot              $LFS/usr/sbin
mkdir -pv $LFS/usr/share/man/man8
mv -v $LFS/usr/share/man/man1/chroot.1 $LFS/usr/share/man/man8/chroot.8
sed -i 's/"1"/"8"/'                    $LFS/usr/share/man/man8/chroot.8
```

Details on this package are located in Section 8.57.2, “Contents of Coreutils.”

6.6. Diffutils-3.10

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

Approximate build time: 0.1 SBU
Required disk space: 29 MB

6.6.1. Installation of Diffutils

Prepare Diffutils for compilation:

```
./configure --prefix=/usr   \
            --host=$LFS_TGT \
            --build=$(./build-aux/config.guess)
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.59.2, “Contents of Diffutils.”

6.7. File-5.45

The File package contains a utility for determining the type of a given file or files.

Approximate build time: 0.1 SBU
Required disk space: 37 MB

6.7.1. Installation of File

The **file** command on the build host needs to be the same version as the one we are building in order to create the signature file. Run the following commands to make a temporary copy of the **file** command:

```
mkdir build
pushd build
  ../configure --disable-bzlib      \
               --disable-libseccomp \
               --disable-xzlib      \
               --disable-zlib
  make
popd
```

The meaning of the new configure option:

`--disable-*`
The configuration script attempts to use some packages from the host distribution if the corresponding library files exist. It may cause compilation failure if a library file exists, but the corresponding header files do not. These options prevent using these unneeded capabilities from the host.

Prepare File for compilation:

```
./configure --prefix=/usr --host=$LFS_TGT --build=$(./config.guess)
```

Compile the package:

```
make FILE_COMPILE=$(pwd)/build/src/file
```

Install the package:

```
make DESTDIR=$LFS install
```

Remove the libtool archive file because it is harmful for cross compilation:

```
rm -v $LFS/usr/lib/libmagic.la
```

Details on this package are located in Section 8.10.2, “Contents of File.”

6.8. Findutils-4.9.0

The Findutils package contains programs to find files. Programs are provided to search through all the files in a directory tree and to create, maintain, and search a database (often faster than the recursive find, but unreliable unless the database has been updated recently). Findutils also supplies the **xargs** program, which can be used to run a specified command on each file selected by a search.

Approximate build time: 0.1 SBU
Required disk space: 42 MB

6.8.1. Installation of Findutils

Prepare Findutils for compilation:

```
./configure --prefix=/usr                   \
            --localstatedir=/var/lib/locate \
            --host=$LFS_TGT                 \
            --build=$(build-aux/config.guess)
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.61.2, “Contents of Findutils.”

6.9. Gawk-5.3.0

The Gawk package contains programs for manipulating text files.

Approximate build time: 0.1 SBU
Required disk space: 47 MB

6.9.1. Installation of Gawk

First, ensure some unneeded files are not installed:

```
sed -i 's/extras//' Makefile.in
```

Prepare Gawk for compilation:

```
./configure --prefix=/usr   \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.60.2, “Contents of Gawk.”

6.10. Grep-3.11

The Grep package contains programs for searching through the contents of files.

Approximate build time: 0.1 SBU
Required disk space: 27 MB

6.10.1. Installation of Grep

Prepare Grep for compilation:

```
./configure --prefix=/usr   \
            --host=$LFS_TGT \
            --build=$(./build-aux/config.guess)
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.34.2, “Contents of Grep.”

6.11. Gzip-1.13

The Gzip package contains programs for compressing and decompressing files.

Approximate build time: 0.1 SBU
Required disk space: 11 MB

6.11.1. Installation of Gzip

Prepare Gzip for compilation:

```
./configure --prefix=/usr --host=$LFS_TGT
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.64.2, “Contents of Gzip.”

6.12. Make-4.4.1

The Make package contains a program for controlling the generation of executables and other non-source files of a package from source files.

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

6.12.1. Installation of Make

Prepare Make for compilation:

```
./configure --prefix=/usr   \
            --without-guile \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

The meaning of the new configure option:

`--without-guile`
Although we are cross-compiling, configure tries to use guile from the build host if it finds it. This makes compilation fail, so this switch prevents using it.

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.68.2, “Contents of Make.”

6.13. Patch-2.7.6

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

Approximate build time: 0.1 SBU
Required disk space: 12 MB

6.13.1. Installation of Patch

Prepare Patch for compilation:

```
./configure --prefix=/usr   \
            --host=$LFS_TGT \
            --build=$(build-aux/config.guess)
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.69.2, “Contents of Patch.”

6.14. Sed-4.9

The Sed package contains a stream editor.

Approximate build time: 0.1 SBU
Required disk space: 21 MB

6.14.1. Installation of Sed

Prepare Sed for compilation:

```
./configure --prefix=/usr   \
            --host=$LFS_TGT \
            --build=$(./build-aux/config.guess)
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.30.2, “Contents of Sed.”

6.15. Tar-1.35

The Tar package provides the ability to create tar archives as well as perform various other kinds of archive manipulation. Tar can be used on previously created archives to extract files, to store additional files, or to update or list files which were already stored.

Approximate build time: 0.1 SBU
Required disk space: 42 MB

6.15.1. Installation of Tar

Prepare Tar for compilation:

```
./configure --prefix=/usr                     \
            --host=$LFS_TGT                   \
            --build=$(build-aux/config.guess)
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Details on this package are located in Section 8.70.2, “Contents of Tar.”

6.16. Xz-5.4.6

The Xz package contains programs for compressing and decompressing files. It provides capabilities for the lzma and the newer xz compression formats. Compressing text files with **xz** yields a better compression percentage than with the traditional **gzip** or **bzip2** commands.

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

6.16.1. Installation of Xz

Prepare Xz for compilation:

```
./configure --prefix=/usr                     \
            --host=$LFS_TGT                   \
            --build=$(build-aux/config.guess) \
            --disable-static                  \
            --docdir=/usr/share/doc/xz-5.4.6
```

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Remove the libtool archive file because it is harmful for cross compilation:

```
rm -v $LFS/usr/lib/liblzma.la
```

Details on this package are located in Section 8.8.2, “Contents of Xz.”

6.17. Binutils-2.42 - Pass 2

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

Approximate build time: 0.5 SBU
Required disk space: 537 MB

6.17.1. Installation of Binutils

Binutils building system relies on an shipped libtool copy to link against internal static libraries, but the libiberty and zlib copies shipped in the package do not use libtool. This inconsistency may cause produced binaries mistakenly linked against libraries from the host distro. Work around this issue:

```
sed '6009s/$add_dir//' -i ltmain.sh
```

Create a separate build directory again:

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

Prepare Binutils for compilation:

```
../configure                   \
    --prefix=/usr              \
    --build=$(../config.guess) \
    --host=$LFS_TGT            \
    --disable-nls              \
    --enable-shared            \
    --enable-gprofng=no        \
    --disable-werror           \
    --enable-64-bit-bfd        \
    --enable-default-hash-style=gnu
```

The meaning of the new configure options:

`--enable-shared`
Builds `libbfd` as a shared library.

`--enable-64-bit-bfd`
Enables 64-bit support (on hosts with smaller word sizes). This may not be needed on 64-bit systems, but it does no harm.

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

Remove the libtool archive files because they are harmful for cross compilation, and remove unnecessary static libraries:

```
rm -v $LFS/usr/lib/lib{bfd,ctf,ctf-nobfd,opcodes,sframe}.{a,la}
```

Details on this package are located in Section 8.19.2, “Contents of Binutils.”

6.18. GCC-13.2.0 - Pass 2

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

Approximate build time: 4.4 SBU
Required disk space: 4.8 GB

6.18.1. Installation of GCC

As in the first build of GCC, the GMP, MPFR, and MPC packages are required. Unpack the tarballs and move them into the required directories:

```
tar -xf ../mpfr-4.2.1.tar.xz
mv -v mpfr-4.2.1 mpfr
tar -xf ../gmp-6.3.0.tar.xz
mv -v gmp-6.3.0 gmp
tar -xf ../mpc-1.3.1.tar.gz
mv -v mpc-1.3.1 mpc
```

If building on x86_64, change the default directory name for 64-bit libraries to “lib”:

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' \
        -i.orig gcc/config/i386/t-linux64
  ;;
esac
```

Override the building rule of libgcc and libstdc++ headers, to allow building these libraries with POSIX threads support:

```
sed '/thread_header =/s/@.*@/gthr-posix.h/' \
    -i libgcc/Makefile.in libstdc++-v3/include/Makefile.in
```

Create a separate build directory again:

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

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

Now prepare GCC for compilation:

```
../configure                                       \
    --build=$(../config.guess)                     \
    --host=$LFS_TGT                                \
    --target=$LFS_TGT                              \
    LDFLAGS_FOR_TARGET=-L$PWD/$LFS_TGT/libgcc      \
    --prefix=/usr                                  \
    --with-build-sysroot=$LFS                      \
    --enable-default-pie                           \
    --enable-default-ssp                           \
    --disable-nls                                  \
    --disable-multilib                             \
    --disable-libatomic                            \
    --disable-libgomp                              \
    --disable-libquadmath                          \
    --disable-libsanitizer                         \
    --disable-libssp                               \
    --disable-libvtv                               \
    --enable-languages=c,c++
```

The meaning of the new configure options:

`--with-build-sysroot=$LFS`

Normally, using *`--host`* ensures that a cross-compiler is used for building GCC, and that compiler knows that it has to look for headers and libraries in `$LFS`. But the build system for GCC uses other tools, which are not aware of this location. This switch is needed so those tools will find the needed files in `$LFS`, and not on the host.

`--target=$LFS_TGT`

We are cross-compiling GCC, so it's impossible to build target libraries (`libgcc` and `libstdc++`) with the previously compiled GCC binaries—those binaries won't run on the host. The GCC build system will attempt to use the host's C and C++ compilers as a workaround by default. Building the GCC target libraries with a different version of GCC is not supported, so using the host's compilers may cause the build to fail. This parameter ensures the libraries are built by GCC pass 1.

`LDFLAGS_FOR_TARGET=...`

Allow `libstdc++` to use the shared `libgcc` being built in this pass, instead of the static version that was built in GCC pass 1. This is necessary to support C++ exception handling.

`--disable-libsanitizer`

Disable GCC sanitizer runtime libraries. They are not needed for the temporary installation. This switch is necessary to build GCC without `libcrypt` installed for the target. In gcc-pass1 it was implied by *`--disable-libstdcxx`*, but now we have to explicitly pass it.

Compile the package:

```
make
```

Install the package:

```
make DESTDIR=$LFS install
```

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

```
ln -sv gcc $LFS/usr/bin/cc
```

Details on this package are located in Section 8.28.2, “Contents of GCC.”

Chapter 7. Entering Chroot and Building Additional Temporary Tools

7.1. Introduction

This chapter shows how to build the last missing bits of the temporary system: the tools needed to build the various packages. Now that all circular dependencies have been resolved, a “chroot” environment, completely isolated from the host operating system (except for the running kernel), can be used for the build.

For proper operation of the isolated environment, some communication with the running kernel must be established. This is done via the so-called *Virtual Kernel File Systems*, which will be mounted before entering the chroot environment. You may want to verify that they are mounted by issuing the **findmnt** command.

Until Section 7.4, “Entering the Chroot Environment”, the commands must be run as `root`, with the `LFS` variable set. After entering chroot, all commands are run as `root`, fortunately without access to the OS of the computer you built LFS on. Be careful anyway, as it is easy to destroy the whole LFS system with bad commands.

7.2. Changing Ownership

Note

The commands in the remainder of this book must be performed while logged in as user `root` and no longer as user `lfs`. Also, double check that `$LFS` is set in `root`'s environment.

Currently, the whole directory hierarchy in `$LFS` is owned by the user `lfs`, a user that exists only on the host system. If the directories and files under `$LFS` are kept as they are, they will be 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 all the files under `$LFS`, thus exposing these files to possible malicious manipulation.

To address this issue, change the ownership of the `$LFS/*` directories to user `root` by running the following command:

```
chown -R root:root $LFS/{usr,lib,var,etc,bin,sbin,tools}
case $(uname -m) in
  x86_64) chown -R root:root $LFS/lib64 ;;
esac
```

7.3. Preparing Virtual Kernel File Systems

Applications running in userspace utilize various file systems created by the kernel to communicate with the kernel itself. These file systems are virtual: no disk space is used for them. The content of these file systems resides in memory. These file systems must be mounted in the $LFS directory tree so the applications can find them in the chroot environment.

Begin by creating the directories on which these virtual file systems will be mounted:

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

7.3.1. Mounting and Populating /dev

During a normal boot of an LFS system, the kernel automatically mounts the `devtmpfs` file system on the `/dev` directory; the kernel creates device nodes on that virtual file system during the boot process, or when a device is first detected or accessed. The udev daemon may change the ownership or permissions of the device nodes created by the kernel,

and create new device nodes or symlinks, to ease the work of distro maintainers and system administrators. (See Section 9.3.2.2, “Device Node Creation” for details.) If the host kernel supports `devtmpfs`, we can simply mount a `devtmpfs` at `$LFS/dev` and rely on the kernel to populate it.

But some host kernels lack `devtmpfs` support; these host distros use different methods to create the content of `/dev`. So the only host-agnostic way to populate the `$LFS/dev` directory is by bind mounting the host system's `/dev` directory. A bind mount is a special type of mount that makes a directory subtree or a file visible at some other location. Use the following command to do this.

```
mount -v --bind /dev $LFS/dev
```

7.3.2. Mounting Virtual Kernel File Systems

Now mount the remaining virtual kernel file systems:

```
mount -vt devpts devpts -o gid=5,mode=0620 $LFS/dev/pts
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
mount -vt tmpfs tmpfs $LFS/run
```

The meaning of the mount options for devpts:

`gid=5`

This ensures that all devpts-created device nodes are owned by group ID 5. This is the ID we will use later on for the `tty` group. We use the group ID instead of a name, since the host system might use a different ID for its `tty` group.

`mode=0620`

This ensures that all devpts-created device nodes have mode 0620 (user readable and writable, group writable). Together with the option above, this ensures that devpts will create device nodes that meet the requirements of grantpt(), meaning the Glibc **pt_chown** helper binary (which is not installed by default) is not necessary.

In some host systems, `/dev/shm` is a symbolic link to a directory, typically `/run/shm`. The /run tmpfs was mounted above so in this case only a directory needs to be created with the correct permissions.

In other host systems `/dev/shm` is a mount point for a tmpfs. In that case the mount of /dev above will only create /dev/shm as a directory in the chroot environment. In this situation we must explicitly mount a tmpfs:

```
if [ -h $LFS/dev/shm ]; then
  install -v -d -m 1777 $LFS$(realpath /dev/shm)
else
  mount -vt tmpfs -o nosuid,nodev tmpfs $LFS/dev/shm
fi
```

7.4. Entering the Chroot Environment

Now that all the packages which are required to build the rest of the needed tools are on the system, it is time to enter the chroot environment and finish installing the temporary tools. This environment will also be used to install the final system. As user `root`, run the following command to enter the environment that is, at the moment, populated with nothing but temporary tools:

```
chroot "$LFS" /usr/bin/env -i   \
    HOME=/root                  \
    TERM="$TERM"                \
    PS1='(lfs chroot) \u:\w\$ ' \
    PATH=/usr/bin:/usr/sbin     \
    MAKEFLAGS="-j$(nproc)"      \
    TESTSUITEFLAGS="-j$(nproc)" \
    /bin/bash --login
```

If you don't want to use all available logical cores, replace *$(nproc)* with the number of logical cores you want to use for building packages in this chapter and the following chapters. The test suites of some packages (notably Autoconf, Libtool, and Tar) in Chapter 8 are not affected by MAKEFLAGS, they use a TESTSUITEFLAGS environment variable instead. We set that here as well for running these test suites with multiple cores.

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

From this point on, there is no need to use the LFS variable any more because all work will be restricted to the LFS file system; the **chroot** command runs the Bash shell with the root (/) directory set to $LFS.

Notice that /tools/bin is not in the PATH. This means that the cross toolchain will no longer be used.

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

Note

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), ensure that the virtual kernel filesystems are mounted as explained in Section 7.3.1, “Mounting and Populating /dev” and Section 7.3.2, “Mounting Virtual Kernel File Systems” and enter chroot again before continuing with the installation.

7.5. Creating Directories

It is time to create the full directory structure in the LFS file system.

Note

Some of the directories mentioned in this section may have already been created earlier with explicit instructions, or when installing some packages. They are repeated below for completeness.

Create some root-level directories that are not in the limited set required in the previous chapters by issuing the following command:

```
mkdir -pv /{boot,home,mnt,opt,srv}
```

Create the required set of subdirectories below the root-level by issuing the following commands:

```
mkdir -pv /etc/{opt,sysconfig}
mkdir -pv /lib/firmware
mkdir -pv /media/{floppy,cdrom}
mkdir -pv /usr/{,local/}{include,src}
mkdir -pv /usr/local/{bin,lib,sbin}
mkdir -pv /usr/{,local/}share/{color,dict,doc,info,locale,man}
mkdir -pv /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
mkdir -pv /var/{cache,local,log,mail,opt,spool}
mkdir -pv /var/lib/{color,misc,locate}

ln -sfv /run /var/run
ln -sfv /run/lock /var/lock

install -dv -m 0750 /root
install -dv -m 1777 /tmp /var/tmp
```

Directories are, by default, created with permission mode 755, but this is not desirable everywhere. 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—just like a normal user would do with his or her own 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.

7.5.1. FHS Compliance Note

This directory tree is based on the Filesystem Hierarchy Standard (FHS) (available at *https://refspecs.linuxfoundation.org/fhs.shtml*). The FHS also specifies the optional existence of additional directories such as `/usr/local/games` and `/usr/share/games`. In LFS, we create only the directories that are really necessary. However, feel free to create more directories, if you wish.

Warning

The FHS does not mandate the existence of the directory `/usr/lib64`, and the LFS editors have decided not to use it. For the instructions in LFS and BLFS to work correctly, it is imperative that this directory be non-existent. From time to time you should verify that it does not exist, because it is easy to create it inadvertently, and this will probably break your system.

7.6. Creating Essential Files and Symlinks

Historically, Linux maintained a list of the mounted file systems in the file `/etc/mtab`. Modern kernels maintain this list internally and expose it to the user via the `/proc` filesystem. To satisfy utilities that expect to find `/etc/mtab`, create the following symbolic link:

```
ln -sv /proc/self/mounts /etc/mtab
```

Create a basic `/etc/hosts` file to be referenced in some test suites, and in one of Perl's configuration files as well:

```
cat > /etc/hosts << EOF
127.0.0.1  localhost $(hostname)
::1        localhost
EOF
```

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
bin:x:1:1:bin:/dev/null:/usr/bin/false
daemon:x:6:6:Daemon User:/dev/null:/usr/bin/false
messagebus:x:18:18:D-Bus Message Daemon User:/run/dbus:/usr/bin/false
uuidd:x:80:80:UUID Generation Daemon User:/dev/null:/usr/bin/false
nobody:x:65534:65534:Unprivileged User:/dev/null:/usr/bin/false
EOF
```

The actual password for root will be set later.

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

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:daemon
sys:x:2:
kmem:x:3:
tape:x:4:
tty: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:
cdrom:x:15:
adm:x:16:
messagebus:x:18:
input:x:24:
mail:x:34:
kvm:x:61:
uuidd:x:80:
wheel:x:97:
users:x:999:
nogroup:x:65534:
EOF
```

The created groups are not part of any standard—they are groups decided on in part by the requirements of the Udev configuration in Chapter 9, and in part by common conventions employed by a number of existing Linux distributions. In addition, some test suites rely on specific users or groups. The Linux Standard Base (LSB, available at *https://refspecs.linuxfoundation.org/lsb.shtml*) only recommends that, besides the group root with a Group ID (GID) of 0, a group bin with a GID of 1 be present. The GID of 5 is widely used for the tty group, and the number 5 is also used in /etc/fstab for the devpts filesystem. 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 ID 65534 is used by the kernel for NFS and separate user namespaces for unmapped users and groups (those exist on the NFS server or the parent user namespace, but “do not exist” on the local machine or in the separate namespace). We assign nobody and nogroup to avoid an unnamed ID. But other distros may treat this ID differently, so any portable program should not depend on this assignment.

Some tests in Chapter 8 need a regular user. We add this user here and delete this account at the end of that chapter.

```
echo "tester:x:101:101::/home/tester:/bin/bash" >> /etc/passwd
echo "tester:x:101:" >> /etc/group
install -o tester -d /home/tester
```

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

```
exec /usr/bin/bash --login
```

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

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

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/faillog` file records failed login attempts. The `/var/log/btmp` file records the bad login attempts.

Note

The `/run/utmp` file records the users that are currently logged in. This file is created dynamically in the boot scripts.

Note

The `utmp`, `wtmp`, `btmp`, and `lastlog` files use 32-bit integers for timestamps and they'll be fundamentally broken after year 2038. Many packages have stopped using them and other packages are going to stop using them. It is probably best to consider them deprecated.

7.7. Gettext-0.22.4

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.

Approximate build time: 1.1 SBU
Required disk space: 306 MB

7.7.1. Installation of Gettext

For our temporary set of tools, we only need to install three programs from Gettext.

Prepare Gettext for compilation:

```
./configure --disable-shared
```

The meaning of the configure option:

`--disable-shared`
We do not need to install any of the shared Gettext libraries at this time, therefore there is no need to build them.

Compile the package:

```
make
```

Install the **msgfmt**, **msgmerge**, and **xgettext** programs:

```
cp -v gettext-tools/src/{msgfmt,msgmerge,xgettext} /usr/bin
```

Details on this package are located in Section 8.32.2, “Contents of Gettext.”

7.8. Bison-3.8.2

The Bison package contains a parser generator.

Approximate build time: 0.2 SBU
Required disk space: 57 MB

7.8.1. Installation of Bison

Prepare Bison for compilation:

```
./configure --prefix=/usr \
            --docdir=/usr/share/doc/bison-3.8.2
```

The meaning of the new configure option:

`--docdir=/usr/share/doc/bison-3.8.2`
This tells the build system to install bison documentation into a versioned directory.

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 8.33.2, “Contents of Bison.”

7.9. Perl-5.38.2

The Perl package contains the Practical Extraction and Report Language.

Approximate build time: 0.6 SBU
Required disk space: 280 MB

7.9.1. Installation of Perl

Prepare Perl for compilation:

```
sh Configure -des                                         \
             -Dprefix=/usr                                \
             -Dvendorprefix=/usr                          \
             -Duseshrplib                                 \
             -Dprivlib=/usr/lib/perl5/5.38/core_perl      \
             -Darchlib=/usr/lib/perl5/5.38/core_perl      \
             -Dsitelib=/usr/lib/perl5/5.38/site_perl      \
             -Dsitearch=/usr/lib/perl5/5.38/site_perl     \
             -Dvendorlib=/usr/lib/perl5/5.38/vendor_perl \
             -Dvendorarch=/usr/lib/perl5/5.38/vendor_perl
```

The meaning of the Configure options:

`-des`
This is a combination of three options: -d uses defaults for all items; -e ensures completion of all tasks; -s silences non-essential output.

`-Dvendorprefix=/usr`
This ensures **perl** knows how to tell packages where they should install their Perl modules.

`-Duseshrplib`
Build `libperl` needed by some Perl modules as a shared library, instead of a static library.

`-Dprivlib`, `-Darchlib`, `-Dsitelib`, ...
These settings define where Perl looks for installed modules. The LFS editors chose to put them in a directory structure based on the MAJOR.MINOR version of Perl (5.38) which allows upgrading Perl to newer patch levels (the patch level is the last dot separated part in the full version string like 5.38.2) without reinstalling all of the modules.

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 8.42.2, “Contents of Perl.”

7.10. Python-3.12.2

The Python 3 package contains the Python development environment. It is useful for object-oriented programming, writing scripts, prototyping large programs, and developing entire applications. Python is an interpreted computer language.

Approximate build time: 0.5 SBU
Required disk space: 598 MB

7.10.1. Installation of Python

Note

There are two package files whose name starts with the “python” prefix. The one to extract from is `Python-3.12.2.tar.xz` (notice the uppercase first letter).

Prepare Python for compilation:

```
./configure --prefix=/usr   \
            --enable-shared \
            --without-ensurepip
```

The meaning of the configure option:

`--enable-shared`
This switch prevents installation of static libraries.

`--without-ensurepip`
This switch disables the Python package installer, which is not needed at this stage.

Compile the package:

```
make
```

Note

Some Python 3 modules can't be built now because the dependencies are not installed yet. For the `ssl` module, a message `Python requires a OpenSSL 1.1.1 or newer` is outputted. The message should be ignored. Just make sure the toplevel **make** command has not failed. The optional modules are not needed now and they will be built in Chapter 8.

Install the package:

```
make install
```

Details on this package are located in Section 8.51.2, “Contents of Python 3.”

7.11. Texinfo-7.1

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

Approximate build time: 0.2 SBU
Required disk space: 130 MB

7.11.1. Installation of Texinfo

Prepare Texinfo for compilation:

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

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 8.71.2, “Contents of Texinfo.”

7.12. Util-linux-2.39.3

The Util-linux package contains miscellaneous utility programs.

Approximate build time: 0.2 SBU
Required disk space: 172 MB

7.12.1. Installation of Util-linux

The FHS recommends using the `/var/lib/hwclock` directory instead of the usual `/etc` directory as the location for the `adjtime` file. Create this directory with:

```
mkdir -pv /var/lib/hwclock
```

Prepare Util-linux for compilation:

```
./configure --libdir=/usr/lib    \
            --runstatedir=/run   \
            --disable-chfn-chsh  \
            --disable-login      \
            --disable-nologin    \
            --disable-su         \
            --disable-setpriv    \
            --disable-runuser    \
            --disable-pylibmount \
            --disable-static     \
            --without-python     \
            ADJTIME_PATH=/var/lib/hwclock/adjtime \
            --docdir=/usr/share/doc/util-linux-2.39.3
```

The meaning of the configure options:

`ADJTIME_PATH=/var/lib/hwclock/adjtime`
This sets the location of the file recording information about the hardware clock in accordance to the FHS. This is not strictly needed for this temporary tool, but it prevents creating a file at another location, which would not be overwritten or removed when building the final util-linux package.

`--libdir=/usr/lib`
This switch ensures the `.so` symlinks targeting the shared library file in the same directory (`/usr/lib`) directly.

`--disable-*`
These switches prevent warnings about building components that require packages not in LFS or not installed yet.

`--without-python`
This switch disables using Python. It avoids trying to build unneeded bindings.

`runstatedir=/run`
This switch sets the location of the socket used by **uuidd** and `libuuid` correctly.

Compile the package:

```
make
```

Install the package:

```
make install
```

Details on this package are located in Section 8.78.2, “Contents of Util-linux.”

7.13. Cleaning up and Saving the Temporary System

7.13.1. Cleaning

First, remove the currently installed documentation files to prevent them from ending up in the final system, and to save about 35 MB:

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

Second, on a modern Linux system, the libtool .la files are only useful for libltdl. No libraries in LFS are loaded by libltdl, and it's known that some .la files can cause BLFS package failures. Remove those files now:

```
find /usr/{lib,libexec} -name \*.la -delete
```

The current system size is now about 3 GB, however the /tools directory is no longer needed. It uses about 1 GB of disk space. Delete it now:

```
rm -rf /tools
```

7.13.2. Backup

At this point the essential programs and libraries have been created and your current LFS system is in a good state. Your system can now be backed up for later reuse. In case of fatal failures in the subsequent chapters, it often turns out that removing everything and starting over (more carefully) is the best way to recover. Unfortunately, all the temporary files will be removed, too. To avoid spending extra time to redo something which has been done successfully, creating a backup of the current LFS system may prove useful.

Note

All the remaining steps in this section are optional. Nevertheless, as soon as you begin installing packages in Chapter 8, the temporary files will be overwritten. So it may be a good idea to do a backup of the current system as described below.

The following steps are performed from outside the chroot environment. That means you have to leave the chroot environment first before continuing. The reason for that is to get access to file system locations outside of the chroot environment to store/read the backup archive, which ought not be placed within the `$LFS` hierarchy.

If you have decided to make a backup, leave the chroot environment:

```
exit
```

Important

All of the following instructions are executed by `root` on your host system. Take extra care about the commands you're going to run as mistakes made here can modify your host system. Be aware that the environment variable `LFS` is set for user `lfs` by default but may *not* be set for `root`.

Whenever commands are to be executed by `root`, make sure you have set `LFS`.

This has been discussed in Section 2.6, “Setting The $LFS Variable”.

Before making a backup, unmount the virtual file systems:

```
mountpoint -q $LFS/dev/shm && umount $LFS/dev/shm
umount $LFS/dev/pts
umount $LFS/{sys,proc,run,dev}
```

Make sure you have at least 1 GB free disk space (the source tarballs will be included in the backup archive) on the file system containing the directory where you create the backup archive.

Note that the instructions below specify the home directory of the host system's `root` user, which is typically found on the root file system. Replace `$HOME` by a directory of your choice if you do not want to have the backup stored in `root`'s home directory.

Create the backup archive by running the following command:

Note

Because the backup archive is compressed, it takes a relatively long time (over 10 minutes) even on a reasonably fast system.

```
cd $LFS
tar -cJpf $HOME/lfs-temp-tools-12.1.tar.xz .
```

Note

If continuing to chapter 8, don't forget to reenter the chroot environment as explained in the “Important” box below.

7.13.3. Restore

In case some mistakes have been made and you need to start over, you can use this backup to restore the system and save some recovery time. Since the sources are located under `$LFS`, they are included in the backup archive as well, so they do not need to be downloaded again. After checking that `$LFS` is set properly, you can restore the backup by executing the following commands:

Warning

The following commands are extremely dangerous. If you run **rm -rf ./*** as the `root` user and you do not change to the $LFS directory or the `LFS` environment variable is not set for the `root` user, it will destroy your entire host system. YOU ARE WARNED.

```
cd $LFS
rm -rf ./*
tar -xpf $HOME/lfs-temp-tools-12.1.tar.xz
```

Again, double check that the environment has been set up properly and continue building the rest of the system.

Important

If you left the chroot environment to create a backup or restart building using a restore, remember to check that the virtual file systems are still mounted (**findmnt | grep $LFS**). If they are not mounted, remount them now as described in Section 7.3, “Preparing Virtual Kernel File Systems” and re-enter the chroot environment (see Section 7.4, “Entering the Chroot Environment”) before continuing.

Part IV. Building the LFS System

Chapter 8. Installing Basic System Software

8.1. Introduction

In this chapter, we start constructing the LFS 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 you (or the system) may need it.

We do not recommend using customized optimizations. They 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 with a customized optimization, try to compile it without optimization and see if that fixes the problem. Even if the package does compile when using a customized optimization, there is the risk it may have been compiled incorrectly because of the complex interactions between the code and the build tools. Also note that the `-march` and `-mtune` options using values not specified in the book have not been tested. This may cause problems with the toolchain packages (Binutils, GCC and Glibc). The small potential gains achieved by customizing compiler optimizations are often outweighed by the risks. First-time builders of LFS are encouraged to build without custom optimizations.

On the other hand, we keep the optimizations enabled by the default configuration of the packages. In addition, we sometimes explicitly enable an optimized configuration provided by a package but not enabled by default. The package maintainers have already tested these configurations and consider them safe, so it's not likely they would break the build. Generally the default configuration already enables `-O2` or `-O3`, so the resulting system will still run very fast without any customized optimization, and be stable at the same time.

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

Note

The SBU values and required disk space include test suite data for all applicable packages in Chapter 8. SBU values have been calculated using four CPU cores (-j4) for all operations unless specified otherwise.

8.1.1. About Libraries

In general, the LFS editors discourage building and installing static libraries. Most static libraries have been made obsolete in a modern Linux system. In addition, linking a static library into a program can be detrimental. If an update to the library is needed to remove a security problem, every program that uses the static library will need to be relinked with the new library. Since the use of static libraries is not always obvious, the relevant programs (and the procedures needed to do the linking) may not even be known.

The procedures in this chapter remove or disable installation of most static libraries. Usually this is done by passing a `--disable-static` option to **configure**. In other cases, alternate means are needed. In a few cases, especially Glibc and GCC, the use of static libraries remains an essential feature of the package building process.

For a more complete discussion of libraries, see *Libraries: Static or shared?* in the BLFS book.

8.2. Package Management

Package Management is an often requested addition to the LFS Book. A Package Manager tracks the installation of files, making it easier to remove and upgrade packages. A good package manager will also handle the configuration files specially to keep the user configuration when the package is reinstalled or upgraded. Before you begin to wonder, NO—this section will not talk about nor recommend any particular package manager. What it does provide is a roundup of the more popular techniques and how they work. The perfect package manager for you may be among these techniques, or it 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 package manager is mentioned in LFS 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. Finding one solution that satisfies all audiences is difficult.

There are some hints written on the topic of package management. Visit the *Hints Project* and see if one of them fits your needs.

8.2.1. Upgrade Issues

A Package Manager makes it easy to upgrade to newer versions when they are released. Generally the instructions in the LFS and BLFS books 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 the Linux kernel needs to be upgraded (for example, from 5.10.17 to 5.10.18 or 5.11.1), nothing else needs to be rebuilt. The system will keep working fine thanks to the well-defined interface between the kernel and userspace. Specifically, Linux API headers need not be upgraded along with the kernel. You will merely need to reboot your system to use the upgraded kernel.
- If Glibc needs to be upgraded to a newer version, (e.g., from Glibc-2.36 to Glibc-2.39), some extra steps are needed to avoid breaking the system. Read Section 8.5, “Glibc-2.39” for details.
- If a package containing a shared library is updated, and if the name of the library changes, then any packages dynamically linked to the library must 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 the name `libfoo.so.1`. Suppose you upgrade the package to a newer version foo-1.2.4 that installs a shared library with the name `libfoo.so.2`. In this case, any packages that are dynamically linked to `libfoo.so.1` need to be recompiled to link against `libfoo.so.2` in order to use the new library version. You should not remove the old libraries until all the dependent packages have been recompiled.
- If a package is (directly or indirectly) linked to both the old and new names of a shared library (for example, the package links to both `libfoo.so.2` and `libbar.so.1`, while the latter links to `libfoo.so.3`), the package may malfunction because the different revisions of the shared library present incompatible definitions for some symbol names. This can be caused by recompiling some, but not all, of the packages linked to the old shared library after the package providing the shared library is upgraded. To avoid the issue, users will need to rebuild every package linked to a shared library with an updated revision (e.g. libfoo.so.2 to libfoo.so.3) as soon as possible.
- If a package containing a shared library is updated, and the name of the library doesn't change, but the version number of the library **file** decreases (for example, the library is still named `libfoo.so.1`, but the name of the library file is changed from `libfoo.so.1.25` to `libfoo.so.1.24`), you should remove the library file from the previously

installed version (`libfoo.so.1.25` in this case). Otherwise, a **ldconfig** command (invoked by yourself from the command line, or by the installation of some package) will reset the symlink `libfoo.so.1` to point to the old library file because it seems to be a "newer" version; its version number is larger. This situation may arise if you have to downgrade a package, or if the authors change the versioning scheme for library files.

- If a package containing a shared library is updated, and the name of the library doesn't change, but a severe issue (especially, a security vulnerability) is fixed, all running programs linked to the shared library should be restarted. The following command, run as `root` after the update is complete, will list which processes are using the old versions of those libraries (replace *`libfoo`* with the name of the library):

```
grep -l 'libfoo.*deleted' /proc/*/maps | tr -cd 0-9\\n | xargs -r ps u
```

  If OpenSSH is being used to access the system and it is linked to the updated library, you must restart the **sshd** service, then logout, login again, and run the preceding command again to confirm that nothing is still using the deleted libraries.

- If an executable program or a shared library is overwritten, the processes using the code or data in that program or library may crash. The correct way to update a program or a shared library without causing the process to crash is to remove it first, then install the new version. The **install** command provided by coreutils has already implemented this, and most packages use that command to install binary files and libraries. This means that you won't be troubled by this issue most of the time. However, the install process of some packages (notably SpiderMonkey in BLFS) just overwrites the file if it exists; this causes a crash. So it's safer to save your work and close unneeded running processes before updating a package.

8.2.2. 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 each particular scheme.

8.2.2.1. It is All in My Head!

Yes, this is a package management technique. Some folks do not need a package manager because they know the packages intimately and know which files are installed by each package. Some users also do not need any package management because they plan on rebuilding the entire system whenever a package is changed.

8.2.2.2. Install in Separate Directories

This is a simplistic package management technique that does not need a special program to manage the packages. Each package is installed in a separate directory. For example, package foo-1.1 is installed in `/opt/foo-1.1` and a symlink is made from `/opt/foo` to `/opt/foo-1.1`. When a new version foo-1.2 comes along, it is installed in `/opt/foo-1.2` and the previous symlink is replaced by a symlink to the new version.

Environment variables such as `PATH`, `MANPATH`, `INFOPATH`, `PKG_CONFIG_PATH`, `CPPFLAGS`, `LDFLAGS`, and the configuration file `/etc/ld.so.conf` may need to be expanded to include the corresponding subdirectories in `/opt/foo-x.y`.

This scheme is used by the BLFS book to install some very large packages to make it easier to upgrade them. If you install more than a few packages, this scheme becomes unmanageable. And some packages (for example Linux API headers and Glibc) may not work well with this scheme. **Never use this scheme system-wide.**

8.2.2.3. Symlink Style Package Management

This is a variation of the previous package management technique. Each package is installed as in the previous scheme. But instead of making the symlink via a generic package name, 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, many package managers use this approach, and automate the creation of the symlinks. A few of the popular ones include Stow, Epkg, Graft, and Depot.

The installation script needs to be fooled, so the package thinks 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, suppose 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` variable to direct the installation. 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 install the package manually, or you may find that it is easier to install some problematic packages into `/opt`.

8.2.2.4. 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 that uses 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 packages are installed one at a time. The logs are not reliable if two packages are installed simultaneously from two different consoles.

8.2.2.5. Tracing Installation Scripts

In this approach, the commands that the installation scripts perform are recorded. There are two techniques that one can use:

The `LD_PRELOAD` environment variable can be set to point to a library to be 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's a good idea to perform some tests to ensure that the package manager does not break anything, and that it logs all the appropriate files.

Another technique is to use **strace**, which logs all the system calls made during the execution of the installation scripts.

8.2.2.6. Creating Package Archives

In this scheme, the package installation is faked into a separate tree as previously described in the symlink style package management section. After the installation, a package archive is created using the installed files. This archive is then used to install the package on the local machine or even 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 LFS systems is located at *https://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt*.

The creation of package files that include dependency information is complex, and beyond the scope of LFS.

Slackware uses a **tar**-based system for package archives. This system purposely does not handle package dependencies as more complex package managers do. For details of Slackware package management, see *https://www.slackbook.org/html/package-management.html*.

8.2.2.7. User Based Management

This scheme, unique to LFS, was devised by Matthias Benkmann, and is available from the *Hints Project*. In this scheme, each package is installed as a separate user into the standard locations. Files belonging to a package are easily identified by checking the user ID. The features and shortcomings of this approach are too complex to describe in this section. For the details please see the hint at *https://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt*.

8.2.3. Deploying LFS on Multiple Systems

One of the advantages of an LFS system is that there are no files that depend on the position of files on a disk system. Cloning an LFS build to another computer with the same architecture as the base system is as simple as using **tar** on the LFS partition that contains the root directory (about 900MB uncompressed for a basic LFS build), copying that file via network transfer or CD-ROM / USB stick to the new system, and expanding it. After that, a few configuration files will have to be changed. Configuration files that may need to be updated include: `/etc/hosts`, `/etc/fstab`, `/etc/passwd`, `/etc/group`, `/etc/shadow`, `/etc/ld.so.conf`, `/etc/sysconfig/rc.site`, `/etc/sysconfig/network`, and `/etc/sysconfig/ifconfig.eth0`.

A custom kernel may be needed for the new system, depending on differences in system hardware and the original kernel configuration.

Note

There have been some reports of issues when copying between similar but not identical architectures. For instance, the instruction set for an Intel system is not identical with the AMD processor's instructions, and later versions of some processors may provide instructions that are unavailable with earlier versions.

Finally, the new system has to be made bootable via Section 10.4, “Using GRUB to Set Up the Boot Process”.

8.3. Man-pages-6.06

The Man-pages package contains over 2,400 man pages.

Approximate build time: less than 0.1 SBU
Required disk space: 33 MB

8.3.1. Installation of Man-pages

Remove two man pages for password hashing functions. Libxcrypt will provide a better version of these man pages:

```
rm -v man3/crypt*
```

Install Man-pages by running:

```
make prefix=/usr install
```

8.3.2. Contents of Man-pages

Installed files: various man pages

Short Descriptions

`man pages` Describe C programming language functions, important device files, and significant configuration files

8.4. Iana-Etc-20240125

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

Approximate build time: less than 0.1 SBU
Required disk space: 4.8 MB

8.4.1. Installation of Iana-Etc

For this package, we only need to copy the files into place:

```
cp services protocols /etc
```

8.4.2. 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

8.5. Glibc-2.39

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

Approximate build time: 12 SBU
Required disk space: 3.1 GB

8.5.1. Installation of Glibc

Some of the Glibc programs use the non-FHS compliant `/var/db` directory to store their runtime data. Apply the following patch to make such programs store their runtime data in the FHS-compliant locations:

```
patch -Np1 -i ../glibc-2.39-fhs-1.patch
```

The Glibc documentation recommends building Glibc in a dedicated build directory:

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

Ensure that the **ldconfig** and **sln** utilities will be installed into `/usr/sbin`:

```
echo "rootsbindir=/usr/sbin" > configparms
```

Prepare Glibc for compilation:

```
../configure --prefix=/usr                            \
             --disable-werror                         \
             --enable-kernel=4.19                     \
             --enable-stack-protector=strong          \
             --disable-nscd                           \
             libc_cv_slibdir=/usr/lib
```

The meaning of the configure options:

`--disable-werror`
This option disables the -Werror option passed to GCC. This is necessary for running the test suite.

`--enable-kernel=4.19`
This option tells the build system that this Glibc may be used with kernels as old as 4.19. This means generating workarounds in case a system call introduced in a later version cannot be used.

`--enable-stack-protector=strong`
This option increases system security by adding extra code to check for buffer overflows, such as stack smashing attacks. Note that Glibc always explicitly overrides the default of GCC, so this option is still needed even though we've already specified `--enable-default-ssp` for GCC.

`--disable-nscd`
Do not build the name service cache daemon which is no longer used.

`libc_cv_slibdir=/usr/lib`
This variable sets the correct library for all systems. We do not want lib64 to be used.

Compile the package:

```
make
```

Important

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

Generally a few tests do not pass. The test failures listed below are usually safe to ignore.

```
make check
```

You may see some test failures. The Glibc test suite is somewhat dependent on the host system. A few failures out of over 5000 tests can generally be ignored. This is a list of the most common issues seen for recent versions of LFS:

- *io/tst-lchmod* is known to fail in the LFS chroot environment.
- Some tests, for example *nss/tst-nss-files-hosts-multi* and *nptl/tst-thread-affinity** are known to fail due to a timeout (especially when the system is relatively slow and/or running the test suite with multiple parallel make jobs). These tests can be identified with:

  ```
  grep "Timed out" -l $(find -name \*.out)
  ```

  It's possible to re-run a single test with enlarged timeout with **TIMEOUTFACTOR=<*factor*> make test t=<*test name*>**. For example, **TIMEOUTFACTOR=10 make test t=nss/tst-nss-files-hosts-multi** will re-run *nss/tst-nss-files-hosts-multi* with ten times the original timeout.
- Additionally, some tests may fail with a relatively old CPU model (for example *elf/tst-cpu-features-cpuinfo*) or host kernel version (for example *stdlib/tst-arc4random-thread*).

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
```

Fix the Makefile to skip an outdated sanity check that fails with a modern Glibc configuration:

```
sed '/test-installation/s@$(PERL)@echo not running@' -i ../Makefile
```

Important

If upgrading Glibc to a new minor version (for example, from Glibc-2.36 to Glibc-2.39) on a running LFS system, you need to take some extra precautions to avoid breaking the system:

- Upgrading Glibc on a LFS system prior to 11.0 (exclusive) is not supported. Rebuild LFS if you are running such an old LFS system but you need a newer Glibc.
- If upgrading on a LFS system prior to 12.0 (exclusive), install Libxcrypt following Section 8.26, “Libxcrypt-4.4.36.” In addition to a normal Libxcrypt installation, **you MUST follow the note in Libxcrypt section to install `libcrypt.so.1*` (replacing `libcrypt.so.1` from the prior Glibc installation)**.
- If upgrading on a LFS system prior to 12.1 (exclusive), remove the **nscd** program:

```
rm -f /usr/sbin/nscd
```

- Upgrade the kernel and reboot if it's older than 4.19 (check the current version with **uname -r**) or if you want to upgrade it anyway, following Section 10.3, “Linux-6.7.4.”
- Upgrade the kernel API headers if it's older than 4.19 (check the current version with **cat /usr/include/linux/version.h**) or if you want to upgrade it anyway, following Section 5.4, “Linux-6.7.4 API Headers” (but removing `$LFS` from the **cp** command).
- Perform a `DESTDIR` installation and upgrade the Glibc shared libraries on the system using one single **install** command:

```
make DESTDIR=$PWD/dest install
install -vm755 dest/usr/lib/*.so.* /usr/lib
```

It's imperative to strictly follow these steps above unless you completely understand what you are doing. **Any unexpected deviation may render the system completely unusable. YOU ARE WARNED.**

Then continue to run the **make install** command, the **sed** command against `/usr/bin/ldd`, and the commands to install the locales. Once they are finished, reboot the system immediately.

Install the package:

```
make install
```

Fix a hardcoded path to the executable loader in the **ldd** script:

```
sed '/RTLDLIST=/s@/usr@@g' -i /usr/bin/ldd
```

Next, install the locales that can make the system respond in a different language. None of these locales are required, but if some of them are missing, the test suites of some packages will skip important test cases.

Individual locales can be installed using the **localedef** program. E.g., the second **localedef** command below combines the `/usr/share/i18n/locales/cs_CZ` charset-independent locale definition with the `/usr/share/i18n/charmaps/UTF-8.gz` charmap definition and appends the result to the `/usr/lib/locale/locale-archive` file. The following instructions will install the minimum set of locales necessary for the optimal coverage of tests:

```
mkdir -pv /usr/lib/locale
localedef -i C -f UTF-8 C.UTF-8
localedef -i cs_CZ -f UTF-8 cs_CZ.UTF-8
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i de_DE -f UTF-8 de_DE.UTF-8
localedef -i el_GR -f ISO-8859-7 el_GR
localedef -i en_GB -f ISO-8859-1 en_GB
localedef -i en_GB -f UTF-8 en_GB.UTF-8
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 en_US -f UTF-8 en_US.UTF-8
localedef -i es_ES -f ISO-8859-15 es_ES@euro
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 fr_FR -f UTF-8 fr_FR.UTF-8
localedef -i is_IS -f ISO-8859-1 is_IS
localedef -i is_IS -f UTF-8 is_IS.UTF-8
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i it_IT -f ISO-8859-15 it_IT@euro
localedef -i it_IT -f UTF-8 it_IT.UTF-8
localedef -i ja_JP -f EUC-JP ja_JP
localedef -i ja_JP -f SHIFT_JIS ja_JP.SJIS 2> /dev/null || true
localedef -i ja_JP -f UTF-8 ja_JP.UTF-8
localedef -i nl_NL@euro -f ISO-8859-15 nl_NL@euro
localedef -i ru_RU -f KOI8-R ru_RU.KOI8-R
localedef -i ru_RU -f UTF-8 ru_RU.UTF-8
localedef -i se_NO -f UTF-8 se_NO.UTF-8
localedef -i ta_IN -f UTF-8 ta_IN.UTF-8
localedef -i tr_TR -f UTF-8 tr_TR.UTF-8
localedef -i zh_CN -f GB18030 zh_CN.GB18030
localedef -i zh_HK -f BIG5-HKSCS zh_HK.BIG5-HKSCS
localedef -i zh_TW -f UTF-8 zh_TW.UTF-8
```

In addition, install the locale for your own country, language and character set.

Alternatively, install all the locales listed in the `glibc-2.39/localedata/SUPPORTED` file (it includes every locale listed above and many more) at once with the following time-consuming command:

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

Then use the **localedef** command to create and install locales not listed in the `glibc-2.39/localedata/SUPPORTED` file when you need them. For instance, the following two locales are needed for some tests later in this chapter:

```
localedef -i C -f UTF-8 C.UTF-8
localedef -i ja_JP -f SHIFT_JIS ja_JP.SJIS 2> /dev/null || true
```

Note

Glibc now uses libidn2 when resolving internationalized domain names. This is a run time dependency. If this capability is needed, the instructions for installing libidn2 are in the *BLFS libidn2 page*.

8.5.2. Configuring Glibc

8.5.2.1. Adding nsswitch.conf

The `/etc/nsswitch.conf` file needs to be created because the Glibc defaults do not work well in a networked environment.

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
```

8.5.2.2. Adding Time Zone Data

Install and set up the time zone data with the following:

```
tar -xf ../../tzdata2024a.tar.gz

ZONEINFO=/usr/share/zoneinfo
mkdir -pv $ZONEINFO/{posix,right}

for tz in etcetera southamerica northamerica europe africa antarctica  \
          asia australasia backward; do
    zic -L /dev/null   -d $ZONEINFO       ${tz}
    zic -L /dev/null   -d $ZONEINFO/posix ${tz}
    zic -L leapseconds -d $ZONEINFO/right ${tz}
done

cp -v zone.tab zone1970.tab iso3166.tab $ZONEINFO
zic -d $ZONEINFO -p America/New_York
unset ZONEINFO
```

The meaning of the zic commands:

`zic -L /dev/null ...`

This creates posix time zones without any leap seconds. It is conventional to put these in both `zoneinfo` and `zoneinfo/posix`. It is necessary to put the POSIX time zones in `zoneinfo`, otherwise various test suites will report errors. On an embedded system, where space is tight and you do not intend to ever update the time zones, you could save 1.9 MB by not using the `posix` directory, but some applications or test suites might produce some failures.

`zic -L leapseconds ...`

This creates right time zones, including leap seconds. On an embedded system, where space is tight and you do not intend to ever update the time zones, or care about the correct time, you could save 1.9MB by omitting the `right` directory.

zic ... -p ...

This creates the `posixrules` file. We use New York because POSIX requires the daylight saving time rules to be in accordance with US rules.

One way to determine the local time zone is to 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., *America/Edmonton*). There are also some other possible time zones listed in `/usr/share/zoneinfo` such as *Canada/Eastern* or *EST5EDT* that are not identified by the script but can be used.

Then create the `/etc/localtime` file by running:

```
ln -sfv /usr/share/zoneinfo/<xxx> /etc/localtime
```

Replace *<xxx>* with the name of the time zone selected (e.g., Canada/Eastern).

8.5.2.3. Configuring the Dynamic Loader

By default, the dynamic loader (`/lib/ld-linux.so.2`) searches through `/usr/lib` for dynamic libraries that are needed by programs as they are run. However, if there are libraries in directories other than `/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

EOF
```

If desired, the dynamic loader can also search a directory and include the contents of files found there. Generally the files in this include directory are one line specifying the desired library path. To add this capability run the following commands:

```
cat >> /etc/ld.so.conf << "EOF"
# Add an include directory
include /etc/ld.so.conf.d/*.conf

EOF
mkdir -pv /etc/ld.so.conf.d
```

8.5.3. Contents of Glibc

Installed programs: gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, ld.so (symlink to ld-linux-x86-64.so.2 or ld-linux.so.2), locale, localedef, makedb, mtrace, pcprofiledump, pldd, sln, sotruss, sprof, tzselect, xtrace, zdump, and zic

Installed libraries: ld-linux-x86-64.so.2, ld-linux.so.2, libBrokenLocale.{a,so}, libanl.{a,so}, libc.{a,so}, libc_nonshared.a, libc_malloc_debug.so, libdl.{a,so.2}, libg.a, libm.{a,so}, libmcheck.a, libmemusage.so, libmvec.{a,so}, libnsl.so.1, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libpcprofile.so, libpthread.{a,so.0}, libresolv.{a,so}, librt.{a,so.1}, libthread_db.so, and libutil.{a,so.1}

Installed directories: /usr/include/arpa, /usr/include/bits, /usr/include/gnu, /usr/include/net, /usr/include/netash, /usr/include/netatalk, /usr/include/netax25, /usr/include/neteconet, /usr/include/netinet, /usr/include/netipx, /usr/include/netiucv, /usr/include/netpacket, /usr/include/netrom, /usr/include/netrose, /usr/include/nfs, /usr/include/protocols, /usr/include/rpc, /usr/include/sys, /usr/lib/audit, /usr/lib/gconv, /usr/lib/locale, /usr/libexec/getconf, /usr/share/i18n, /usr/share/zoneinfo, and /var/lib/nss_db

Short Descriptions

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. It does not exist on newer architectures like x86_64
locale	Prints various information about the current locale
localedef	Compiles locale specifications
makedb	Creates a simple database from textual input
mtrace	Reads and interprets a memory trace file and displays a summary in human-readable format
pcprofiledump	Dump information generated by PC profiling
pldd	Lists dynamic shared objects used by running processes
sln	A statically linked **ln** program
sotruss	Traces shared library procedure calls of a specified command
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 internally by Glibc as a gross hack to get broken programs (e.g., some Motif applications) running. See comments in `glibc-2.39/locale/broken_cur_max.c` for more information
`libanl`	Dummy library containing no functions. Previously was the asynchronous name lookup library, whose functions are now in `libc`
`libc`	The main C library
`libc_malloc_debug`	Turns on memory allocation checking when preloaded
`libdl`	Dummy library containing no functions. Previously was the dynamic linking interface library, whose functions are now in `libc`
`libg`	Dummy library containing no functions. Previously was a runtime library for **g++**
`libm`	The mathematical library
`libmvec`	The vector math library, linked in as needed when `libm` is used
`libmcheck`	Turns on memory allocation checking when linked to
`libmemusage`	Used by **memusage** to help collect information about the memory usage of a program
`libnsl`	The network services library, now deprecated
`libnss_*`	The Name Service Switch modules, containing functions for resolving host names, user names, group names, aliases, services, protocols, etc. Loaded by `libc` according to the configuration in `/etc/nsswitch.conf`
`libpcprofile`	Can be preloaded to PC profile an executable
`libpthread`	Dummy library containing no functions. Previously contained functions providing most of the interfaces specified by the POSIX.1c Threads Extensions and the semaphore interfaces specified by the POSIX.1b Real-time Extensions, now the functions are in `libc`
`libresolv`	Contains functions for creating, sending, and interpreting packets to the Internet domain name servers
`librt`	Contains functions providing most of the interfaces specified by the POSIX.1b Real-time Extensions
`libthread_db`	Contains functions useful for building debuggers for multi-threaded programs
`libutil`	Dummy library containing no functions. Previously contained code for “standard” functions used in many different Unix utilities. These functions are now in `libc`

8.6. Zlib-1.3.1

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

Approximate build time: less than 0.1 SBU
Required disk space: 6.4 MB

8.6.1. Installation of Zlib

Prepare Zlib for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

Remove a useless static library:

```
rm -fv /usr/lib/libz.a
```

8.6.2. Contents of Zlib

Installed libraries: libz.so

Short Descriptions

`libz` Contains compression and decompression functions used by some programs

8.7. Bzip2-1.0.8

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**.

Approximate build time: less than 0.1 SBU
Required disk space: 7.2 MB

8.7.1. Installation of Bzip2

Apply a patch that will install the documentation for this package:

```
patch -Np1 -i ../bzip2-1.0.8-install_docs-1.patch
```

The following command ensures installation of symbolic links are relative:

```
sed -i 's@\(ln -s -f \)$(PREFIX)/bin/@\1@' Makefile
```

Ensure the man pages are installed into the correct location:

```
sed -i "s@(PREFIX)/man@(PREFIX)/share/man@g" Makefile
```

Prepare Bzip2 for compilation with:

```
make -f Makefile-libbz2_so
make clean
```

The meaning of the make parameter:

`-f Makefile-libbz2_so`

This will cause Bzip2 to be built using a different `Makefile` file, in this case the `Makefile-libbz2_so` file, which creates a dynamic `libbz2.so` library and links the Bzip2 utilities against it.

Compile and test the package:

```
make
```

Install the programs:

```
make PREFIX=/usr install
```

Install the shared library:

```
cp -av libbz2.so.* /usr/lib
ln -sv libbz2.so.1.0.8 /usr/lib/libbz2.so
```

Install the shared **bzip2** binary into the `/usr/bin` directory, and replace two copies of **bzip2** with symlinks:

```
cp -v bzip2-shared /usr/bin/bzip2
for i in /usr/bin/{bzcat,bunzip2}; do
  ln -sfv bzip2 $i
done
```

Remove a useless static library:

```
rm -fv /usr/lib/libbz2.a
```

8.7.2. Contents of Bzip2

Installed programs: bunzip2 (link to bzip2), bzcat (link to bzip2), bzcmp (link to bzdiff), bzdiff, bzegrep (link to bzgrep), bzfgrep (link to bzgrep), bzgrep, bzip2, bzip2recover, bzless (link to bzmore), and bzmore
Installed libraries: libbz2.so
Installed directory: /usr/share/doc/bzip2-1.0.8

Short Descriptions

bunzip2 Decompresses bzipped files

bzcat Decompresses to standard output

bzcmp Runs **cmp** on bzipped files

bzdiff Runs **diff** on bzipped files

bzegrep Runs **egrep** on bzipped files

bzfgrep Runs **fgrep** on bzipped files

bzgrep Runs **grep** 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

8.8. Xz-5.4.6

The Xz package contains programs for compressing and decompressing files. It provides capabilities for the lzma and the newer xz compression formats. Compressing text files with **xz** yields a better compression percentage than with the traditional **gzip** or **bzip2** commands.

Approximate build time: 0.1 SBU
Required disk space: 25 MB

8.8.1. Installation of Xz

Prepare Xz for compilation with:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/xz-5.4.6
```

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.8.2. Contents of Xz

Installed programs: lzcat (link to xz), lzcmp (link to xzdiff), lzdiff (link to xzdiff), lzegrep (link to xzgrep), lzfgrep (link to xzgrep), lzgrep (link to xzgrep), lzless (link to xzless), lzma (link to xz), lzmadec, lzmainfo, lzmore (link to xzmore), unlzma (link to xz), unxz (link to xz), xz, xzcat (link to xz), xzcmp (link to xzdiff), xzdec, xzdiff, xzegrep (link to xzgrep), xzfgrep (link to xzgrep), xzgrep, xzless, and xzmore
Installed libraries: liblzma.so
Installed directories: /usr/include/lzma and /usr/share/doc/xz-5.4.6

Short Descriptions

lzcat Decompresses to standard output
lzcmp Runs **cmp** on LZMA compressed files
lzdiff Runs **diff** on LZMA compressed files
lzegrep Runs **egrep** on LZMA compressed files
lzfgrep Runs **fgrep** on LZMA compressed files
lzgrep Runs **grep** on LZMA compressed files
lzless Runs **less** on LZMA compressed files
lzma Compresses or decompresses files using the LZMA format
lzmadec A small and fast decoder for LZMA compressed files
lzmainfo Shows information stored in the LZMA compressed file header

lzmore	Runs **more** on LZMA compressed files
unlzma	Decompresses files using the LZMA format
unxz	Decompresses files using the XZ format
xz	Compresses or decompresses files using the XZ format
xzcat	Decompresses to standard output
xzcmp	Runs **cmp** on XZ compressed files
xzdec	A small and fast decoder for XZ compressed files
xzdiff	Runs **diff** on XZ compressed files
xzegrep	Runs **egrep** on XZ compressed files
xzfgrep	Runs **fgrep** on XZ compressed files
xzgrep	Runs **grep** on XZ compressed files
xzless	Runs **less** on XZ compressed files
xzmore	Runs **more** on XZ compressed files
`liblzma`	The library implementing lossless, block-sorting data compression, using the Lempel-Ziv-Markov chain algorithm

8.9. Zstd-1.5.5

Zstandard is a real-time compression algorithm, providing high compression ratios. It offers a very wide range of compression / speed trade-offs, while being backed by a very fast decoder.

Approximate build time: 0.5 SBU
Required disk space: 77 MB

8.9.1. Installation of Zstd

Compile the package:

```
make prefix=/usr
```

Note

In the test output there are several places that indicate 'failed'. These are expected and only 'FAIL' is an actual test failure. There should be no test failures.

To test the results, issue:

```
make check
```

Install the package:

```
make prefix=/usr install
```

Remove the static library:

```
rm -v /usr/lib/libzstd.a
```

8.9.2. Contents of Zstd

Installed programs: zstd, zstdcat (link to zstd), zstdgrep, zstdless, zstdmt (link to zstd), and unzstd (link to zstd)
Installed library: libzstd.so

Short Descriptions

zstd Compresses or decompresses files using the ZSTD format

zstdgrep Runs **grep** on ZSTD compressed files

zstdless Runs **less** on ZSTD compressed files

`libzstd` The library implementing lossless data compression, using the ZSTD algorithm

8.10. File-5.45

The File package contains a utility for determining the type of a given file or files.

Approximate build time: less than 0.1 SBU
Required disk space: 17 MB

8.10.1. Installation of File

Prepare File for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.10.2. Contents of File

Installed programs: file
Installed library: libmagic.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

8.11. Readline-8.2

The Readline package is a set of libraries that offer command-line editing and history capabilities.

Approximate build time: less than 0.1 SBU
Required disk space: 16 MB

8.11.1. Installation of Readline

Reinstalling Readline will cause the old libraries to be moved to <libraryname>.old. While this is normally not a problem, in some cases it can trigger a linking bug in **ldconfig**. This can be avoided by issuing the following two seds:

```
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
```

Now fix a problem identified upstream:

```
patch -Np1 -i ../readline-8.2-upstream_fixes-3.patch
```

Prepare Readline for compilation:

```
./configure --prefix=/usr    \
            --disable-static \
            --with-curses    \
            --docdir=/usr/share/doc/readline-8.2
```

The meaning of the new configure option:

`--with-curses`
This option tells Readline that it can find the termcap library functions in the curses library, not a separate termcap library. This will generate the correct `readline.pc` file.

Compile the package:

```
make SHLIB_LIBS="-lncursesw"
```

The meaning of the make option:

`SHLIB_LIBS="-lncursesw"`
This option forces Readline to link against the `libncursesw` library.

This package does not come with a test suite.

Install the package:

```
make SHLIB_LIBS="-lncursesw" install
```

If desired, install the documentation:

```
install -v -m644 doc/*.{ps,pdf,html,dvi} /usr/share/doc/readline-8.2
```

8.11.2. Contents of Readline

Installed libraries: libhistory.so and libreadline.so
Installed directories: /usr/include/readline and /usr/share/doc/readline-8.2

Short Descriptions

`libhistory` Provides a consistent user interface for recalling lines of history

`libreadline`	Provides a set of commands for manipulating text entered in an interactive session of a program

8.12. M4-1.4.19

The M4 package contains a macro processor.

Approximate build time: 0.3 SBU
Required disk space: 49 MB

8.12.1. 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
```

8.12.2. 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

8.13. Bc-6.7.5

The Bc package contains an arbitrary precision numeric processing language.

Approximate build time: less than 0.1 SBU
Required disk space: 7.8 MB

8.13.1. Installation of Bc

Prepare Bc for compilation:

```
CC=gcc ./configure --prefix=/usr -G -O3 -r
```

The meaning of the configure options:

CC=gcc
This parameter specifies the compiler to use.

-G
Omit parts of the test suite that won't work until the bc program has been installed.

-O3
Specify the optimization to use.

-r
Enable the use of Readline to improve the line editing feature of bc.

Compile the package:

```
make
```

To test bc, run:

```
make test
```

Install the package:

```
make install
```

8.13.2. Contents of Bc

Installed programs: bc and dc

Short Descriptions

bc A command line calculator

dc A reverse-polish command line calculator

8.14. Flex-2.6.4

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

Approximate build time: 0.1 SBU
Required disk space: 33 MB

8.14.1. Installation of Flex

Prepare Flex for compilation:

```
./configure --prefix=/usr \
            --docdir=/usr/share/doc/flex-2.6.4 \
            --disable-static
```

Compile the package:

```
make
```

To test the results (about 0.5 SBU), issue:

```
make check
```

Install the package:

```
make install
```

A few programs do not know about **flex** yet and try to run its predecessor, **lex**. To support those programs, create a symbolic link named `lex` that runs `flex` in **lex** emulation mode, and also create the man page of **lex** as a symlink:

```
ln -sv flex   /usr/bin/lex
ln -sv flex.1 /usr/share/man/man1/lex.1
```

8.14.2. Contents of Flex

Installed programs: flex, flex++ (link to flex), and lex (link to flex)
Installed libraries: libfl.so
Installed directory: /usr/share/doc/flex-2.6.4

Short Descriptions

flex A tool for generating programs that recognize patterns in text; it allows for the versatility to specify the rules for pattern-finding, eradicating the need to develop a specialized program

flex++ An extension of flex, is used for generating C++ code and classes. It is a symbolic link to **flex**

lex A symbolic link that runs **flex** in **lex** emulation mode

`libfl` The `flex` library

8.15. Tcl-8.6.13

The Tcl package contains the Tool Command Language, a robust general-purpose scripting language. The Expect package is written in Tcl (pronounced "tickle").

Approximate build time: 2.7 SBU
Required disk space: 89 MB

8.15.1. Installation of Tcl

This package and the next two (Expect and DejaGNU) are installed to support running the test suites for Binutils, GCC and other packages. 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.

Prepare Tcl for compilation:

```
SRCDIR=$(pwd)
cd unix
./configure --prefix=/usr           \
            --mandir=/usr/share/man
```

Build the package:

```
make

sed -e "s|$SRCDIR/unix|/usr/lib|" \
    -e "s|$SRCDIR|/usr/include|"  \
    -i tclConfig.sh

sed -e "s|$SRCDIR/unix/pkgs/tdbc1.1.5|/usr/lib/tdbc1.1.5|" \
    -e "s|$SRCDIR/pkgs/tdbc1.1.5/generic|/usr/include|"    \
    -e "s|$SRCDIR/pkgs/tdbc1.1.5/library|/usr/lib/tcl8.6|" \
    -e "s|$SRCDIR/pkgs/tdbc1.1.5|/usr/include|"            \
    -i pkgs/tdbc1.1.5/tdbcConfig.sh

sed -e "s|$SRCDIR/unix/pkgs/itcl4.2.3|/usr/lib/itcl4.2.3|" \
    -e "s|$SRCDIR/pkgs/itcl4.2.3/generic|/usr/include|"    \
    -e "s|$SRCDIR/pkgs/itcl4.2.3|/usr/include|"            \
    -i pkgs/itcl4.2.3/itclConfig.sh

unset SRCDIR
```

The various “sed” instructions after the “make” command remove references to the build directory from the configuration files and replace them with the install directory. This is not mandatory for the remainder of LFS, but may be needed if a package built later uses Tcl.

To test the results, issue:

```
make test
```

Install the package:

```
make install
```

Make the installed library writable so debugging symbols can be removed later:

```
chmod -v u+w /usr/lib/libtcl8.6.so
```

Install Tcl's headers. The next package, Expect, requires them.

```
make install-private-headers
```

Now make a necessary symbolic link:

```
ln -sfv tclsh8.6 /usr/bin/tclsh
```

Rename a man page that conflicts with a Perl man page:

```
mv /usr/share/man/man3/{Thread,Tcl_Thread}.3
```

Optionally, install the documentation by issuing the following commands:

```
cd ..
tar -xf ../tcl8.6.13-html.tar.gz --strip-components=1
mkdir -v -p /usr/share/doc/tcl-8.6.13
cp -v -r  ./html/* /usr/share/doc/tcl-8.6.13
```

8.15.2. Contents of Tcl

Installed programs: tclsh (link to tclsh8.6) and tclsh8.6
Installed library: libtcl8.6.so and libtclstub8.6.a

Short Descriptions

tclsh8.6 The Tcl command shell

tclsh A link to tclsh8.6

`libtcl8.6.so` The Tcl library

`libtclstub8.6.a` The Tcl Stub library

8.16. Expect-5.45.4

The Expect package contains tools for automating, via scripted dialogues, interactive applications such as **telnet**, **ftp**, **passwd**, **fsck**, **rlogin**, and **tip**. Expect is also useful for testing these same applications as well as easing all sorts of tasks that are prohibitively difficult with anything else. The DejaGnu framework is written in Expect.

Approximate build time: 0.2 SBU
Required disk space: 3.9 MB

8.16.1. Installation of Expect

Expect needs PTYs to work. Verify that the PTYs are working properly inside the chroot environment by performing a simple test:

```
python3 -c 'from pty import spawn; spawn(["echo", "ok"])'
```

This command should output `ok`. If, instead, the output includes `OSError: out of pty devices`, then the environment is not set up for proper PTY operation. You need to exit from the chroot environment, read Section 7.3, “Preparing Virtual Kernel File Systems” again, and ensure the `devpts` file system (and other virtual kernel file systems) mounted correctly. Then reenter the chroot environment following Section 7.4, “Entering the Chroot Environment”. This issue needs to be resolved before continuing, or the test suites requiring Expect (for example the test suites of Bash, Binutils, GCC, GDBM, and of course Expect itself) will fail catastrophically, and other subtle breakages may also happen.

Prepare Expect for compilation:

```
./configure --prefix=/usr           \
            --with-tcl=/usr/lib     \
            --enable-shared         \
            --mandir=/usr/share/man \
            --with-tclinclude=/usr/include
```

The meaning of the configure options:

`--with-tcl=/usr/lib`
This parameter is needed to tell **configure** where the **tclConfig.sh** script is located.

`--with-tclinclude=/usr/include`
This explicitly tells Expect where to find Tcl's internal headers.

Build the package:

```
make
```

To test the results, issue:

```
make test
```

Install the package:

```
make install
ln -svf expect5.45.4/libexpect5.45.4.so /usr/lib
```

8.16.2. Contents of Expect

Installed program: expect
Installed library: libexpect5.45.4.so

Short Descriptions

expect	Communicates with other interactive programs according to a script
`libexpect-5.45.4.so`	Contains functions that allow Expect to be used as a Tcl extension or to be used directly from C or C++ (without Tcl)

8.17. DejaGNU-1.6.3

The DejaGnu package contains a framework for running test suites on GNU tools. It is written in **expect**, which itself uses Tcl (Tool Command Language).

Approximate build time: 0.1 SBU
Required disk space: 6.9 MB

8.17.1. Installation of DejaGNU

The upstream recommends building DejaGNU in a dedicated build directory:

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

Prepare DejaGNU for compilation:

```
../configure --prefix=/usr
makeinfo --html --no-split -o doc/dejagnu.html ../doc/dejagnu.texi
makeinfo --plaintext       -o doc/dejagnu.txt  ../doc/dejagnu.texi
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
install -v -dm755  /usr/share/doc/dejagnu-1.6.3
install -v -m644   doc/dejagnu.{html,txt} /usr/share/doc/dejagnu-1.6.3
```

8.17.2. Contents of DejaGNU

Installed program: dejagnu and runtest

Short Descriptions

dejagnu DejaGNU auxiliary command launcher

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

8.18. Pkgconf-2.1.1

The pkgconf package is a successor to pkg-config and contains a tool for passing the include path and/or library paths to build tools during the configure and make phases of package installations.

Approximate build time: less than 0.1 SBU
Required disk space: 4.6 MB

8.18.1. Installation of Pkgconf

Prepare Pkgconf for compilation:

```
./configure --prefix=/usr              \
            --disable-static           \
            --docdir=/usr/share/doc/pkgconf-2.1.1
```

Compile the package:

```
make
```

Install the package:

```
make install
```

To maintain compatibility with the original Pkg-config create two symlinks:

```
ln -sv pkgconf   /usr/bin/pkg-config
ln -sv pkgconf.1 /usr/share/man/man1/pkg-config.1
```

8.18.2. Contents of Pkgconf

Installed programs: pkgconf, pkg-config (link to pkgconf), and bomtool
Installed library: libpkgconf.so
Installed directory: /usr/share/doc/pkgconf-2.1.1

Short Descriptions

pkgconf Returns meta information for the specified library or package

bomtool Generates a Software Bill Of Materials from pkg-config .pc files

`libpkgconf` Contains most of pkgconf's functionality, while allowing other tools like IDEs and compilers to use its frameworks

8.19. Binutils-2.42

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

Approximate build time: 2.2 SBU
Required disk space: 2.7 GB

8.19.1. Installation of Binutils

The Binutils documentation recommends building Binutils in a dedicated build directory:

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

Prepare Binutils for compilation:

```
../configure --prefix=/usr       \
             --sysconfdir=/etc   \
             --enable-gold       \
             --enable-ld=default \
             --enable-plugins    \
             --enable-shared     \
             --disable-werror    \
             --enable-64-bit-bfd \
             --with-system-zlib  \
             --enable-default-hash-style=gnu
```

The meaning of the new configure parameters:

`--enable-gold`
Build the gold linker and install it as ld.gold (alongside the default linker).

`--enable-ld=default`
Build the original bfd linker and install it as both ld (the default linker) and ld.bfd.

`--enable-plugins`
Enables plugin support for the linker.

`--with-system-zlib`
Use the installed zlib library instead of building the included version.

Compile the package:

```
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)`. For example, x86_64 machines would expand that to `/usr/x86_64-pc-linux-gnu`. Because this is a custom system, this target-specific directory in `/usr` is not required. `$(exec_prefix)/$(target_alias)` would be used if the system were used to cross-compile (for example, compiling a package on an Intel machine that generates code that can be executed on PowerPC machines).

Important

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

Test the results:

```
make -k check
```

For a list of failed tests, run:

```
grep '^FAIL:' $(find -name '*.log')
```

Twelve tests fail in the gold test suite when the `--enable-default-pie` and `--enable-default-ssp` options are passed to GCC.

Install the package:

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

Remove useless static libraries:

```
rm -fv /usr/lib/lib{bfd,ctf,ctf-nobfd,gprofng,opcodes,sframe}.a
```

8.19.2. Contents of Binutils

Installed programs:	addr2line, ar, as, c++filt, dwp, elfedit, gprof, gprofng, ld, ld.bfd, ld.gold, nm, objcopy, objdump, ranlib, readelf, size, strings, and strip
Installed libraries:	libbfd.so, libctf.so, libctf-nobfd.so, libgprofng.so, libopcodes.so, and libsframe.so
Installed directory:	/usr/lib/ldscripts

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
dwp	The DWARF packaging utility
elfedit	Updates the ELF headers of ELF files
gprof	Displays call graph profile data
gprofng	Gathers and analyzes performance 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
ld.gold	A cut down version of ld that only supports the elf object file format
ld.bfd	A hard link to **ld**
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
`libbfd`	The Binary File Descriptor library
`libctf`	The Compat ANSI-C Type Format debugging support library
`libctf-nobfd`	A libctf variant which does not use libbfd functionality
`libgprofng`	A library containing most routines used by **gprofng**
`libopcodes`	A library for dealing with opcodes—the “readable text” versions of instructions for the processor; it is used for building utilities like **objdump**
`libsframe`	A library to support online backtracing using a simple unwinder

8.20. GMP-6.3.0

The GMP package contains math libraries. These have useful functions for arbitrary precision arithmetic.

Approximate build time: 0.3 SBU
Required disk space: 54 MB

8.20.1. Installation of GMP

Note

If you are building for 32-bit x86, but you have a CPU which is capable of running 64-bit code *and* you have specified CFLAGS in the environment, the configure script will attempt to configure for 64-bits and fail. Avoid this by invoking the configure command below with

```
ABI=32 ./configure ...
```

Note

The default settings of GMP produce libraries optimized for the host processor. If libraries suitable for processors less capable than the host's CPU are desired, generic libraries can be created by appending the `--host=none-linux-gnu` option to the **configure** command.

Prepare GMP for compilation:

```
./configure --prefix=/usr    \
            --enable-cxx     \
            --disable-static \
            --docdir=/usr/share/doc/gmp-6.3.0
```

The meaning of the new configure options:

`--enable-cxx`
This parameter enables C++ support

`--docdir=/usr/share/doc/gmp-6.3.0`
This variable specifies the correct place for the documentation.

Compile the package and generate the HTML documentation:

```
make
make html
```

Important

The test suite for GMP in this section is considered critical. Do not skip it under any circumstances.

Test the results:

```
make check 2>&1 | tee gmp-check-log
```

Caution

The code in gmp is highly optimized for the processor where it is built. Occasionally, the code that detects the processor misidentifies the system capabilities and there will be errors in the tests or other applications using the gmp libraries with the message `Illegal instruction`. In this case, gmp should be reconfigured with the option `--host=none-linux-gnu` and rebuilt.

Ensure that at least 199 tests in the test suite passed. Check the results by issuing the following command:

```
awk '/# PASS:/{total+=$3} ; END{print total}' gmp-check-log
```

Install the package and its documentation:

```
make install
make install-html
```

8.20.2. Contents of GMP

Installed libraries: libgmp.so and libgmpxx.so
Installed directory: /usr/share/doc/gmp-6.3.0

Short Descriptions

`libgmp` Contains precision math functions

`libgmpxx` Contains C++ precision math functions

8.21. MPFR-4.2.1

The MPFR package contains functions for multiple precision math.

Approximate build time: 0.3 SBU
Required disk space: 44 MB

8.21.1. Installation of MPFR

Prepare MPFR for compilation:

```
./configure --prefix=/usr        \
            --disable-static     \
            --enable-thread-safe \
            --docdir=/usr/share/doc/mpfr-4.2.1
```

Compile the package and generate the HTML documentation:

```
make
make html
```

Important

The test suite for MPFR in this section is considered critical. Do not skip it under any circumstances.

Test the results and ensure that all 198 tests passed:

```
make check
```

Install the package and its documentation:

```
make install
make install-html
```

8.21.2. Contents of MPFR

Installed libraries: libmpfr.so
Installed directory: /usr/share/doc/mpfr-4.2.1

Short Descriptions

`libmpfr` Contains multiple-precision math functions

8.22. MPC-1.3.1

The MPC package contains a library for the arithmetic of complex numbers with arbitrarily high precision and correct rounding of the result.

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

8.22.1. Installation of MPC

Prepare MPC for compilation:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/mpc-1.3.1
```

Compile the package and generate the HTML documentation:

```
make
make html
```

To test the results, issue:

```
make check
```

Install the package and its documentation:

```
make install
make install-html
```

8.22.2. Contents of MPC

Installed libraries: libmpc.so
Installed directory: /usr/share/doc/mpc-1.3.1

Short Descriptions

`libmpc` Contains complex math functions

8.23. Attr-2.5.2

The Attr package contains utilities to administer the extended attributes of filesystem objects.

Approximate build time: less than 0.1 SBU
Required disk space: 4.2 MB

8.23.1. Installation of Attr

Prepare Attr for compilation:

```
./configure --prefix=/usr     \
            --disable-static  \
            --sysconfdir=/etc \
            --docdir=/usr/share/doc/attr-2.5.2
```

Compile the package:

```
make
```

The tests must be run on a filesystem that supports extended attributes such as the ext2, ext3, or ext4 filesystems. To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.23.2. Contents of Attr

Installed programs: attr, getfattr, and setfattr
Installed library: libattr.so
Installed directories: /usr/include/attr and /usr/share/doc/attr-2.5.2

Short Descriptions

attr Extends attributes on filesystem objects

getfattr Gets the extended attributes of filesystem objects

setfattr Sets the extended attributes of filesystem objects

`libattr` Contains the library functions for manipulating extended attributes

8.24. Acl-2.3.2

The Acl package contains utilities to administer Access Control Lists, which are used to define fine-grained discretionary access rights for files and directories.

Approximate build time: less than 0.1 SBU
Required disk space: 6.3 MB

8.24.1. Installation of Acl

Prepare Acl for compilation:

```
./configure --prefix=/usr         \
            --disable-static      \
            --docdir=/usr/share/doc/acl-2.3.2
```

Compile the package:

```
make
```

The Acl tests must be run on a filesystem that supports access controls, but not until the Coreutils package has been built, using the Acl libraries. If desired, return to this package and run **make check** after the Coreutils package has been built.

Install the package:

```
make install
```

8.24.2. Contents of Acl

Installed programs: chacl, getfacl, and setfacl
Installed library: libacl.so
Installed directories: /usr/include/acl and /usr/share/doc/acl-2.3.2

Short Descriptions

chacl Changes the access control list of a file or directory

getfacl Gets file access control lists

setfacl Sets file access control lists

`libacl` Contains the library functions for manipulating Access Control Lists

8.25. Libcap-2.69

The Libcap package implements the userspace interface to the POSIX 1003.1e capabilities available in Linux kernels. These capabilities partition the all-powerful root privilege into a set of distinct privileges.

Approximate build time: less than 0.1 SBU
Required disk space: 2.9 MB

8.25.1. Installation of Libcap

Prevent static libraries from being installed:

```
sed -i '/install -m.*STA/d' libcap/Makefile
```

Compile the package:

```
make prefix=/usr lib=lib
```

The meaning of the make option:

lib=lib
This parameter sets the library directory to `/usr/lib` rather than `/usr/lib64` on x86_64. It has no effect on x86.

To test the results, issue:

```
make test
```

Install the package:

```
make prefix=/usr lib=lib install
```

8.25.2. Contents of Libcap

Installed programs: capsh, getcap, getpcaps, and setcap
Installed library: libcap.so and libpsx.so

Short Descriptions

capsh A shell wrapper to explore and constrain capability support
getcap Examines file capabilities
getpcaps Displays the capabilities of the queried process(es)
setcap Sets file capabilities
`libcap` Contains the library functions for manipulating POSIX 1003.1e capabilities
`libpsx` Contains functions to support POSIX semantics for syscalls associated with the pthread library

8.26. Libxcrypt-4.4.36

The Libxcrypt package contains a modern library for one-way hashing of passwords.

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

8.26.1. Installation of Libxcrypt

Prepare Libxcrypt for compilation:

```
./configure --prefix=/usr                \
            --enable-hashes=strong,glibc \
            --enable-obsolete-api=no     \
            --disable-static             \
            --disable-failure-tokens
```

The meaning of the new configure options:

`--enable-hashes=strong,glibc`
Build strong hash algorithms recommended for security use cases, and the hash algorithms provided by traditional Glibc `libcrypt` for compatibility.

`--enable-obsolete-api=no`
Disable obsolete API functions. They are not needed for a modern Linux system built from source.

`--disable-failure-tokens`
Disable failure token feature. It's needed for compatibility with the traditional hash libraries of some platforms, but a Linux system based on Glibc does not need it.

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

Note

The instructions above disabled obsolete API functions since no package installed by compiling from sources would link against them at runtime. However, the only known binary-only applications that link against these functions require ABI version 1. If you must have such functions because of some binary-only application or to be compliant with LSB, build the package again with the following commands:

```
make distclean
./configure --prefix=/usr                \
            --enable-hashes=strong,glibc \
            --enable-obsolete-api=glibc  \
            --disable-static             \
            --disable-failure-tokens
make
cp -av --remove-destination .libs/libcrypt.so.1* /usr/lib
```

8.26.2. Contents of Libxcrypt

Installed libraries: libcrypt.so

Short Descriptions

`libcrypt` Contains functions to hash passwords

8.27. Shadow-4.14.5

The Shadow package contains programs for handling passwords in a secure way.

Approximate build time: 0.1 SBU
Required disk space: 49 MB

8.27.1. Installation of Shadow

Note

If you would like to enforce the use of strong passwords, refer to *https://www.linuxfromscratch.org/blfs/view/12.1/postlfs/cracklib.html* for installing CrackLib prior to building Shadow. Then add `--with-libcrack` to the **configure** command below.

Disable the installation of the **groups** program and its man pages, as Coreutils provides a better version. Also, prevent the installation of manual pages that were already installed in Section 8.3, “Man-pages-6.06”:

```
sed -i 's/groups$(EXEEXT) //' src/Makefile.in
find man -name Makefile.in -exec sed -i 's/groups\.1 / /'   {} \;
find man -name Makefile.in -exec sed -i 's/getspnam\.3 / /' {} \;
find man -name Makefile.in -exec sed -i 's/passwd\.5 / /'   {} \;
```

Instead of using the default *crypt* method, use the much more secure *YESCRYPT* 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. And, remove `/bin` and `/sbin` from the `PATH`, since they are simply symlinks to their counterparts in `/usr`.

Note

If you wish to include `/bin` and/or `/sbin` in the `PATH` for some reason, modify the `PATH` in `.bashrc` after LFS has been built.

```
sed -e 's:#ENCRYPT_METHOD DES:ENCRYPT_METHOD YESCRYPT:' \
    -e 's:/var/spool/mail:/var/mail:'                   \
    -e '/PATH=/{s@/sbin:@@;s@/bin:@@}'                  \
    -i etc/login.defs
```

Note

If you chose to build Shadow with Cracklib support, issue this command:

```
sed -i 's:DICTPATH.*:DICTPATH\t/lib/cracklib/pw_dict:' etc/login.defs
```

Prepare Shadow for compilation:

```
touch /usr/bin/passwd
./configure --sysconfdir=/etc   \
            --disable-static    \
            --with-{b,yes}crypt \
            --without-libbsd    \
            --with-group-name-max-length=32
```

The meaning of the new configuration options:

touch /usr/bin/passwd
The file `/usr/bin/passwd` needs to exist because its location is hardcoded in some programs; if it does not already exist, the installation script will create it in the wrong place.

`--with-{b,yes}crypt`
The shell expands this to two switches, *`--with-bcrypt`* and *`--with-yescrypt`*. They allow shadow to use the Bcrypt and Yescrypt algorithms implemented by Libxcrypt for hashing passwords. These algorithms are more secure (in particular, much more resistant to GPU-based attacks) than the traditional SHA algorithms.

`--with-group-name-max-length=32`
The longest permissible user name is 32 characters. Make the maximum length of a group name the same.

`--without-libbsd`
Do not use the readpassphrase function from libbsd which is not in LFS. Use the internal copy instead.

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make exec_prefix=/usr install
make -C man install-man
```

8.27.2. 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 you use 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 must be able to work with shadowed passwords.

To enable shadowed passwords, run the following command:

```
pwconv
```

To enable shadowed group passwords, run:

```
grpconv
```

Shadow's default configuration for the **useradd** utility needs some explanation. First, the default action for the **useradd** utility is to create the user and a group with the same name as the user. By default the user ID (UID) and group ID (GID) numbers will begin at 1000. This means if you don't pass extra parameters to **useradd**, each user will be a member of a unique group on the system. If this behavior is undesirable, you'll need to pass either the *`-g`* or *`-N`* parameter to **useradd**, or else change the setting of *`USERGROUPS_ENAB`* in `/etc/login.defs`. See *useradd(8)* for more information.

Second, to change the default parameters, the file `/etc/default/useradd` must be created and tailored to suit your particular needs. Create it with:

```
mkdir -p /etc/default
useradd -D --gid 999
```

`/etc/default/useradd` parameter explanations

`GROUP=999`

This parameter sets the beginning of the group numbers used in the `/etc/group` file. The particular value 999 comes from the *`--gid`* parameter above. You may set it to any desired value. Note that **useradd** will never reuse a UID or GID. If the number identified in this parameter is used, it will use the next available number. Note also that if you don't have a group with an ID equal to this number on your system, then the first time you use **useradd** without the *`-g`* parameter, an error message will be generated—`useradd: unknown GID 999`, even though the account has been created correctly. That is why we created the group `users` with this group ID in Section 7.6, “Creating Essential Files and Symlinks”.

`CREATE_MAIL_SPOOL=yes`

This parameter causes **useradd** to create a mailbox file for each new user. **useradd** will assign the group ownership of this file to the `mail` group with 0660 permissions. If you would rather not create these files, issue the following command:

```
sed -i '/MAIL/s/yes/no/' /etc/default/useradd
```

8.27.3. Setting the Root Password

Choose a password for user *root* and set it by running:

```
passwd root
```

8.27.4. Contents of Shadow

Installed programs:	chage, chfn, chgpasswd, chpasswd, chsh, expiry, faillog, getsubids, gpasswd, groupadd, groupdel, groupmems, groupmod, grpck, grpconv, grpunconv, login, logoutd, newgidmap, newgrp, newuidmap, newusers, nologin, passwd, pwck, pwconv, pwunconv, sg (link to newgrp), su, useradd, userdel, usermod, vigr (link to vipw), and vipw
Installed directories:	/etc/default and /usr/include/shadow
Installed libraries:	libsubid.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 user passwords in batch mode
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, and to reset the failure count
getsubids	Is used to list the subordinate id ranges for a user
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

groupmems Allows a user to administer his/her own group membership list without the requirement of super user privileges.

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

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

newgidmap Is used to set the gid mapping of a user namespace

newgrp Is used to change the current GID during a login session

newuidmap Is used to set the uid mapping of a user namespace

newusers Is used to create or update an entire series of user accounts

nologin Displays a message saying an account is not available; it is designed to be used as the default shell for disabled accounts

passwd Is used to change the password for a user or group account

pwck Verifies the integrity of the password files `/etc/passwd` and `/etc/shadow`

pwconv Creates or updates the shadow password file from the normal password file

pwunconv Updates `/etc/passwd` from `/etc/shadow` and then deletes the latter

sg Executes a given command while the user's GID is set to that of the given group

su Runs a shell with substitute user and group IDs

useradd Creates a new user with the given name, or updates the default new-user information

userdel Deletes the specified 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

`libsubid` library to handle subordinate id ranges for users and groups

8.28. GCC-13.2.0

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

Approximate build time: 42 SBU (with tests)
Required disk space: 5.5 GB

8.28.1. Installation of GCC

If building on x86_64, change the default directory name for 64-bit libraries to “lib”:

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' \
        -i.orig gcc/config/i386/t-linux64
  ;;
esac
```

The GCC documentation recommends building GCC in a dedicated build directory:

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

Prepare GCC for compilation:

```
../configure --prefix=/usr            \
             LD=ld                    \
             --enable-languages=c,c++ \
             --enable-default-pie     \
             --enable-default-ssp     \
             --disable-multilib       \
             --disable-bootstrap      \
             --disable-fixincludes    \
             --with-system-zlib
```

GCC supports seven different computer languages, but the prerequisites for most of them have not yet been installed. See the *BLFS Book GCC page* for instructions on how to build all of GCC's supported languages.

The meaning of the new configure parameters:

`LD=ld`
This parameter makes the configure script use the ld program installed by the Binutils package built earlier in this chapter, rather than the cross-built version which would otherwise be used.

`--disable-fixincludes`
By default, during the installation of GCC some system headers would be “fixed” to be used with GCC. This is not necessary for a modern Linux system, and potentially harmful if a package is reinstalled after installing GCC. This switch prevents GCC from “fixing” the headers.

`--with-system-zlib`
This switch tells GCC to link to the system installed copy of the Zlib library, rather than its own internal copy.

Note

PIE (position-independent executables) are binary programs that can be loaded anywhere in memory. Without PIE, the security feature named ASLR (Address Space Layout Randomization) can be applied for the shared libraries, but not for the executables themselves. Enabling PIE allows ASLR for the executables in addition to the shared libraries, and mitigates some attacks based on fixed addresses of sensitive code or data in the executables.

SSP (Stack Smashing Protection) is a technique to ensure that the parameter stack is not corrupted. Stack corruption can, for example, alter the return address of a subroutine, thus transferring control to some dangerous code (existing in the program or shared libraries, or injected by the attacker somehow).

Compile the package:

```
make
```

Important

In this section, the test suite for GCC is considered important, but it takes a long time. First-time builders are encouraged to run the test suite. The time to run the tests can be reduced significantly by adding -jx to the **make -k check** command below, where x is the number of CPU cores on your system.

One set of tests in the GCC test suite is known to exhaust the default stack, so increase the stack size prior to running the tests:

```
ulimit -s 32768
```

Test the results as a non-privileged user, but do not stop at errors:

```
chown -R tester .
su tester -c "PATH=$PATH make -k check"
```

To extract a summary of the test suite results, run:

```
../contrib/test_summary
```

To filter out only the summaries, pipe the output through **`grep -A7 Summ`**.

Results can be compared with those located at *https://www.linuxfromscratch.org/lfs/build-logs/12.1/* and *https://gcc.gnu.org/ml/gcc-testresults/*.

Eight gcc tests (out of over 185,000): `pr56837.c` and seven tests in the `analyzer` directory are known to fail. One libstdc++ test (out of over 15000), `copy.cc`, is known to fail. For g++, 21 tests (out of approximately 250,000): 14 “AddressSanitizer*” tests and 7 `interception-malloc-test-1.C` tests, are known to fail. Additionally, several tests in the `vect` directory are known to fail if the hardware does not support AVX.

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

Install the package:

```
make install
```

The GCC build directory is owned by `tester` now, and the ownership of the installed header directory (and its content) is incorrect. Change the ownership to the `root` user and group:

```
chown -v -R root:root \
    /usr/lib/gcc/$(gcc -dumpmachine)/13.2.0/include{,-fixed}
```

Create a symlink required by the *FHS* for "historical" reasons.

```
ln -svr /usr/bin/cpp /usr/lib
```

Many packages use the name **cc** to call the C compiler. We've already created **cc** as a symlink in gcc-pass2, create its man page as a symlink as well:

```
ln -sv gcc.1 /usr/share/man/man1/cc.1
```

Add a compatibility symlink to enable building programs with Link Time Optimization (LTO):

```
ln -sfv ../../libexec/gcc/$(gcc -dumpmachine)/13.2.0/liblto_plugin.so \
        /usr/lib/bfd-plugins/
```

Now that our final toolchain is in place, it is important to again ensure that compiling and linking will work as expected. We do this by performing some sanity checks:

```
echo 'int main(){}' > dummy.c
cc dummy.c -v -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

There should be no errors, and the output of the last command will be (allowing for platform-specific differences in the dynamic linker name):

```
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

Now make sure that we're set up to use the correct start files:

```
grep -E -o '/usr/lib.*/S?crt[1in].*succeeded' dummy.log
```

The output of the last command should be:

```
/usr/lib/gcc/x86_64-pc-linux-gnu/13.2.0/../../../../lib/Scrt1.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/13.2.0/../../../../lib/crti.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/13.2.0/../../../../lib/crtn.o succeeded
```

Depending on your machine architecture, the above may differ slightly. The difference will be the name of the directory after `/usr/lib/gcc`. The important thing to look for here is that **gcc** has found all three `crt*.o` files under the `/usr/lib` directory.

Verify that the compiler is searching for the correct header files:

```
grep -B4 '^ /usr/include' dummy.log
```

This command should return the following output:

```
#include <...> search starts here:
 /usr/lib/gcc/x86_64-pc-linux-gnu/13.2.0/include
 /usr/local/include
 /usr/lib/gcc/x86_64-pc-linux-gnu/13.2.0/include-fixed
 /usr/include
```

Again, the directory named after your target triplet may be different than the above, depending on your system architecture.

Next, verify that the new linker is being used with the correct search paths:

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |\n|g'
```

References to paths that have components with '-linux-gnu' should be ignored, but otherwise the output of the last command should be:

```
SEARCH_DIR("/usr/x86_64-pc-linux-gnu/lib64")
SEARCH_DIR("/usr/local/lib64")
SEARCH_DIR("/lib64")
SEARCH_DIR("/usr/lib64")
SEARCH_DIR("/usr/x86_64-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

A 32-bit system may use a few other directories. For example, here is the output from an i686 machine:

```
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib32")
SEARCH_DIR("/usr/local/lib32")
SEARCH_DIR("/lib32")
SEARCH_DIR("/usr/lib32")
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

Next make sure that we're using the correct libc:

```
grep "/lib.*/libc.so.6 " dummy.log
```

The output of the last command should be:

```
attempt to open /usr/lib/libc.so.6 succeeded
```

Make sure GCC is using the correct dynamic linker:

```
grep found dummy.log
```

The output of the last command should be (allowing for platform-specific differences in dynamic linker name):

```
found ld-linux-x86-64.so.2 at /usr/lib/ld-linux-x86-64.so.2
```

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. Any issues should be resolved before continuing with the process.

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

```
rm -v dummy.c a.out dummy.log
```

Finally, move a misplaced file:

```
mkdir -pv /usr/share/gdb/auto-load/usr/lib
mv -v /usr/lib/*gdb.py /usr/share/gdb/auto-load/usr/lib
```

8.28.2. Contents of GCC

Installed programs:	c++, cc (link to gcc), cpp, g++, gcc, gcc-ar, gcc-nm, gcc-ranlib, gcov, gcov-dump, gcov-tool, and lto-dump
Installed libraries:	libasan.{a,so}, libatomic.{a,so}, libcc1.so, libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libhwasan.{a,so}, libitm.{a,so}, liblsan.{a,so}, liblto_plugin.so, libquadmath.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.{a,so}, libstdc++exp.a, libstdc++fs.a, libsupc++.a, libtsan.{a,so}, and libubsan.{a,so}
Installed directories:	/usr/include/c++, /usr/lib/gcc, /usr/libexec/gcc, and /usr/share/gcc-13.2.0

Short Descriptions

c++	The C++ compiler
cc	The C compiler
cpp	The C preprocessor; it is used by the compiler to expand the #include, #define, and similar directives in the source files
g++	The C++ compiler
gcc	The C compiler
gcc-ar	A wrapper around **ar** that adds a plugin to the command line. This program is only used to add "link time optimization" and is not useful with the default build options.
gcc-nm	A wrapper around **nm** that adds a plugin to the command line. This program is only used to add "link time optimization" and is not useful with the default build options.
gcc-ranlib	A wrapper around **ranlib** that adds a plugin to the command line. This program is only used to add "link time optimization" and is not useful with the default build options.
gcov	A coverage testing tool; it is used to analyze programs to determine where optimizations will have the greatest effect
gcov-dump	Offline gcda and gcno profile dump tool
gcov-tool	Offline gcda profile processing tool
lto-dump	Tool for dumping object files produced by GCC with LTO enabled
`libasan`	The Address Sanitizer runtime library
`libatomic`	GCC atomic built-in runtime library
`libcc1`	A library that allows GDB to make use of GCC
`libgcc`	Contains run-time support for **gcc**
`libgcov`	This library is linked into a program when GCC is instructed to enable profiling
`libgomp`	GNU implementation of the OpenMP API for multi-platform shared-memory parallel programming in C/C++ and Fortran
`libhwasan`	The Hardware-assisted Address Sanitizer runtime library
`libitm`	The GNU transactional memory library
`liblsan`	The Leak Sanitizer runtime library
`liblto_plugin`	GCC's LTO plugin allows Binutils to process object files produced by GCC with LTO enabled
`libquadmath`	GCC Quad Precision Math Library API
`libssp`	Contains routines supporting GCC's stack-smashing protection functionality. Normally it is not used, because Glibc also provides those routines.
`libstdc++`	The standard C++ library
`libstdc++exp`	Experimental C++ Contracts library
`libstdc++fs`	ISO/IEC TS 18822:2015 Filesystem library
`libsupc++`	Provides supporting routines for the C++ programming language
`libtsan`	The Thread Sanitizer runtime library
`libubsan`	The Undefined Behavior Sanitizer runtime library

8.29. Ncurses-6.4-20230520

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

Approximate build time: 0.2 SBU
Required disk space: 45 MB

8.29.1. Installation of Ncurses

Prepare Ncurses for compilation:

```
./configure --prefix=/usr           \
            --mandir=/usr/share/man \
            --with-shared           \
            --without-debug         \
            --without-normal        \
            --with-cxx-shared       \
            --enable-pc-files       \
            --enable-widec          \
            --with-pkg-config-libdir=/usr/lib/pkgconfig
```

The meaning of the new configure options:

`--with-shared`
This makes Ncurses build and install shared C libraries.

`--without-normal`
This prevents Ncurses building and installing static C libraries.

`--without-debug`
This prevents Ncurses building and installing debug libraries.

`--with-cxx-shared`
This makes Ncurses build and install shared C++ bindings. It also prevents it building and installing static C++ bindings.

`--enable-pc-files`
This switch generates and installs .pc files for pkg-config.

`--enable-widec`
This switch causes wide-character libraries (e.g., `libncursesw.so.6.4`) to be built instead of normal ones (e.g., `libncurses.so.6.4`). These wide-character libraries are usable in both multibyte and traditional 8-bit locales, while normal libraries work properly only in 8-bit locales. Wide-character and normal libraries are source-compatible, but not binary-compatible.

Compile the package:

```
make
```

This package has a test suite, but it can only be run after the package has been installed. The tests reside in the `test/` directory. See the `README` file in that directory for further details.

The installation of this package will overwrite `libncursesw.so.6.4` in-place. It may crash the shell process which is using code and data from the library file. Install the package with `DESTDIR`, and replace the library file correctly using **install** command (the header `curses.h` is also edited to ensure the wide-character ABI to be used as what we've done in Section 6.3, “Ncurses-6.4-20230520”):

```
make DESTDIR=$PWD/dest install
install -vm755 dest/usr/lib/libncursesw.so.6.4 /usr/lib
rm -v  dest/usr/lib/libncursesw.so.6.4
sed -e 's/^#if.*XOPEN.*$/#if 1/' \
    -i dest/usr/include/curses.h
cp -av dest/* /
```

Many applications still expect the linker to be able to find non-wide-character Ncurses libraries. Trick such applications into linking with wide-character libraries by means of symlinks (note that the `.so` links are only safe with `curses.h` edited to always use the wide-character ABI):

```
for lib in ncurses form panel menu ; do
    ln -sfv lib${lib}w.so /usr/lib/lib${lib}.so
    ln -sfv ${lib}w.pc    /usr/lib/pkgconfig/${lib}.pc
done
```

Finally, make sure that old applications that look for `-lcurses` at build time are still buildable:

```
ln -sfv libncursesw.so /usr/lib/libcurses.so
```

If desired, install the Ncurses documentation:

```
cp -v -R doc -T /usr/share/doc/ncurses-6.4-20230520
```

Note

The instructions above don't create non-wide-character Ncurses libraries since no package installed by compiling from sources would link against them at runtime. However, the only known binary-only applications that link against non-wide-character Ncurses libraries require version 5. If you must have such libraries because of some binary-only application or to be compliant with LSB, build the package again with the following commands:

```
make distclean
./configure --prefix=/usr    \
            --with-shared    \
            --without-normal \
            --without-debug  \
            --without-cxx-binding \
            --with-abi-version=5
make sources libs
cp -av lib/lib*.so.5* /usr/lib
```

8.29.2. Contents of Ncurses

Installed programs:	captoinfo (link to tic), clear, infocmp, infotocap (link to tic), ncursesw6-config, reset (link to tset), tabs, tic, toe, tput, and tset
Installed libraries:	libcurses.so (symlink), libform.so (symlink), libformw.so, libmenu.so (symlink), libmenuw.so, libncurses.so (symlink), libncursesw.so, libncurses++w.so, libpanel.so (symlink), and libpanelw.so,
Installed directories:	/usr/share/tabset, /usr/share/terminfo, and /usr/share/doc/ncurses-6.4-20230520

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
ncursesw6-config	Provides configuration information for ncurses
reset	Reinitializes a terminal to its default values
tabs	Clears and sets tab stops on a terminal
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
`libncursesw`	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**
`libncurses++w`	Contains C++ binding for other libraries in this package
`libformw`	Contains functions to implement forms
`libmenuw`	Contains functions to implement menus
`libpanelw`	Contains functions to implement panels

8.30. Sed-4.9

The Sed package contains a stream editor.

Approximate build time: 0.3 SBU
Required disk space: 30 MB

8.30.1. Installation of Sed

Prepare Sed for compilation:

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

Compile the package and generate the HTML documentation:

```
make
make html
```

To test the results, issue:

```
chown -R tester .
su tester -c "PATH=$PATH make check"
```

Install the package and its documentation:

```
make install
install -d -m755           /usr/share/doc/sed-4.9
install -m644 doc/sed.html /usr/share/doc/sed-4.9
```

8.30.2. Contents of Sed

Installed program: sed
Installed directory: /usr/share/doc/sed-4.9

Short Descriptions

sed Filters and transforms text files in a single pass

8.31. Psmisc-23.6

The Psmisc package contains programs for displaying information about running processes.

Approximate build time: less than 0.1 SBU
Required disk space: 6.6 MB

8.31.1. Installation of Psmisc

Prepare Psmisc for compilation:

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

Compile the package:

```
make
```

To run the test suite, run:

```
make check
```

Install the package:

```
make install
```

8.31.2. Contents of Psmisc

Installed programs: fuser, killall, peekfd, prtstat, pslog, 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

peekfd Peek at file descriptors of a running process, given its PID

prtstat Prints information about a process

pslog Reports current logs path of a process

pstree Displays running processes as a tree

pstree.x11 Same as **pstree**, except that it waits for confirmation before exiting

8.32. Gettext-0.22.4

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.

Approximate build time: 1.4 SBU
Required disk space: 250 MB

8.32.1. Installation of Gettext

Prepare Gettext for compilation:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/gettext-0.22.4
```

Compile the package:

```
make
```

To test the results (this takes a long time, around 3 SBUs), issue:

```
make check
```

Install the package:

```
make install
chmod -v 0755 /usr/lib/preloadable_libintl.so
```

8.32.2. Contents of Gettext

Installed programs: autopoint, envsubst, gettext, gettext.sh, gettextize, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, recode-sr-latin, and xgettext
Installed libraries: libasprintf.so, libgettextlib.so, libgettextpo.so, libgettextsrc.so, libtextstyle.so, and preloadable_libintl.so
Installed directories: /usr/lib/gettext, /usr/share/doc/gettext-0.22.4, /usr/share/gettext, and /usr/share/gettext-0.22.4

Short Descriptions

autopoint Copies standard Gettext infrastructure files into a source package

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

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 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
recode-sr-latin	Recodes Serbian text from Cyrillic to Latin script
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`	Contains 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`	Provides common routines used by the various Gettext programs; these are not intended for general use
`libtextstyle`	Text styling library
`preloadable_libintl`	A library, intended to be used by LD_PRELOAD, that helps `libintl` log untranslated messages

8.33. Bison-3.8.2

The Bison package contains a parser generator.

Approximate build time: 2.3 SBU
Required disk space: 62 MB

8.33.1. Installation of Bison

Prepare Bison for compilation:

```
./configure --prefix=/usr --docdir=/usr/share/doc/bison-3.8.2
```

Compile the package:

```
make
```

To test the results (about 5.5 SBU), issue:

```
make check
```

Install the package:

```
make install
```

8.33.2. Contents of Bison

Installed programs: bison and yacc
Installed library: liby.a
Installed directory: /usr/share/bison

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` The Yacc library containing implementations of Yacc-compatible `yyerror` and `main` functions; this library is normally not very useful, but POSIX requires it

8.34. Grep-3.11

The Grep package contains programs for searching through the contents of files.

Approximate build time: 0.4 SBU
Required disk space: 39 MB

8.34.1. Installation of Grep

First, remove a warning about using egrep and fgrep that makes tests on some packages fail:

```
sed -i "s/echo/#echo/" src/egrep.sh
```

Prepare Grep for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.34.2. Contents of Grep

Installed programs: egrep, fgrep, and grep

Short Descriptions

egrep Prints lines matching an extended regular expression. It is obsolete, use **grep -E** instead

fgrep Prints lines matching a list of fixed strings. It is obsolete, use **grep -F** instead

grep Prints lines matching a basic regular expression

8.35. Bash-5.2.21

The Bash package contains the Bourne-Again Shell.

Approximate build time: 1.2 SBU
Required disk space: 52 MB

8.35.1. Installation of Bash

First, fix some issues identified upstream:

```
patch -Np1 -i ../bash-5.2.21-upstream_fixes-1.patch
```

Prepare Bash for compilation:

```
./configure --prefix=/usr             \
            --without-bash-malloc     \
            --with-installed-readline \
            --docdir=/usr/share/doc/bash-5.2.21
```

The meaning of the new 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
```

Skip down to “Install the package” if not running the test suite.

To prepare the tests, ensure that the `tester` user can write to the sources tree:

```
chown -R tester .
```

The test suite of this package is designed to be run as a non-`root` user who owns the terminal connected to standard input. To satisfy the requirement, spawn a new pseudo terminal using Expect and run the tests as the `tester` user:

```
su -s /usr/bin/expect tester << "EOF"
set timeout -1
spawn make tests
expect eof
lassign [wait] _ _ _ value
exit $value
EOF
```

The test suite uses **diff** to detect the difference between test script output and the expected output. Any output from **diff** (prefixed with < and >) indicates a test failure, unless there is a message saying the difference can be ignored. One test named `run-builtins` is known to fail on some host distros with a difference on the first line of the output.

Install the package:

```
make install
```

Run the newly compiled **bash** program (replacing the one that is currently being executed):

```
exec /usr/bin/bash --login
```

8.35.2. Contents of Bash

Installed programs: bash, bashbug, and sh (link to bash)
Installed directory: /usr/include/bash, /usr/lib/bash, and /usr/share/doc/bash-5.2.21

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

8.36. Libtool-2.4.7

The Libtool package contains the GNU generic library support script. It makes the use of shared libraries simpler with a consistent, portable interface.

Approximate build time: 0.6 SBU
Required disk space: 45 MB

8.36.1. Installation of Libtool

Prepare Libtool for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make -k check
```

Five tests are known to fail in the LFS build environment due to a circular dependency, but these tests pass if rechecked after automake has been installed. Additionally, with grep-3.8 or newer, two tests will trigger a warning for non-POSIX regular expressions and fail.

Install the package:

```
make install
```

Remove a useless static library:

```
rm -fv /usr/lib/libltdl.a
```

8.36.2. Contents of Libtool

Installed programs: libtool and libtoolize
Installed libraries: libltdl.so
Installed directories: /usr/include/libltdl and /usr/share/libtool

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 opening dynamically loaded libraries

8.37. GDBM-1.23

The GDBM package contains the GNU Database Manager. It is a library of database functions that uses extensible hashing and works like the standard UNIX dbm. The library provides primitives for storing key/data pairs, searching and retrieving the data by its key and deleting a key along with its data.

Approximate build time: 0.1 SBU
Required disk space: 13 MB

8.37.1. Installation of GDBM

Prepare GDBM for compilation:

```
./configure --prefix=/usr    \
            --disable-static \
            --enable-libgdbm-compat
```

The meaning of the configure option:

`--enable-libgdbm-compat`
This switch enables building the libgdbm compatibility library. Some packages outside of LFS may require the older DBM routines it provides.

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.37.2. Contents of GDBM

Installed programs: gdbm_dump, gdbm_load, and gdbmtool
Installed libraries: libgdbm.so and libgdbm_compat.so

Short Descriptions

gdbm_dump	Dumps a GDBM database to a file
gdbm_load	Recreates a GDBM database from a dump file
gdbmtool	Tests and modifies a GDBM database
`libgdbm`	Contains functions to manipulate a hashed database
`libgdbm_compat`	Compatibility library containing older DBM functions

8.38. Gperf-3.1

Gperf generates a perfect hash function from a key set.

Approximate build time: less than 0.1 SBU
Required disk space: 6.1 MB

8.38.1. Installation of Gperf

Prepare Gperf for compilation:

```
./configure --prefix=/usr --docdir=/usr/share/doc/gperf-3.1
```

Compile the package:

```
make
```

The tests are known to fail if running multiple simultaneous tests (-j option greater than 1). To test the results, issue:

```
make -j1 check
```

Install the package:

```
make install
```

8.38.2. Contents of Gperf

Installed program: gperf
Installed directory: /usr/share/doc/gperf-3.1

Short Descriptions

gperf Generates a perfect hash from a key set

8.39. Expat-2.6.0

The Expat package contains a stream oriented C library for parsing XML.

Approximate build time: 0.1 SBU
Required disk space: 13 MB

8.39.1. Installation of Expat

Prepare Expat for compilation:

```
./configure --prefix=/usr    \
            --disable-static \
            --docdir=/usr/share/doc/expat-2.6.0
```

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

If desired, install the documentation:

```
install -v -m644 doc/*.{html,css} /usr/share/doc/expat-2.6.0
```

8.39.2. Contents of Expat

Installed program: xmlwf
Installed libraries: libexpat.so
Installed directory: /usr/share/doc/expat-2.6.0

Short Descriptions

xmlwf Is a non-validating utility to check whether or not XML documents are well formed

`libexpat` Contains API functions for parsing XML

8.40. Inetutils-2.5

The Inetutils package contains programs for basic networking.

Approximate build time: 0.2 SBU
Required disk space: 35 MB

8.40.1. Installation of Inetutils

Prepare Inetutils for compilation:

```
./configure --prefix=/usr        \
            --bindir=/usr/bin    \
            --localstatedir=/var \
            --disable-logger     \
            --disable-whois      \
            --disable-rcp        \
            --disable-rexec      \
            --disable-rlogin     \
            --disable-rsh        \
            --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 more recent version.

`--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-r*`
These parameters disable building obsolete programs that should not be used due to security issues. The functions provided by these programs can be provided by the openssh package in the BLFS book.

`--disable-servers`
This disables the installation of the various network servers included as part of the Inetutils package. These servers are deemed not appropriate in a basic LFS system. Some are insecure by nature and are only considered safe on trusted networks. Note that better replacements are available for many of these servers.

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

Move a program to the proper location:

```
mv -v /usr/{,s}bin/ifconfig
```

8.40.2. Contents of Inetutils

Installed programs: dnsdomainname, ftp, ifconfig, hostname, ping, ping6, talk, telnet, tftp, and traceroute

Short Descriptions

dnsdomainname	Show the system's DNS domain name
ftp	Is the file transfer protocol program
hostname	Reports or sets the name of the host
ifconfig	Manages network interfaces
ping	Sends echo-request packets and reports how long the replies take
ping6	A version of **ping** for IPv6 networks
talk	Is used to chat with another user
telnet	An interface to the TELNET protocol
tftp	A trivial file transfer program
traceroute	Traces the route your packets take from the host you are working on to another host on a network, showing all the intermediate hops (gateways) along the way

8.41. Less-643

The Less package contains a text file viewer.

Approximate build time: less than 0.1 SBU
Required disk space: 12 MB

8.41.1. Installation of Less

Prepare Less for compilation:

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

The meaning of the configure options:

`--sysconfdir=/etc`
This option tells the programs created by the package to look in `/etc` for the configuration files.

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.41.2. 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**

8.42. Perl-5.38.2

The Perl package contains the Practical Extraction and Report Language.

Approximate build time: 1.5 SBU
Required disk space: 239 MB

8.42.1. Installation of Perl

This version of Perl builds the Compress::Raw::Zlib and Compress::Raw::BZip2 modules. By default Perl will use an internal copy of the sources for the build. Issue the following command so that Perl will use the libraries installed on the system:

```
export BUILD_ZLIB=False
export BUILD_BZIP2=0
```

To have full control over the way Perl is set up, you can remove the "-des" options from the following command and hand-pick the way this package is built. Alternatively, use the command exactly as shown below to use the defaults that Perl auto-detects:

```
sh Configure -des                                         \
             -Dprefix=/usr                                \
             -Dvendorprefix=/usr                          \
             -Dprivlib=/usr/lib/perl5/5.38/core_perl      \
             -Darchlib=/usr/lib/perl5/5.38/core_perl      \
             -Dsitelib=/usr/lib/perl5/5.38/site_perl      \
             -Dsitearch=/usr/lib/perl5/5.38/site_perl     \
             -Dvendorlib=/usr/lib/perl5/5.38/vendor_perl  \
             -Dvendorarch=/usr/lib/perl5/5.38/vendor_perl \
             -Dman1dir=/usr/share/man/man1                \
             -Dman3dir=/usr/share/man/man3                \
             -Dpager="/usr/bin/less -isR"                 \
             -Duseshrplib                                 \
             -Dusethreads
```

The meaning of the new Configure options:

`-Dpager="/usr/bin/less -isR"`
This ensures that **`less`** is used instead of **`more`**.

`-Dman1dir=/usr/share/man/man1 -Dman3dir=/usr/share/man/man3`
Since Groff is not installed yet, **Configure** will not create man pages for Perl. These parameters override this behavior.

`-Dusethreads`
Build Perl with support for threads.

Compile the package:

```
make
```

To test the results (approximately 11 SBU), issue:

```
TEST_JOBS=$(nproc) make test_harness
```

Install the package and clean up:

```
make install
unset BUILD_ZLIB BUILD_BZIP2
```

8.42.2. Contents of Perl

Installed programs: corelist, cpan, enc2xs, encguess, h2ph, h2xs, instmodsh, json_pp, libnetcfg, perl, perl5.38.2 (hard link to perl), perlbug, perldoc, perlivp, perlthanks (hard link to perlbug), piconv, pl2pm, pod2html, pod2man, pod2text, pod2usage, podchecker, podselect, prove, ptar, ptardiff, ptargrep, shasum, splain, xsubpp, and zipdetails
Installed libraries: Many which cannot all be listed here
Installed directory: /usr/lib/perl5

Short Descriptions

corelist A command line front end to Module::CoreList

cpan Interact with the Comprehensive Perl Archive Network (CPAN) from the command line

enc2xs Builds a Perl extension for the Encode module from either Unicode Character Mappings or Tcl Encoding Files

encguess Guess the encoding type of one or several files

h2ph Converts `.h` C header files to `.ph` Perl header files

h2xs Converts `.h` C header files to Perl extensions

instmodsh Shell script for examining installed Perl modules; it can create a tarball from an installed module

json_pp Converts data between certain input and output formats

libnetcfg Can be used to configure the `libnet` Perl module

perl Combines some of the best features of C, **sed**, **awk** and **sh** into a single Swiss Army language

perl5.38.2 A hard link to **perl**

perlbug Used to generate bug reports about Perl, or the modules that come with it, and mail them

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

perlthanks Used to generate thank you messages to mail to the Perl developers

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

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 Command line tool for running tests against the Test::Harness module

ptar A **tar**-like program written in Perl

ptardiff A Perl program that compares an extracted archive with an unextracted one

ptargrep	A Perl program that applies pattern matching to the contents of files in a tar archive
shasum	Prints or checks SHA checksums
splain	Is used to force verbose warning diagnostics in Perl
xsubpp	Converts Perl XS code into C code
zipdetails	Displays details about the internal structure of a Zip file

8.43. XML::Parser-2.47

The XML::Parser module is a Perl interface to James Clark's XML parser, Expat.

Approximate build time: less than 0.1 SBU
Required disk space: 2.4 MB

8.43.1. Installation of XML::Parser

Prepare XML::Parser for compilation:

```
perl Makefile.PL
```

Compile the package:

```
make
```

To test the results, issue:

```
make test
```

Install the package:

```
make install
```

8.43.2. Contents of XML::Parser

Installed module: Expat.so

Short Descriptions

`Expat` provides the Perl Expat interface

8.44. Intltool-0.51.0

The Intltool is an internationalization tool used for extracting translatable strings from source files.

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

8.44.1. Installation of Intltool

First fix a warning that is caused by perl-5.22 and later:

```
sed -i 's:\\\${:\\\$\\{:' intltool-update.in
```

Note

The above regular expression looks unusual because of all the backslashes. What it does is add a backslash before the right brace character in the sequence '\${' resulting in '\$\{'.

Prepare Intltool for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
install -v -Dm644 doc/I18N-HOWTO /usr/share/doc/intltool-0.51.0/I18N-HOWTO
```

8.44.2. Contents of Intltool

Installed programs: intltool-extract, intltool-merge, intltool-prepare, intltool-update, and intltoolize
Installed directories: /usr/share/doc/intltool-0.51.0 and /usr/share/intltool

Short Descriptions

intltoolize Prepares a package to use intltool

intltool-extract Generates header files that can be read by **gettext**

intltool-merge Merges translated strings into various file types

intltool-prepare Updates pot files and merges them with translation files

intltool-update Updates the po template files and merges them with the translations

8.45. Autoconf-2.72

The Autoconf package contains programs for producing shell scripts that can automatically configure source code.

Approximate build time: less than 0.1 SBU (about 0.5 SBU with tests)
Required disk space: 25 MB

8.45.1. Installation of Autoconf

Prepare Autoconf for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.45.2. Contents of Autoconf

Installed programs: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate, and ifnames
Installed directory: /usr/share/autoconf

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 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**.]

8.46. Automake-1.16.5

The Automake package contains programs for generating Makefiles for use with Autoconf.

Approximate build time: less than 0.1 SBU (about 1.6 SBU with tests)
Required disk space: 115 MB

8.46.1. Installation of Automake

Prepare Automake for compilation:

```
./configure --prefix=/usr --docdir=/usr/share/doc/automake-1.16.5
```

Compile the package:

```
make
```

Using four parallel jobs speeds up the tests, even on systems with less logical cores, due to internal delays in individual tests. To test the results, issue:

```
make -j$(($(nproc)>4?$(nproc):4)) check
```

Replace *`$((...))`* with the number of logical cores you want to use if you don't want to use all.

The test t/subobj.sh is known to fail.

Install the package:

```
make install
```

8.46.2. Contents of Automake

Installed programs: aclocal, aclocal-1.16 (hard linked with aclocal), automake, and automake-1.16 (hard linked with automake)
Installed directories: /usr/share/aclocal-1.16, /usr/share/automake-1.16, and /usr/share/doc/automake-1.16.5

Short Descriptions

aclocal Generates `aclocal.m4` files based on the contents of `configure.in` files

aclocal-1.16 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-1.16 A hard link to **automake**

8.47. OpenSSL-3.2.1

The OpenSSL package contains management tools and libraries relating to cryptography. These are useful for providing cryptographic functions to other packages, such as OpenSSH, email applications, and web browsers (for accessing HTTPS sites).

Approximate build time: 1.8 SBU
Required disk space: 805 MB

8.47.1. Installation of OpenSSL

Prepare OpenSSL for compilation:

```
./config --prefix=/usr         \
         --openssldir=/etc/ssl \
         --libdir=lib          \
         shared                \
         zlib-dynamic
```

Compile the package:

```
make
```

To test the results, issue:

```
HARNESS_JOBS=$(nproc) make test
```

One test, 30-test_afalg.t, is known to fail if the host kernel does not have `CONFIG_CRYPTO_USER_API_SKCIPHER` enabled, or does not have any options providing an AES with CBC implementation (for example, the combination of `CONFIG_CRYPTO_AES` and `CONFIG_CRYPTO_CBC`, or `CONFIG_CRYPTO_AES_NI_INTEL` if the CPU supports AES-NI) enabled. If it fails, it can safely be ignored.

Install the package:

```
sed -i '/INSTALL_LIBS/s/libcrypto.a libssl.a//' Makefile
make MANSUFFIX=ssl install
```

Add the version to the documentation directory name, to be consistent with other packages:

```
mv -v /usr/share/doc/openssl /usr/share/doc/openssl-3.2.1
```

If desired, install some additional documentation:

```
cp -vfr doc/* /usr/share/doc/openssl-3.2.1
```

Note

You should update OpenSSL when a new version which fixes vulnerabilities is announced. Since OpenSSL 3.0.0, the OpenSSL versioning scheme follows the MAJOR.MINOR.PATCH format. API/ABI compatibility is guaranteed for the same MAJOR version number. Because LFS installs only the shared libraries, there is no need to recompile packages which link to `libcrypto.so` or `libssl.so` *when upgrading to a version with the same MAJOR version number*.

However, any running programs linked to those libraries need to be stopped and restarted. Read the related entries in Section 8.2.1, “Upgrade Issues” for details.

8.47.2. Contents of OpenSSL

Installed programs: c_rehash and openssl
Installed libraries: libcrypto.so and libssl.so
Installed directories: /etc/ssl, /usr/include/openssl, /usr/lib/engines and /usr/share/doc/openssl-3.2.1

Short Descriptions

c_rehash is a Perl script that scans all files in a directory and adds symbolic links to their hash values. Use of **c_rehash** is considered obsolete and should be replaced by **openssl rehash** command

openssl is a command-line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for various functions which are documented in *openssl(1)*

`libcrypto.so` implements a wide range of cryptographic algorithms used in various Internet standards. The services provided by this library are used by the OpenSSL implementations of SSL, TLS and S/MIME, and they have also been used to implement OpenSSH, OpenPGP, and other cryptographic standards

`libssl.so` implements the Transport Layer Security (TLS v1) protocol. It provides a rich API, documentation on which can be found in *ssl(7)*

8.48. Kmod-31

The Kmod package contains libraries and utilities for loading kernel modules

Approximate build time: less than 0.1 SBU
Required disk space: 12 MB

8.48.1. Installation of Kmod

Prepare Kmod for compilation:

```
./configure --prefix=/usr          \
            --sysconfdir=/etc      \
            --with-openssl         \
            --with-xz              \
            --with-zstd            \
            --with-zlib
```

The meaning of the configure options:

`--with-openssl`
This option enables Kmod to handle PKCS7 signatures for kernel modules.

`--with-xz`, *`--with-zlib`*, and *`--with-zstd`*
These options enable Kmod to handle compressed kernel modules.

Compile the package:

```
make
```

The test suite of this package requires raw kernel headers (not the “sanitized” kernel headers installed earlier), which are beyond the scope of LFS.

Install the package and create symlinks for compatibility with Module-Init-Tools (the package that previously handled Linux kernel modules):

```
make install

for target in depmod insmod modinfo modprobe rmmod; do
  ln -sfv ../bin/kmod /usr/sbin/$target
done

ln -sfv kmod /usr/bin/lsmod
```

8.48.2. Contents of Kmod

Installed programs: depmod (link to kmod), insmod (link to kmod), kmod, lsmod (link to kmod), modinfo (link to kmod), modprobe (link to kmod), and rmmod (link to kmod)
Installed library: libkmod.so

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

insmod Installs a loadable module in the running kernel

kmod Loads and unloads kernel modules

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
`libkmod`	This library is used by other programs to load and unload kernel modules

8.49. Libelf from Elfutils-0.190

Libelf is a library for handling ELF (Executable and Linkable Format) files.

Approximate build time: 0.3 SBU
Required disk space: 124 MB

8.49.1. Installation of Libelf

Libelf is part of the elfutils-0.190 package. Use the elfutils-0.190.tar.bz2 file as the source tarball.

Prepare Libelf for compilation:

```
./configure --prefix=/usr                \
            --disable-debuginfod         \
            --enable-libdebuginfod=dummy
```

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install only Libelf:

```
make -C libelf install
install -vm644 config/libelf.pc /usr/lib/pkgconfig
rm /usr/lib/libelf.a
```

8.49.2. Contents of Libelf

Installed library: libelf.so
Installed directory: /usr/include/elfutils

Short Descriptions

`libelf.so` Contains API functions to handle ELF object files

8.50. Libffi-3.4.4

The Libffi library provides a portable, high level programming interface to various calling conventions. This allows a programmer to call any function specified by a call interface description at run time.

FFI stands for Foreign Function Interface. An FFI allows a program written in one language to call a program written in another language. Specifically, Libffi can provide a bridge between an interpreter like Perl, or Python, and shared library subroutines written in C, or C++.

Approximate build time: 1.8 SBU
Required disk space: 11 MB

8.50.1. Installation of Libffi

Note

Like GMP, Libffi builds with optimizations specific to the processor in use. If building for another system, change the value of the *--with-gcc-arch=* parameter in the following command to an architecture name fully implemented by the CPU on that system. If this is not done, all applications that link to `libffi` will trigger Illegal Operation Errors.

Prepare Libffi for compilation:

```
./configure --prefix=/usr          \
            --disable-static       \
            --with-gcc-arch=native
```

The meaning of the configure option:

--with-gcc-arch=native
Ensure GCC optimizes for the current system. If this is not specified, the system is guessed and the code generated may not be correct. If the generated code will be copied from the native system to a less capable system, use the less capable system as a parameter. For details about alternative system types, see *the x86 options in the GCC manual*.

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.50.2. Contents of Libffi

Installed library: libffi.so

Short Descriptions

`libffi` Contains the foreign function interface API functions

8.51. Python-3.12.2

The Python 3 package contains the Python development environment. It is useful for object-oriented programming, writing scripts, prototyping large programs, and developing entire applications. Python is an interpreted computer language.

Approximate build time: 1.8 SBU
Required disk space: 485 MB

8.51.1. Installation of Python 3

Prepare Python for compilation:

```
./configure --prefix=/usr        \
            --enable-shared      \
            --with-system-expat  \
            --enable-optimizations
```

The meaning of the configure options:

`--with-system-expat`
This switch enables linking against the system version of Expat.

`--enable-optimizations`
This switch enables extensive, but time-consuming, optimization steps. The interpreter is built twice; tests performed on the first build are used to improve the optimized final version.

Compile the package:

```
make
```

Running the tests at this point is not recommended. The tests are known to hang indefinitely in the partial LFS environment. If desired, the tests can be rerun at the end of this chapter, or when Python 3 is reinstalled in BLFS. To run the tests anyway, issue **make test**.

Install the package:

```
make install
```

We use the **pip3** command to install Python 3 programs and modules for all users as `root` in several places in this book. This conflicts with the Python developers' recommendation: to install packages into a virtual environment, or into the home directory of a regular user (by running **pip3** as this user). A multi-line warning is triggered whenever **pip3** is issued by the `root` user.

The main reason for the recommendation is to avoid conflicts with the system's package manager (**dpkg**, for example). LFS does not have a system-wide package manager, so this is not a problem. Also, **pip3** will check for a new version of itself whenever it's run. Since domain name resolution is not yet configured in the LFS chroot environment, **pip3** cannot check for a new version of itself, and will produce a warning.

After we boot the LFS system and set up a network connection, a different warning will be issued, telling the user to update **pip3** from a pre-built wheel on PyPI (whenever a new version is available). But LFS considers **pip3** to be a part of Python 3, so it should not be updated separately. Also, an update from a pre-built wheel would deviate

from our objective: to build a Linux system from source code. So the warning about a new version of **pip3** should be ignored as well. If you wish, you can suppress all these warnings by running the following command, which creates a configuration file:

```
cat > /etc/pip.conf << EOF
[global]
root-user-action = ignore
disable-pip-version-check = true
EOF
```

Important

In LFS and BLFS we normally build and install Python modules with the **pip3** command. Please be sure that the **pip3 install** commands in both books are run as the `root` user (unless it's for a Python virtual environment). Running **pip3 install** as a non-`root` user may seem to work, but it will cause the installed module to be inaccessible by other users.

pip3 install will not reinstall an already installed module automatically. When using the **pip3 install** command to upgrade a module (for example, from meson-0.61.3 to meson-0.62.0), insert the option *`--upgrade`* into the command line. If it's really necessary to downgrade a module, or reinstall the same version for some reason, insert *`--force-reinstall --no-deps`* into the command line.

If desired, install the preformatted documentation:

```
install -v -dm755 /usr/share/doc/python-3.12.2/html

tar --no-same-owner \
    -xvf ../python-3.12.2-docs-html.tar.bz2
cp -R --no-preserve=mode python-3.12.2-docs-html/* \
    /usr/share/doc/python-3.12.2/html
```

The meaning of the documentation install commands:

`--no-same-owner` (tar) and `--no-preserve=mode` (cp)
Ensure the installed files have the correct ownership and permissions. Without these options, tar will install the package files with the upstream creator's values and files would have restrictive permissions.

8.51.2. Contents of Python 3

Installed programs: 2to3, idle3, pip3, pydoc3, python3, and python3-config
Installed library: libpython3.12.so and libpython3.so
Installed directories: /usr/include/python3.12, /usr/lib/python3, and /usr/share/doc/python-3.12.2

Short Descriptions

2to3 is a Python program that reads Python 2.x source code and applies a series of fixes to transform it into valid Python 3.x code

idle3 is a wrapper script that opens a Python aware GUI editor. For this script to run, you must have installed Tk before Python, so that the Tkinter Python module is built.

pip3 The package installer for Python. You can use pip to install packages from Python Package Index and other indexes.

pydoc3 is the Python documentation tool

python3 is the interpreter for Python, an interpreted, interactive, object-oriented programming language

8.52. Flit-Core-3.9.0

Flit-core is the distribution-building parts of Flit (a packaging tool for simple Python modules).

Approximate build time: less than 0.1 SBU
Required disk space: 1.6 MB

8.52.1. Installation of Flit-Core

Build the package:

```
pip3 wheel -w dist --no-cache-dir --no-build-isolation --no-deps $PWD
```

Install the package:

```
pip3 install --no-index --no-user --find-links dist flit_core
```

The meaning of the pip3 configuration options and commands:

wheel
This command builds the wheel archive for this package.

`-w dist`
Instructs pip to put the created wheel into the `dist` directory.

`--no-cache-dir`
Prevents pip from copying the created wheel into the `/root/.cache/pip` directory.

install
This command installs the package.

`--no-build-isolation`, *`--no-deps`*, and *`--no-index`*
These options prevent fetching files from the online package repository (PyPI). If packages are installed in the correct order, pip won't need to fetch any files in the first place; these options add some safety in case of user error.

`--find-links dist`
Instructs pip to search for wheel archives in the `dist` directory.

8.52.2. Contents of Flit-Core

Installed directory: /usr/lib/python3.12/site-packages/flit_core and /usr/lib/python3.12/site-packages/flit_core-3.9.0.dist-info

8.53. Wheel-0.42.0

Wheel is a Python library that is the reference implementation of the Python wheel packaging standard.

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

8.53.1. Installation of Wheel

Compile Wheel with the following command:

```
pip3 wheel -w dist --no-cache-dir --no-build-isolation --no-deps $PWD
```

Install Wheel with the following command:

```
pip3 install --no-index --find-links=dist wheel
```

8.53.2. Contents of Wheel

Installed program: wheel
Installed directories: /usr/lib/python3.12/site-packages/wheel and /usr/lib/python3.12/site-packages/wheel-0.42.0.dist-info

Short Descriptions

wheel is a utility to unpack, pack, or convert wheel archives

8.54. Setuptools-69.1.0

Setuptools is a tool used to download, build, install, upgrade, and uninstall Python packages.

Approximate build time: less than 0.1 SBU
Required disk space: 20 MB

8.54.1. Installation of Setuptools

Build the package:

```
pip3 wheel -w dist --no-cache-dir --no-build-isolation --no-deps $PWD
```

Install the package:

```
pip3 install --no-index --find-links dist setuptools
```

8.54.2. Contents of Setuptools

Installed directory: /usr/lib/python3.12/site-packages/_distutils_hack, /usr/lib/python3.12/site-packages/pkg_resources, /usr/lib/python3.12/site-packages/setuptools, and /usr/lib/python3.12/site-packages/setuptools-69.1.0.dist-info

8.55. Ninja-1.11.1

Ninja is a small build system with a focus on speed.

Approximate build time: 0.3 SBU
Required disk space: 75 MB

8.55.1. Installation of Ninja

When run, **ninja** normally utilizes the greatest possible number of processes in parallel. By default this is the number of cores on the system, plus two. This may overheat the CPU, or make the system run out of memory. When **ninja** is invoked from the command line, passing the -jN parameter will limit the number of parallel processes. Some packages embed the execution of **ninja**, and do not pass the -j parameter on to it.

Using the *optional* procedure below allows a user to limit the number of parallel processes via an environment variable, NINJAJOBS. **For example**, setting:

```
export NINJAJOBS=4
```

will limit **ninja** to four parallel processes.

If desired, make **ninja** recognize the environment variable NINJAJOBS by running the stream editor:

```
sed -i '/int Guess/a \
  int   j = 0;\
  char* jobs = getenv( "NINJAJOBS" );\
  if ( jobs != NULL ) j = atoi( jobs );\
  if ( j > 0 ) return j;\
' src/ninja.cc
```

Build Ninja with:

```
python3 configure.py --bootstrap
```

The meaning of the build option:

`--bootstrap`
This parameter forces Ninja to rebuild itself for the current system.

To test the results, issue:

```
./ninja ninja_test
./ninja_test --gtest_filter=-SubprocessTest.SetWithLots
```

Install the package:

```
install -vm755 ninja /usr/bin/
install -vDm644 misc/bash-completion /usr/share/bash-completion/completions/ninja
install -vDm644 misc/zsh-completion  /usr/share/zsh/site-functions/_ninja
```

8.55.2. Contents of Ninja

Installed programs: ninja

Short Descriptions

ninja is the Ninja build system

8.56. Meson-1.3.2

Meson is an open source build system designed to be both extremely fast and as user friendly as possible.

Approximate build time: less than 0.1 SBU
Required disk space: 42 MB

8.56.1. Installation of Meson

Compile Meson with the following command:

```
pip3 wheel -w dist --no-cache-dir --no-build-isolation --no-deps $PWD
```

The test suite requires some packages outside the scope of LFS.

Install the package:

```
pip3 install --no-index --find-links dist meson
install -vDm644 data/shell-completions/bash/meson /usr/share/bash-completion/completions/meson
install -vDm644 data/shell-completions/zsh/_meson /usr/share/zsh/site-functions/_meson
```

The meaning of the install parameters:

`-w dist`
Puts the created wheels into the `dist` directory.

`--find-links dist`
Installs wheels from the `dist` directory.

8.56.2. Contents of Meson

Installed programs: meson
Installed directory: /usr/lib/python3.12/site-packages/meson-1.3.2.dist-info and /usr/lib/python3.12/site-packages/mesonbuild

Short Descriptions

meson A high productivity build system

8.57. Coreutils-9.4

The Coreutils package contains the basic utility programs needed by every operating system.

Approximate build time: 1.0 SBU
Required disk space: 175 MB

8.57.1. Installation of Coreutils

POSIX requires that programs from Coreutils recognize character boundaries correctly even in multibyte locales. The following patch fixes this non-compliance and other internationalization-related bugs.

```
patch -Np1 -i ../coreutils-9.4-i18n-1.patch
```

Note

Many bugs have been found in this patch. When reporting new bugs to the Coreutils maintainers, please check first to see if those bugs are reproducible without this patch.

Fix a security vulnerability in the **split** utility:

```
sed -e '/n_out += n_hold/,+4 s|.*bufsize.*|//&|' \
    -i src/split.c
```

Now prepare Coreutils for compilation:

```
autoreconf -fiv
FORCE_UNSAFE_CONFIGURE=1 ./configure \
            --prefix=/usr            \
            --enable-no-install-program=kill,uptime
```

The meaning of the configure options:

autoreconf
The patch for internationalization has modified the build system, so the configuration files must be regenerated.

`FORCE_UNSAFE_CONFIGURE=1`
This environment variable allows the package to be built by the `root` user.

`--enable-no-install-program=kill,uptime`
The purpose of this switch is to prevent Coreutils from installing programs that will be installed by other packages.

Compile the package:

```
make
```

Skip down to “Install the package” if not running the test suite.

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=tester check-root
```

We're going to run the remainder of the tests as the `tester` user. Certain tests require that the user be a member of more than one group. So that these tests are not skipped, add a temporary group and make the user `tester` a part of it:

```
groupadd -g 102 dummy -U tester
```

Fix some of the permissions so that the non-`root` user can compile and run the tests:

```
chown -R tester .
```

Now run the tests:

```
su tester -c "PATH=$PATH make RUN_EXPENSIVE_TESTS=yes check"
```

Remove the temporary group:

```
groupdel dummy
```

Install the package:

```
make install
```

Move programs to the locations specified by the FHS:

```
mv -v /usr/bin/chroot /usr/sbin
mv -v /usr/share/man/man1/chroot.1 /usr/share/man/man8/chroot.8
sed -i 's/"1"/"8"/' /usr/share/man/man8/chroot.8
```

8.57.2. Contents of Coreutils

Installed programs: [, b2sum, base32, base64, basename, basenc, cat, chcon, 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, id, install, join, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mktemp, mv, nice, nl, nohup, nproc, numfmt, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, realpath, rm, rmdir, runcon, seq, sha1sum, sha224sum, sha256sum, sha384sum, sha512sum, shred, shuf, sleep, sort, split, stat, stdbuf, stty, sum, sync, tac, tail, tee, test, timeout, touch, tr, true, truncate, tsort, tty, uname, unexpand, uniq, unlink, users, vdir, wc, who, whoami, and yes
Installed library: libstdbuf.so (in /usr/libexec/coreutils)
Installed directory: /usr/libexec/coreutils

Short Descriptions

[Is an actual command, /usr/bin/[; it is a synonym for the **test** command

base32 Encodes and decodes data according to the base32 specification (RFC 4648)

base64 Encodes and decodes data according to the base64 specification (RFC 4648)

b2sum Prints or checks BLAKE2 (512-bit) checksums

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

basenc Encodes or decodes data using various algorithms

cat Concatenates files to standard output

chcon Changes security context for files and directories

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 be made, 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 date and time in the given format, or sets the system date and time
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	Extracts the directory portion(s) of the given name(s)
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 the specified integers
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
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), "named pipes" 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
mktemp	Creates temporary files in a secure manner; it is used in scripts

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
nproc	Prints the number of processing units available to a process
numfmt	Converts numbers to or from human-readable strings
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
realpath	Prints the resolved path
rm	Removes files or directories
rmdir	Removes directories if they are empty
runcon	Runs a command with specified security context
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
sha224sum	Prints or checks 224-bit Secure Hash Algorithm checksums
sha256sum	Prints or checks 256-bit Secure Hash Algorithm checksums
sha384sum	Prints or checks 384-bit Secure Hash Algorithm checksums
sha512sum	Prints or checks 512-bit Secure Hash Algorithm checksums
shred	Overwrites the given files repeatedly with complex patterns, making it difficult to recover the data
shuf	Shuffles lines of text
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
stdbuf	Runs commands with altered buffering operations for its standard streams
stty	Sets or reports terminal line settings
sum	Prints checksum and block counts for each given file
sync	Flushes file system buffers; it forces changed blocks to disk and updates the super block

tac	Concatenates the given files in reverse
tail	Prints the last ten lines (or the given number of lines) of each given file
tee	Reads from standard input while writing both to standard output and to the given files
test	Compares values and checks file types
timeout	Runs a command with a time limit
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
truncate	Shrinks or expands a file to the specified size
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 grand totals 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
`libstdbuf`	Library used by **stdbuf**

8.58. Check-0.15.2

Check is a unit testing framework for C.

Approximate build time: 0.1 SBU (about 1.6 SBU with tests)
Required disk space: 12 MB

8.58.1. Installation of Check

Prepare Check for compilation:

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

Build the package:

```
make
```

Compilation is now complete. To run the Check test suite, issue the following command:

```
make check
```

Install the package:

```
make docdir=/usr/share/doc/check-0.15.2 install
```

8.58.2. Contents of Check

Installed program: checkmk
Installed library: libcheck.so

Short Descriptions

checkmk	Awk script for generating C unit tests for use with the Check unit testing framework
`libcheck.so`	Contains functions that allow Check to be called from a test program

8.59. Diffutils-3.10

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

Approximate build time: 0.3 SBU
Required disk space: 36 MB

8.59.1. Installation of Diffutils

Prepare Diffutils for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.59.2. Contents of Diffutils

Installed programs: cmp, diff, diff3, and sdiff

Short Descriptions

cmp Compares two files and reports any differences byte by byte

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

8.60. Gawk-5.3.0

The Gawk package contains programs for manipulating text files.

Approximate build time: 0.1 SBU
Required disk space: 42 MB

8.60.1. Installation of Gawk

First, ensure some unneeded files are not installed:

```
sed -i 's/extras//' Makefile.in
```

Prepare Gawk for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
chown -R tester .
su tester -c "PATH=$PATH make check"
```

Install the package:

```
rm -f /usr/bin/gawk-5.3.0
make install
```

The meaning of the command:

rm -f /usr/bin/gawk-5.3.0
The building system will not recreate the hard link `gawk-5.3.0` if it already exists. Remove it to ensure that the previous hard link installed in Section 6.9, “Gawk-5.3.0” is updated here.

The installation process already created **awk** as a symlink to **gawk**, create its man page as a symlink as well:

```
ln -sv gawk.1 /usr/share/man/man1/awk.1
```

If desired, install the documentation:

```
mkdir -pv                                   /usr/share/doc/gawk-5.3.0
cp    -v doc/{awkforai.txt,*.{eps,pdf,jpg}} /usr/share/doc/gawk-5.3.0
```

8.60.2. Contents of Gawk

Installed programs: awk (link to gawk), gawk, and gawk-5.3.0
Installed libraries: filefuncs.so, fnmatch.so, fork.so, inplace.so, intdiv.so, ordchr.so, readdir.so, readfile.so, revoutput.so, revtwoway.so, rwarray.so, and time.so (all in /usr/lib/gawk)
Installed directories: /usr/lib/gawk, /usr/libexec/awk, /usr/share/awk, and /usr/share/doc/gawk-5.3.0

Short Descriptions

awk A link to **gawk**

gawk A program for manipulating text files; it is the GNU implementation of **awk**

gawk-5.3.0 A hard link to **gawk**

8.61. Findutils-4.9.0

The Findutils package contains programs to find files. Programs are provided to search through all the files in a directory tree and to create, maintain, and search a database (often faster than the recursive find, but unreliable unless the database has been updated recently). Findutils also supplies the **xargs** program, which can be used to run a specified command on each file selected by a search.

Approximate build time: 0.4 SBU
Required disk space: 51 MB

8.61.1. Installation of Findutils

Prepare Findutils for compilation:

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

The meaning of the configure options:

`--localstatedir`
This option moves the **locate** database to `/var/lib/locate`, which is the FHS-compliant location.

Compile the package:

```
make
```

To test the results, issue:

```
chown -R tester .
su tester -c "PATH=$PATH make check"
```

Install the package:

```
make install
```

8.61.2. Contents of Findutils

Installed programs: find, locate, updatedb, and xargs
Installed directory: /var/lib/locate

Short Descriptions

find Searches given directory trees for files matching the specified criteria

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

8.62. Groff-1.23.0

The Groff package contains programs for processing and formatting text and images.

Approximate build time: 0.2 SBU
Required disk space: 107 MB

8.62.1. 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. While the default paper size is configured during compilation, it can be overridden later by echoing either "A4" or "letter" to the /etc/papersize file.

Prepare Groff for compilation:

```
PAGE=<paper_size> ./configure --prefix=/usr
```

Build the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.62.2. Contents of Groff

Installed programs: addftinfo, afmtodit, chem, eqn, eqn2graph, gdiffmk, glilypond, gperl, gpinyin, grap2graph, grn, grodvi, groff, groffer, grog, grolbp, grolj4, gropdf, grops, grotty, hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pdfmom, pdfroff, pfbtops, pic, pic2graph, post-grohtml, preconv, pre-grohtml, refer, roff2dvi, roff2html, roff2pdf, roff2ps, roff2text, roff2x, soelim, tbl, tfmtodit, and troff
Installed directories: /usr/lib/groff and /usr/share/doc/groff-1.23.0, /usr/share/groff

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**

chem Groff preprocessor for producing chemical structure diagrams

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

glilypond Transforms sheet music written in the lilypond language into the groff language

gperl Preprocessor for groff, allowing the insertion of perl code into groff files

gpinyin Preprocessor for groff, allowing the insertion of Pinyin (Mandarin Chinese spelled with the Roman alphabet) into groff files.

grap2graph	Converts a grap program file into a cropped bitmap image (grap is an old Unix programming language for creating diagrams)
grn	A **groff** preprocessor for gremlin files
grodvi	A driver for **groff** that produces TeX dvi format output files
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
gropdf	Translates the output of GNU **troff** to PDF
grops	Translates the output of GNU **troff** to PostScript
grotty	Translates the output of GNU **troff** into a form suitable for typewriter-like devices
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**
pdfmom	Is a wrapper around groff that facilitates the production of PDF documents from files formatted with the mom macros.
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
preconv	Converts encoding of input files to something GNU **troff** understands
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

roff2dvi	Transforms roff files into DVI format
roff2html	Transforms roff files into HTML format
roff2pdf	Transforms roff files into PDFs
roff2ps	Transforms roff files into ps files
roff2text	Transforms roff files into text files
roff2x	Transforms roff files into other formats
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

8.63. GRUB-2.12

The GRUB package contains the GRand Unified Bootloader.

Approximate build time: 0.3 SBU
Required disk space: 166 MB

8.63.1. Installation of GRUB

Note

If your system has UEFI support and you wish to boot LFS with UEFI, you can skip this package in LFS, and install GRUB with UEFI support (and its dependencies) by following the instructions on *the BLFS page*.

Warning

Unset any environment variables which may affect the build:

```
unset {C,CPP,CXX,LD}FLAGS
```

Don't try “tuning” this package with custom compilation flags. This package is a bootloader. The low-level operations in the source code may be broken by aggressive optimization.

Add a file missing from the release tarball:

```
echo depends bli part_gpt > grub-core/extra_deps.lst
```

Prepare GRUB for compilation:

```
./configure --prefix=/usr          \
            --sysconfdir=/etc      \
            --disable-efiemu       \
            --disable-werror
```

The meaning of the new configure options:

`--disable-werror`
This allows the build to complete with warnings introduced by more recent versions of Flex.

`--disable-efiemu`
This option minimizes what is built by disabling a feature and eliminating some test programs not needed for LFS.

Compile the package:

```
make
```

The test suite for this packages is not recommended. Most of the tests depend on packages that are not available in the limited LFS environment. To run the tests anyway, run **make check**.

Install the package:

```
make install
mv -v /etc/bash_completion.d/grub /usr/share/bash-completion/completions
```

Making your LFS system bootable with GRUB will be discussed in Section 10.4, “Using GRUB to Set Up the Boot Process”.

8.63.2. Contents of GRUB

Installed programs: grub-bios-setup, grub-editenv, grub-file, grub-fstest, grub-glue-efi, grub-install, grub-kbdcomp, grub-macbless, grub-menulst2cfg, grub-mkconfig, grub-mkimage, grub-mklayout, grub-mknetdir, grub-mkpasswd-pbkdf2, grub-mkrelpath, grub-mkrescue, grub-mkstandalone, grub-ofpathname, grub-probe, grub-reboot, grub-render-label, grub-script-check, grub-set-default, grub-sparc64-setup, and grub-syslinux2cfg

Installed directories: /usr/lib/grub, /etc/grub.d, /usr/share/grub, and /boot/grub (when grub-install is first run)

Short Descriptions

grub-bios-setup Is a helper program for **grub-install**

grub-editenv Is a tool to edit the environment block

grub-file Checks to see if the given file is of the specified type

grub-fstest Is a tool to debug the filesystem driver

grub-glue-efi Glues 32-bit and 64-bit binaries into a single file (for Apple machines)

grub-install Installs GRUB on your drive

grub-kbdcomp Is a script that converts an xkb layout into one recognized by GRUB

grub-macbless Is the Mac-style bless for HFS or HFS+ file systems (**bless** is peculiar to Apple machines; it makes a device bootable)

grub-menulst2cfg Converts a GRUB Legacy `menu.lst` into a `grub.cfg` for use with GRUB 2

grub-mkconfig Generates a `grub.cfg` file

grub-mkimage Makes a bootable image of GRUB

grub-mklayout Generates a GRUB keyboard layout file

grub-mknetdir Prepares a GRUB netboot directory

grub-mkpasswd-pbkdf2 Generates an encrypted PBKDF2 password for use in the boot menu

grub-mkrelpath Makes a system pathname relative to its root

grub-mkrescue Makes a bootable image of GRUB suitable for a floppy disk, CDROM/DVD, or a USB drive

grub-mkstandalone Generates a standalone image

grub-ofpathname Is a helper program that prints the path to a GRUB device

grub-probe Probes device information for a given path or device

grub-reboot Sets the default boot entry for GRUB for the next boot only

grub-render-label Renders Apple .disk_label for Apple Macs

grub-script-check Checks the GRUB configuration script for syntax errors

grub-set-default Sets the default boot entry for GRUB

grub-sparc64-setup Is a helper program for grub-setup

grub-syslinux2cfg Transforms a syslinux config file into grub.cfg format

8.64. Gzip-1.13

The Gzip package contains programs for compressing and decompressing files.

Approximate build time: 0.3 SBU
Required disk space: 21 MB

8.64.1. Installation of Gzip

Prepare Gzip for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.64.2. Contents of Gzip

Installed programs: gunzip, gzexe, gzip, uncompress (hard link with gunzip), zcat, zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore, and znew

Short Descriptions

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`

8.65. IPRoute2-6.7.0

The IPRoute2 package contains programs for basic and advanced IPV4-based networking.

Approximate build time: 0.1 SBU
Required disk space: 17 MB

8.65.1. Installation of IPRoute2

The **arpd** program included in this package will not be built since it depends on Berkeley DB, which is not installed in LFS. However, a directory and a man page for **arpd** will still be installed. Prevent this by running the commands shown below.

```
sed -i /ARPD/d Makefile
rm -fv man/man8/arpd.8
```

Compile the package:

```
make NETNS_RUN_DIR=/run/netns
```

This package does not have a working test suite.

Install the package:

```
make SBINDIR=/usr/sbin install
```

If desired, install the documentation:

```
mkdir -pv             /usr/share/doc/iproute2-6.7.0
cp -v COPYING README* /usr/share/doc/iproute2-6.7.0
```

8.65.2. Contents of IPRoute2

Installed programs: bridge, ctstat (link to lnstat), genl, ifstat, ip, lnstat, nstat, routel, rtacct, rtmon, rtpr, rtstat (link to lnstat), ss, and tc
Installed directories: /etc/iproute2, /usr/lib/tc, and /usr/share/doc/iproute2-6.7.0

Short Descriptions

bridge Configures network bridges

ctstat Connection status utility

genl Generic netlink utility front end

ifstat Shows interface statistics, including the number of packets transmitted and received, by interface

ip The main executable. It has several different functions, including these:
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 continuously 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 Displays network statistics

routel A component of **ip route**, 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** into a readable form

rtstat Route status utility

ss Similar to the **netstat** command; shows active connections

tc Traffic control for Quality of Service (QoS) and Class of Service (CoS) implementations
tc qdisc allows users to set up the queueing discipline
tc class allows users to set up classes based on the queuing discipline scheduling
tc filter allows users to set up the QoS/CoS packet filtering
tc monitor can be used to view changes made to Traffic Control in the kernel.

8.66. Kbd-2.6.4

The Kbd package contains key-table files, console fonts, and keyboard utilities.

Approximate build time: 0.1 SBU
Required disk space: 34 MB

8.66.1. Installation of Kbd

The behavior of the backspace and delete keys is not consistent across the keymaps in the Kbd package. The following patch fixes this issue for i386 keymaps:

```
patch -Np1 -i ../kbd-2.6.4-backspace-1.patch
```

After patching, the backspace key generates the character with code 127, and the delete key generates a well-known escape sequence.

Remove the redundant **resizecons** program (it requires the defunct svgalib to provide the video mode files - for normal use **setfont** sizes the console appropriately) together with its manpage.

```
sed -i '/RESIZECONS_PROGS=/s/yes/no/' configure
sed -i 's/resizecons.8 //' docs/man/man8/Makefile.in
```

Prepare Kbd for compilation:

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

The meaning of the configure option:

`--disable-vlock`
This option prevents the vlock utility from being built because it requires the PAM library, which isn't available in the chroot environment.

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

Note

For some languages (e.g., Belarusian) the Kbd package doesn't provide a useful keymap where the stock “by” keymap assumes the ISO-8859-5 encoding, and the CP1251 keymap is normally used. Users of such languages have to download working keymaps separately.

If desired, install the documentation:

```
cp -R -v docs/doc -T /usr/share/doc/kbd-2.6.4
```

8.66.2. Contents of Kbd

Installed programs: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbdinfo, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (link to psfxtable), psfgettable (link to psfxtable), psfstriptable (link to psfxtable), psfxtable, setfont, setkeycodes, setleds, setmetamode, setvtrgb, showconsolefont, showkey, unicode_start, and unicode_stop

Installed directories: /usr/share/consolefonts, /usr/share/consoletrans, /usr/share/keymaps, /usr/share/doc/kbd-2.6.4, and /usr/share/unimaps

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
kbdinfo	Obtains information about the status of a console
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	Adds a Unicode character table to a console font
psfgettable	Extracts the embedded Unicode character table from a console font
psfstriptable	Removes the embedded Unicode character table from a console font
psfxtable	Handles Unicode character tables for console fonts
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
setvtrgb	Sets the console color map in all virtual terminals
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 [Don't use this program unless your keymap file is in the ISO-8859-1 encoding. For other encodings, this utility produces incorrect results.]
unicode_stop	Reverts keyboard and console from UNICODE mode

8.67. Libpipeline-1.5.7

The Libpipeline package contains a library for manipulating pipelines of subprocesses in a flexible and convenient way.

Approximate build time: 0.1 SBU
Required disk space: 10 MB

8.67.1. Installation of Libpipeline

Prepare Libpipeline for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.67.2. Contents of Libpipeline

Installed library: libpipeline.so

Short Descriptions

`libpipeline` This library is used to safely construct pipelines between subprocesses

8.68. Make-4.4.1

The Make package contains a program for controlling the generation of executables and other non-source files of a package from source files.

Approximate build time: 0.5 SBU
Required disk space: 13 MB

8.68.1. Installation of Make

Prepare Make for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
chown -R tester .
su tester -c "PATH=$PATH make check"
```

Install the package:

```
make install
```

8.68.2. 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

8.69. Patch-2.7.6

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

Approximate build time: 0.1 SBU
Required disk space: 12 MB

8.69.1. Installation of Patch

Prepare Patch for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.69.2. 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.)

8.70. Tar-1.35

The Tar package provides the ability to create tar archives as well as perform various other kinds of archive manipulation. Tar can be used on previously created archives to extract files, to store additional files, or to update or list files which were already stored.

Approximate build time: 0.5 SBU
Required disk space: 43 MB

8.70.1. Installation of Tar

Prepare Tar for compilation:

```
FORCE_UNSAFE_CONFIGURE=1  \
./configure --prefix=/usr
```

The meaning of the configure option:

`FORCE_UNSAFE_CONFIGURE=1`
This forces the test for `mknod` to be run as `root`. It is generally considered dangerous to run this test as the `root` user, but as it is being run on a system that has only been partially built, overriding it is OK.

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

One test, capabilities: binary store/restore, is known to fail if it is run because LFS lacks selinux, but will be skipped if the host kernel does not support extended attributes or security labels on the filesystem used for building LFS.

Install the package:

```
make install
make -C doc install-html docdir=/usr/share/doc/tar-1.35
```

8.70.2. Contents of Tar

Installed programs: tar
Installed directory: /usr/share/doc/tar-1.35

Short Descriptions

tar Creates, extracts files from, and lists the contents of archives, also known as tarballs

8.71. Texinfo-7.1

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

Approximate build time: 0.3 SBU
Required disk space: 139 MB

8.71.1. Installation of Texinfo

Prepare Texinfo for compilation:

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

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

Optionally, install the components belonging in a TeX installation:

```
make TEXMF=/usr/share/texmf install-tex
```

The meaning of the make parameter:

TEXMF=/usr/share/texmf
The TEXMF makefile variable holds the location of the root of the TeX tree if, for example, a TeX package will be installed later.

The Info documentation system uses a plain text file to hold its list of menu entries. The file is located at /usr/share/info/dir. Unfortunately, due to occasional problems in the Makefiles of various packages, it can sometimes get out of 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:

```
pushd /usr/share/info
  rm -v dir
  for f in *
    do install-info $f dir 2>/dev/null
  done
popd
```

8.71.2. Contents of Texinfo

Installed programs: info, install-info, makeinfo (link to texi2any), pdftexi2dvi, pod2texi, texi2any, texi2dvi, texi2pdf, and texindex
Installed library: MiscXS.so, Parsetexi.so, and XSParagraph.so (all in /usr/lib/texinfo)
Installed directories: /usr/share/texinfo and /usr/lib/texinfo

Short Descriptions

info Used to read info pages which are similar to man pages, but often go much deeper than just explaining all the available command line options [For example, compare **man bison** and **info bison**.]

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
pdftexi2dvi	Used to format the given Texinfo document into a Portable Document Format (PDF) file
pod2texi	Converts Pod to Texinfo format
texi2any	Translate Texinfo source documentation to various other formats
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

8.72. Vim-9.1.0041

The Vim package contains a powerful text editor.

Approximate build time: 2.5 SBU
Required disk space: 236 MB

Alternatives to Vim

If you prefer another editor—such as Emacs, Joe, or Nano—please refer to *https://www.linuxfromscratch.org/blfs/view/12.1/postlfs/editors.html* for suggested installation instructions.

8.72.1. Installation of Vim

First, 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
```

Compile the package:

```
make
```

To prepare the tests, ensure that user `tester` can write to the source tree:

```
chown -R tester .
```

Now run the tests as user `tester`:

```
su tester -c "TERM=xterm-256color LANG=en_US.UTF-8 make -j1 test" \
   &> vim-test.log
```

The test suite outputs a lot of binary data to the screen. This can cause issues with the settings of the current terminal (especially while we are overriding the `TERM` variable to satisify some assumptions of the test suite). The problem can be avoided by redirecting the output to a log file as shown above. A successful test will result in the words `ALL DONE` in the log file at completion.

Install the package:

```
make install
```

Many users reflexively type **vi** instead of **vim**. To allow execution of **vim** when users habitually enter **vi**, create a symlink for both the binary and the man page in the provided languages:

```
ln -sv vim /usr/bin/vi
for L in  /usr/share/man/{,*/}man1/vim.1; do
    ln -sv vim.1 $(dirname $L)/vi.1
done
```

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-9.1.0041`, making it consistent with the location of documentation for other packages:

```
ln -sv ../vim/vim91/doc /usr/share/doc/vim-9.1.0041
```

If an X Window System is going to be installed on the LFS system, it may be necessary to recompile Vim after installing X. Vim comes with a GUI version of the editor that requires X and some additional libraries to be installed. For more information on this process, refer to the Vim documentation and the Vim installation page in the BLFS book at *https://www.linuxfromscratch.org/blfs/view/12.1/postlfs/vim.html*.

8.72.2. 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

" Ensure defaults are set before customizing settings, not after
source $VIMRUNTIME/defaults.vim
let skip_defaults_vim=1

set nocompatible
set backspace=2
set mouse=
syntax on
if (&term == "xterm") || (&term == "putty")
  set background=dark
endif

" End /etc/vimrc
EOF
```

The *set nocompatible* setting 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* setting allows backspacing over line breaks, autoindents, and the start of an insert. The *syntax on* parameter enables vim's syntax highlighting. The *set mouse=* setting enables proper pasting of text with the mouse when working in chroot or over a remote connection. Finally, the *if* statement with the *set background=dark* setting 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'
```

Note

By default, vim only installs spell-checking files for the English language. To install spell-checking files for your preferred language, copy the `.spl` and optionally, the `.sug` files for your language and character encoding from `runtime/spell` into `/usr/share/vim/vim91/spell/`.

To use these spell-checking files, some configuration in `/etc/vimrc` is needed, e.g.:

```
set spelllang=en,ru
set spell
```

For more information, see `runtime/spell/README.txt`.

8.72.3. Contents of Vim

Installed programs: ex (link to vim), rview (link to vim), rvim (link to vim), vi (link to vim), view (link to vim), vim, vimdiff (link to vim), vimtutor, and xxd

Installed directory: /usr/share/vim

Short Descriptions

ex	Starts **vim** in ex mode
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
vi	Link to **vim**
view	Starts **vim** in read-only mode
vim	Is the editor
vimdiff	Edits two or three versions of a file with **vim** and shows differences
vimtutor	Teaches the basic keys and commands of **vim**
xxd	Creates a hex dump of the given file; it can also perform the inverse operation, so it can be used for binary patching

8.73. MarkupSafe-2.1.5

MarkupSafe is a Python module that implements an XML/HTML/XHTML Markup safe string.

Approximate build time: less than 0.1 SBU
Required disk space: 508 KB

8.73.1. Installation of MarkupSafe

Compile MarkupSafe with the following command:

```
pip3 wheel -w dist --no-cache-dir --no-build-isolation --no-deps $PWD
```

This package does not come with a test suite.

Install the package:

```
pip3 install --no-index --no-user --find-links dist Markupsafe
```

8.73.2. Contents of MarkupSafe

Installed directory: /usr/lib/python3.12/site-packages/MarkupSafe-2.1.5.dist-info

8.74. Jinja2-3.1.3

Jinja2 is a Python module that implements a simple pythonic template language.

Approximate build time: less than 0.1 SBU
Required disk space: 3.2 MB

8.74.1. Installation of Jinja2

Build the package:

```
pip3 wheel -w dist --no-cache-dir --no-build-isolation --no-deps $PWD
```

Install the package:

```
pip3 install --no-index --no-user --find-links dist Jinja2
```

8.74.2. Contents of Jinja2

Installed directory: /usr/lib/python3.12/site-packages/Jinja2-3.1.3.dist-info

8.75. Udev from Systemd-255

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

Approximate build time: 0.2 SBU
Required disk space: 144 MB

8.75.1. Installation of Udev

Udev is part of the systemd-255 package. Use the systemd-255.tar.xz file as the source tarball.

Remove two unneeded groups, `render` and `sgx`, from the default udev rules:

```
sed -i -e 's/GROUP="render"/GROUP="video"/' \
       -e 's/GROUP="sgx", //' rules.d/50-udev-default.rules.in
```

Remove one udev rule requiring a full Systemd installation:

```
sed '/systemd-sysctl/s/^/#/' -i rules.d/99-systemd.rules.in
```

Adjust the hardcoded paths to network configuration files for the standalone udev installation:

```
sed '/NETWORK_DIRS/s/systemd/udev/' -i src/basic/path-lookup.h
```

Prepare Udev for compilation:

```
mkdir -p build
cd       build

meson setup \
      --prefix=/usr                 \
      --buildtype=release           \
      -Dmode=release                \
      -Ddev-kvm-mode=0660           \
      -Dlink-udev-shared=false      \
      -Dlogind=false                \
      -Dvconsole=false              \
      ..
```

The meaning of the meson options:

`--buildtype=release`
This switch overrides the default buildtype (“debug”), which produces unoptimized binaries.

`-Dmode=release`
Disable some features considered experimental by upstream.

`-Ddev-kvm-mode=0660`
The default udev rule would allow all users to access `/dev/kvm`. The editors consider it dangerous. This option overrides it.

`-Dlink-udev-shared=false`
This option prevents udev from linking to the internal systemd shared library, `libsystemd-shared`. This library is designed to be shared by many Systemd components and it's too overkill for a udev-only installation.

`-Dlogind=false -Dvconsole=false`
These options prevent the generation of several udev rule files belonging to the other Systemd components that we won't install.

Get the list of the shipped udev helpers and save it into an environment variable (exporting it is not strictly necessary, but it makes building as a regular user or using a package manager easier):

```
export udev_helpers=$(grep "'name' :" ../src/udev/meson.build | \
                      awk '{print $3}' | tr -d ",'" | grep -v 'udevadm')
```

Only build the components needed for udev:

```
ninja udevadm systemd-hwdb                                          \
      $(ninja -n | grep -Eo '(src/(lib)?udev|rules.d|hwdb.d)/[^ ]*') \
      $(realpath libudev.so --relative-to .)                        \
      $udev_helpers
```

Install the package:

```
install -vm755 -d {/usr/lib,/etc}/udev/{hwdb.d,rules.d,network}
install -vm755 -d /usr/{lib,share}/pkgconfig
install -vm755 udevadm                             /usr/bin/
install -vm755 systemd-hwdb                        /usr/bin/udev-hwdb
ln      -svfn  ../bin/udevadm                      /usr/sbin/udevd
cp      -av    libudev.so{,*[0-9]}                 /usr/lib/
install -vm644 ../src/libudev/libudev.h            /usr/include/
install -vm644 src/libudev/*.pc                    /usr/lib/pkgconfig/
install -vm644 src/udev/*.pc                       /usr/share/pkgconfig/
install -vm644 ../src/udev/udev.conf               /etc/udev/
install -vm644 rules.d/* ../rules.d/README         /usr/lib/udev/rules.d/
install -vm644 $(find ../rules.d/*.rules \
                      -not -name '*power-switch*') /usr/lib/udev/rules.d/
install -vm644 hwdb.d/*  ../hwdb.d/{*.hwdb,README} /usr/lib/udev/hwdb.d/
install -vm755 $udev_helpers                       /usr/lib/udev
install -vm644 ../network/99-default.link          /usr/lib/udev/network
```

Install some custom rules and support files useful in an LFS environment:

```
tar -xvf ../../udev-lfs-20230818.tar.xz
make -f udev-lfs-20230818/Makefile.lfs install
```

Install the man pages:

```
tar -xf ../../systemd-man-pages-255.tar.xz                          \
    --no-same-owner --strip-components=1                            \
    -C /usr/share/man --wildcards '*/udev*' '*/libudev*'            \
                                  '*/systemd.link.5'                \
                                  '*/systemd-'{hwdb,udevd.service}.8

sed 's|systemd/network|udev/network|'                               \
    /usr/share/man/man5/systemd.link.5                              \
  > /usr/share/man/man5/udev.link.5

sed 's/systemd\(\\?-\)/udev\1/' /usr/share/man/man8/systemd-hwdb.8  \
                              > /usr/share/man/man8/udev-hwdb.8

sed 's|lib.*udevd|sbin/udevd|'                                      \
    /usr/share/man/man8/systemd-udevd.service.8                     \
  > /usr/share/man/man8/udevd.8

rm /usr/share/man/man*/systemd*
```

Finally, unset the `udev_helpers` variable:

```
unset udev_helpers
```

8.75.2. Configuring Udev

Information about hardware devices is maintained in the `/etc/udev/hwdb.d` and `/usr/lib/udev/hwdb.d` directories. Udev needs that information to be compiled into a binary database `/etc/udev/hwdb.bin`. Create the initial database:

```
udev-hwdb update
```

This command needs to be run each time the hardware information is updated.

8.75.3. Contents of Udev

Installed programs: udevadm, udevd (symlink to udevadm), and udev-hwdb
Installed libraries: libudev.so
Installed directories: /etc/udev and /usr/lib/udev

Short Descriptions

udevadm Generic udev administration tool: controls the udevd daemon, provides info from the Udev database, monitors uevents, waits for uevents to finish, tests Udev configuration, and triggers uevents for a given device

udevd A daemon that listens for uevents on the netlink socket, creates devices and runs the configured external programs in response to these uevents

udev-hwdb Updates or queries the hardware database.

`libudev` A library interface to udev device information

`/etc/udev` Contains Udev configuration files, device permissions, and rules for device naming

8.76. Man-DB-2.12.0

The Man-DB package contains programs for finding and viewing man pages.

Approximate build time: 0.2 SBU
Required disk space: 41 MB

8.76.1. Installation of Man-DB

Prepare Man-DB for compilation:

```
./configure --prefix=/usr                         \
            --docdir=/usr/share/doc/man-db-2.12.0 \
            --sysconfdir=/etc                     \
            --disable-setuid                      \
            --enable-cache-owner=bin              \
            --with-browser=/usr/bin/lynx          \
            --with-vgrind=/usr/bin/vgrind         \
            --with-grap=/usr/bin/grap             \
            --with-systemdtmpfilesdir=            \
            --with-systemdsystemunitdir=
```

The meaning of the configure options:

`--disable-setuid`
This disables making the **man** program setuid to user `man`.

`--enable-cache-owner=bin`
This changes ownership of the system-wide cache files to user `bin`.

`--with-...`
These three parameters are used to set some default programs. **lynx** is a text-based web browser (see BLFS for installation instructions), **vgrind** converts program sources to Groff input, and **grap** is useful for typesetting graphs in Groff documents. The **vgrind** and **grap** programs are not normally needed for viewing manual pages. They are not part of LFS or BLFS, but you should be able to install them yourself after finishing LFS if you wish to do so.

`--with-systemd...`
These parameters prevent installing unneeded systemd directories and files.

Compile the package:

```
make
```

To test the results, issue:

```
make check
```

Install the package:

```
make install
```

8.76.2. Non-English Manual Pages in LFS

The following table shows the character set that Man-DB assumes manual pages installed under `/usr/share/man/<ll>` will be encoded with. In addition to this, Man-DB correctly determines if manual pages installed in that directory are UTF-8 encoded.

Table 8.1. Expected character encoding of legacy 8-bit manual pages

Language (code)	Encoding	Language (code)	Encoding
Danish (da)	ISO-8859-1	Croatian (hr)	ISO-8859-2
German (de)	ISO-8859-1	Hungarian (hu)	ISO-8859-2
English (en)	ISO-8859-1	Japanese (ja)	EUC-JP
Spanish (es)	ISO-8859-1	Korean (ko)	EUC-KR
Estonian (et)	ISO-8859-1	Lithuanian (lt)	ISO-8859-13
Finnish (fi)	ISO-8859-1	Latvian (lv)	ISO-8859-13
French (fr)	ISO-8859-1	Macedonian (mk)	ISO-8859-5
Irish (ga)	ISO-8859-1	Polish (pl)	ISO-8859-2
Galician (gl)	ISO-8859-1	Romanian (ro)	ISO-8859-2
Indonesian (id)	ISO-8859-1	Greek (el)	ISO-8859-7
Icelandic (is)	ISO-8859-1	Slovak (sk)	ISO-8859-2
Italian (it)	ISO-8859-1	Slovenian (sl)	ISO-8859-2
Norwegian Bokmal (nb)	ISO-8859-1	Serbian Latin (sr@latin)	ISO-8859-2
Dutch (nl)	ISO-8859-1	Serbian (sr)	ISO-8859-5
Norwegian Nynorsk (nn)	ISO-8859-1	Turkish (tr)	ISO-8859-9
Norwegian (no)	ISO-8859-1	Ukrainian (uk)	KOI8-U
Portuguese (pt)	ISO-8859-1	Vietnamese (vi)	TCVN5712-1
Swedish (sv)	ISO-8859-1	Simplified Chinese (zh_CN)	GBK
Belarusian (be)	CP1251	Simplified Chinese, Singapore (zh_SG)	GBK
Bulgarian (bg)	CP1251	Traditional Chinese, Hong Kong (zh_HK)	BIG5HKSCS
Czech (cs)	ISO-8859-2	Traditional Chinese (zh_TW)	BIG5

Note

Manual pages in languages not in the list are not supported.

8.76.3. Contents of Man-DB

Installed programs: accessdb, apropos (link to whatis), catman, lexgrog, man, man-recode, mandb, manpath, and whatis
Installed libraries: libman.so and libmandb.so (both in /usr/lib/man-db)
Installed directories: /usr/lib/man-db, /usr/libexec/man-db, and /usr/share/doc/man-db-2.12.0

Short Descriptions

accessdb Dumps the **whatis** database contents in human-readable form

apropos	Searches the **whatis** database and displays the short descriptions of system commands that contain a given string
catman	Creates or updates the pre-formatted manual pages
lexgrog	Displays one-line summary information about a given manual page
man	Formats and displays the requested manual page
man-recode	Converts manual pages to another encoding
mandb	Creates or updates the **whatis** database
manpath	Displays the contents of $MANPATH or (if $MANPATH is not set) a suitable search path based on the settings in man.conf and the user's environment
whatis	Searches the **whatis** database and displays the short descriptions of system commands that contain the given keyword as a separate word
`libman`	Contains run-time support for **man**
`libmandb`	Contains run-time support for **man**

8.77. Procps-ng-4.0.4

The Procps-ng package contains programs for monitoring processes.

Approximate build time: 0.1 SBU
Required disk space: 27 MB

8.77.1. Installation of Procps-ng

Prepare Procps-ng for compilation:

```
./configure --prefix=/usr                           \
            --docdir=/usr/share/doc/procps-ng-4.0.4 \
            --disable-static                        \
            --disable-kill
```

The meaning of the configure option:

`--disable-kill`
This switch disables building the **kill** command; it will be installed from the Util-linux package.

Compile the package:

```
make
```

To run the test suite, run:

```
make -k check
```

One test named `ps with output flag bsdtime,cputime,etime,etimes` is known to fail if the host kernel is not built with `CONFIG_BSD_PROCESS_ACCT` enabled. Two tests named `pmap X with unreachable process` and `pmap XX with unreachable process` are known to fail occasionally.

Install the package:

```
make install
```

8.77.2. Contents of Procps-ng

Installed programs: free, pgrep, pidof, pkill, pmap, ps, pwdx, slabtop, sysctl, tload, top, uptime, vmstat, w, and watch
Installed library: libproc-2.so
Installed directories: /usr/include/procps and /usr/share/doc/procps-ng-4.0.4

Short Descriptions

free Reports the amount of free and used memory (both physical and swap memory) in the system
pgrep Looks up processes based on their name and other attributes
pidof Reports the PIDs of the given programs
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

slabtop	Displays detailed kernel slab cache information in real time
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-2`	Contains the functions used by most programs in this package

8.78. Util-linux-2.39.3

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

Approximate build time: 0.5 SBU
Required disk space: 313 MB

8.78.1. Installation of Util-linux

First, disable a problematic test:

```
sed -i '/test_mkfds/s/^/#/' tests/helpers/Makemodule.am
```

Prepare Util-linux for compilation:

```
./configure --bindir=/usr/bin    \
            --libdir=/usr/lib    \
            --runstatedir=/run   \
            --sbindir=/usr/sbin  \
            --disable-chfn-chsh  \
            --disable-login      \
            --disable-nologin    \
            --disable-su         \
            --disable-setpriv    \
            --disable-runuser    \
            --disable-pylibmount \
            --disable-static     \
            --without-python     \
            --without-systemd    \
            --without-systemdsystemunitdir        \
            ADJTIME_PATH=/var/lib/hwclock/adjtime \
            --docdir=/usr/share/doc/util-linux-2.39.3
```

The --disable and --without options prevent warnings about building components that either require packages not in LFS, or are inconsistent with programs installed by other packages.

Compile the package:

```
make
```

If desired, run the test suite as a non-`root` user:

Warning

Running the test suite as the `root` user can be harmful to your system. To run it, the CONFIG_SCSI_DEBUG option for the kernel must be available in the currently running system and must be built as a module. Building it into the kernel will prevent booting. For complete coverage, other BLFS packages must be installed. If desired, this test can be run by booting into the completed LFS system and running:

```
bash tests/run.sh --srcdir=$PWD --builddir=$PWD
```

```
chown -R tester .
su tester -c "make -k check"
```

The *hardlink* tests will fail if the host's kernel does not have the option `CONFIG_CRYPTO_USER_API_HASH` enabled or does not have any options providing a SHA256 implementation (for example, `CONFIG_CRYPTO_SHA256`, or `CONFIG_CRYPTO_SHA256_SSSE3` if the CPU supports Supplemental SSE3) enabled. In addition, two sub-tests from misc: mbsencode and one sub-test from script: replay are known to fail.

Install the package:

```
make install
```

8.78.2. Contents of Util-linux

Installed programs: addpart, agetty, blkdiscard, blkid, blkzone, blockdev, cal, cfdisk, chcpu, chmem, choom, chrt, col, colcrt, colrm, column, ctrlaltdel, delpart, dmesg, eject, fallocate, fdisk, fincore, findfs, findmnt, flock, fsck, fsck.cramfs, fsck.minix, fsfreeze, fstrim, getopt, hardlink, hexdump, hwclock, i386 (link to setarch), ionice, ipcmk, ipcrm, ipcs, irqtop, isosize, kill, last, lastb (link to last), ldattach, linux32 (link to setarch), linux64 (link to setarch), logger, look, losetup, lsblk, lscpu, lsipc, lsirq, lsfd, lslocks, lslogins, lsmem, lsns, mcookie, mesg, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, mountpoint, namei, nsenter, partx, pivot_root, prlimit, readprofile, rename, renice, resizepart, rev, rfkill, rtcwake, script, scriptlive, scriptreplay, setarch, setsid, setterm, sfdisk, sulogin, swaplabel, swapoff, swapon, switch_root, taskset, uclampset, ul, umount, uname26 (link to setarch), unshare, utmpdump, uuidd, uuidgen, uuidparse, wall, wdctl, whereis, wipefs, x86_64 (link to setarch), and zramctl

Installed libraries: libblkid.so, libfdisk.so, libmount.so, libsmartcols.so, and libuuid.so

Installed directories: /usr/include/blkid, /usr/include/libfdisk, /usr/include/libmount, /usr/include/libsmartcols, /usr/include/uuid, /usr/share/doc/util-linux-2.39.3, and /var/lib/hwclock

Short Descriptions

addpart Informs the Linux kernel of new partitions

agetty Opens a tty port, prompts for a login name, and then invokes the **login** program

blkdiscard Discards sectors on a device

blkid A command line utility to locate and print block device attributes

blkzone Is used to manage zoned storage block devices

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

chcpu Modifies the state of CPUs

chmem Configures memory

choom Displays and adjusts OOM-killer scores, used to determine which process to kill first when Linux is Out Of Memory

chrt Manipulates real-time attributes of a process

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

delpart Asks the Linux kernel to remove a partition

dmesg Dumps the kernel boot messages

eject Ejects removable media

fallocate Preallocates space to a file

fdisk Manipulates the partition table of the given device

fincore Counts pages of file contents in core

findfs Finds a file system, either by label or Universally Unique Identifier (UUID)

findmnt Is a command line interface to the libmount library for working with mountinfo, fstab and mtab files

flock Acquires a file lock and then executes a command with the lock held

fsck Is used to check, and optionally repair, file systems

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

fsfreeze Is a very simple wrapper around FIFREEZE/FITHAW ioctl kernel driver operations

fstrim Discards unused blocks on a mounted filesystem

getopt Parses options in the given command line

hardlink Consolidates duplicate files by creating hard links

hexdump Dumps the given file in hexadecimal, decimal, octal, or ascii

hwclock Reads or sets the system's hardware clock, also called the Real-Time Clock (RTC) or Basic Input-Output System (BIOS) clock

i386 A symbolic link to setarch

ionice Gets or sets the io scheduling class and priority for a program

ipcmk Creates various IPC resources

ipcrm Removes the given Inter-Process Communication (IPC) resource

ipcs Provides IPC status information

irqtop Displays kernel interrupt counter information in *top(1)* style view

isosize Reports the size of an iso9660 file system

kill Sends signals to processes

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`

ldattach Attaches a line discipline to a serial line

linux32 A symbolic link to setarch

linux64 A symbolic link to setarch

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

lsblk Lists information about all or selected block devices in a tree-like format

lscpu	Prints CPU architecture information
lsfd	Displays information about open files; replaces **lsof**
lsipc	Prints information on IPC facilities currently employed in the system
lsirq	Displays kernel interrupt counter information
lslocks	Lists local system locks
lslogins	Lists information about users, groups and system accounts
lsmem	Lists the ranges of available memory with their online status
lsns	Lists namespaces
mcookie	Generates magic cookies (128-bit random hexadecimal numbers) for **xauth**
mesg	Controls whether other users can send messages to the current user's terminal
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
mountpoint	Checks if the directory is a mountpoint
namei	Shows the symbolic links in the given paths
nsenter	Runs a program with namespaces of other processes
partx	Tells the kernel about the presence and numbering of on-disk partitions
pivot_root	Makes the given file system the new root file system of the current process
prlimit	Gets and sets a process's resource limits
readprofile	Reads kernel profiling information
rename	Renames the given files, replacing a given string with another
renice	Alters the priority of running processes
resizepart	Asks the Linux kernel to resize a partition
rev	Reverses the lines of a given file
rfkill	Tool for enabling and disabling wireless devices
rtcwake	Used to enter a system sleep state until the specified wakeup time
script	Makes a typescript of a terminal session
scriptlive	Re-runs session typescripts using timing information
scriptreplay	Plays back typescripts using timing information
setarch	Changes reported architecture in a new program environment, and sets personality flags
setsid	Runs the given program in a new session
setterm	Sets terminal attributes

sfdisk A disk partition table manipulator

sulogin Allows `root` to log in; it is normally invoked by **init** when the system goes into single user mode

swaplabel Makes changes to the swap area's UUID and label

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

switch_root Switches to another filesystem as the root of the mount tree

taskset Retrieves or sets a process's CPU affinity

uclampset Manipulates the utilization clamping attributes of the system or a process

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

uname26 A symbolic link to setarch

unshare Runs a program with some namespaces unshared from parent

utmpdump Displays the content of the given login file in a user-friendly format

uuidd A daemon used by the UUID library to generate time-based UUIDs in a secure and guaranteed-unique fashion

uuidgen Creates new UUIDs. Each new UUID is a random number likely to be unique among all UUIDs created, on the local system and on other systems, in the past and in the future, with extremely high probability (2^{128} UUIDs are possible)

uuidparse A utility to parse unique identifiers

wall Displays the contents of a file or, by default, its standard input, on the terminals of all currently logged in users

wdctl Shows hardware watchdog status

whereis Reports the location of the binary, source, and man page files for the given command

wipefs Wipes a filesystem signature from a device

x86_64 A symbolic link to setarch

zramctl A program to set up and control zram (compressed ram disk) devices

`libblkid` Contains routines for device identification and token extraction

`libfdisk` Contains routines for manipulating partition tables

`libmount` Contains routines for block device mounting and unmounting

`libsmartcols` Contains routines for aiding screen output in tabular form

`libuuid` Contains routines for generating unique identifiers for objects that may be accessible beyond the local system

8.79. E2fsprogs-1.47.0

The E2fsprogs package contains the utilities for handling the `ext2` file system. It also supports the `ext3` and `ext4` journaling file systems.

Approximate build time: 2.4 SBU on a spinning disk, 0.4 SBU on an SSD
Required disk space: 95 MB

8.79.1. 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           \
             --sysconfdir=/etc       \
             --enable-elf-shlibs     \
             --disable-libblkid      \
             --disable-libuuid       \
             --disable-uuidd         \
             --disable-fsck
```

The meaning of the configure options:

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

`--disable-*`
These prevent building and installing the `libuuid` and `libblkid` libraries, the `uuidd` daemon, and the **fsck** wrapper; util-linux installs more recent versions.

Compile the package:

```
make
```

To run the tests, issue:

```
make check
```

One test named `m_assume_storage_prezeroed` is known to fail.

Install the package:

```
make install
```

Remove useless static libraries:

```
rm -fv /usr/lib/{libcom_err,libe2p,libext2fs,libss}.a
```

This package installs a gzipped `.info` file but doesn't update the system-wide `dir` file. Unzip this file and then update the system `dir` file using the following commands:

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir /usr/share/info/libext2fs.info
```

If desired, create and install some additional documentation by issuing the following commands:

```
makeinfo -o      doc/com_err.info ../lib/et/com_err.texinfo
install -v -m644 doc/com_err.info /usr/share/info
install-info --dir-file=/usr/share/info/dir /usr/share/info/com_err.info
```

8.79.2. Configuring E2fsprogs

`/etc/mke2fs.conf` contains the default value of various command line options of **mke2fs**. You may edit the file to make the default values suitable for your need. For example, some utilities (not in LFS or BLFS) cannot recognize a `ext4` file system with `metadata_csum_seed` feature enabled. **If** you need such an utility, you may remove the feature from the default `ext4` feature list with the command:

```
sed 's/metadata_csum_seed,//' -i /etc/mke2fs.conf
```

Read the man page *mke2fs.conf(5)* for details.

8.79.3. Contents of E2fsprogs

Installed programs:	badblocks, chattr, compile_et, debugfs, dumpe2fs, e2freefrag, e2fsck, e2image, e2label, e2mmpstatus, e2scrub, e2scrub_all, e2undo, e4crypt, e4defrag, filefrag, fsck.ext2, fsck.ext3, fsck.ext4, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mkfs.ext4, mklost+found, resize2fs, and tune2fs
Installed libraries:	libcom_err.so, libe2p.so, libext2fs.so, and libss.so
Installed directories:	/usr/include/e2p, /usr/include/et, /usr/include/ext2fs, /usr/include/ss, /usr/lib/e2fsprogs, /usr/share/et, and /usr/share/ss

Short Descriptions

badblocks	Searches a device (usually a disk partition) for bad blocks
chattr	Changes the attributes of files on `ext{234}` 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 `ext{234}` file systems
dumpe2fs	Prints the super block and blocks group information for the file system present on a given device
e2freefrag	Reports free space fragmentation information
e2fsck	Is used to check and optionally repair `ext{234}` file systems
e2image	Is used to save critical `ext{234}` file system data to a file
e2label	Displays or changes the file system label on the `ext{234}` file system on a given device
e2mmpstatus	Checks MMP (Multiple Mount Protection) status of an `ext4` file system
e2scrub	Checks the contents of a mounted `ext{234}` file system
e2scrub_all	Checks all mounted `ext{234}` file systems for errors
e2undo	Replays the undo log for an `ext{234}` file system found on a device. [This can be used to undo a failed operation by an E2fsprogs program.]
e4crypt	`Ext4` file system encryption utility
e4defrag	Online defragmenter for `ext4` file systems
filefrag	Reports on how badly fragmented a particular file might be
fsck.ext2	By default checks `ext2` file systems and is a hard link to **e2fsck**
fsck.ext3	By default checks `ext3` file systems and is a hard link to **e2fsck**
fsck.ext4	By default checks `ext4` file systems and is a hard link to **e2fsck**

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 `ext{234}` file system on the given device
mkfs.ext2	By default creates `ext2` file systems and is a hard link to **mke2fs**
mkfs.ext3	By default creates `ext3` file systems and is a hard link to **mke2fs**
mkfs.ext4	By default creates `ext4` file systems and is a hard link to **mke2fs**
mklost+found	Creates a `lost+found` directory on an `ext{234}` file system; it pre-allocates disk blocks to this directory to lighten the task of **e2fsck**
resize2fs	Can be used to enlarge or shrink `ext{234}` file systems
tune2fs	Adjusts tunable file system parameters on `ext{234}` file systems
`libcom_err`	The common error display routine
`libe2p`	Used by **dumpe2fs**, **chattr**, and **lsattr**
`libext2fs`	Contains routines to enable user-level programs to manipulate `ext{234}` file systems
`libss`	Used by **debugfs**

8.80. Sysklogd-1.5.1

The Sysklogd package contains programs for logging system messages, such as those emitted by the kernel when unusual things happen.

Approximate build time: less than 0.1 SBU
Required disk space: 680 KB

8.80.1. Installation of Sysklogd

First, fix a problem that causes a segmentation fault in klogd under some conditions, and fix an obsolete program construct:

```
sed -i '/Error loading kernel symbols/{n;n;d}' ksym_mod.c
sed -i 's/union wait/int/' syslogd.c
```

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make BINDIR=/sbin install
```

8.80.2. 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
*.emerg *

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

8.80.3. 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.]

8.81. Sysvinit-3.08

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

Approximate build time: less than 0.1 SBU
Required disk space: 3.5 MB

8.81.1. Installation of Sysvinit

First, apply a patch that removes several programs installed by other packages, clarifies a message, and fixes a compiler warning:

```
patch -Np1 -i ../sysvinit-3.08-consolidated-1.patch
```

Compile the package:

```
make
```

This package does not come with a test suite.

Install the package:

```
make install
```

8.81.2. Contents of Sysvinit

Installed programs: bootlogd, fstab-decode, halt, init, killall5, poweroff (link to halt), reboot (link to halt), runlevel, shutdown, and telinit (link to init)

Short Descriptions

bootlogd Logs boot messages to a log file

fstab-decode Runs a command with fstab-encoded arguments

halt Normally invokes **shutdown** with the *-h* option, but when already in run-level 0, it tells the kernel to halt the system; it notes in the file `/var/log/wtmp` that the system is going down

init The first process to be started when the kernel has initialized the hardware; it takes over the boot process and starts all the processes specified in its configuration file

killall5 Sends a signal to all processes, except the processes in its own session; it will not kill its parent shell

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 `/run/utmp`

shutdown Brings the system down in a secure way, signaling all processes and notifying all logged-in users

telinit Tells **init** which run-level to change to

8.82. 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, the debugger can provide not only memory addresses, but also the names of the routines and variables.

The inclusion of these debugging symbols enlarges a program or library significantly. Here are two examples of the amount of space these symbols occupy:

- A **bash** binary with debugging symbols: 1200 KB
- A **bash** binary without debugging symbols: 480 KB (60% smaller)
- Glibc and GCC files (`/lib` and `/usr/lib`) with debugging symbols: 87 MB
- Glibc and GCC files without debugging symbols: 16 MB (82% smaller)

Sizes will vary depending on which compiler and C library were used, but a program that has been stripped of debugging symbols is usually some 50% to 80% smaller than its unstripped counterpart. 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.

8.83. Stripping

This section is optional. If the intended user is not a programmer and does not plan to do any debugging of the system software, the system's size can be decreased by some 2 GB by removing the debugging symbols, and some unnecessary symbol table entries, from binaries and libraries. This causes no real inconvenience for a typical Linux user.

Most people who use the commands mentioned below do not experience any difficulties. However, it is easy to make a mistake and render the new system unusable. So before running the **strip** commands, it is a good idea to make a backup of the LFS system in its current state.

A **strip** command with the *--strip-unneeded* option removes all debug symbols from a binary or library. It also removes all symbol table entries not needed by the linker (for static libraries) or dynamic linker (for dynamically linked binaries and shared libraries).

The debugging symbols from selected libraries are compressed with Zlib and preserved in separate files. That debugging information is needed to run regression tests with *valgrind* or *gdb* later, in BLFS.

Note that **strip** will overwrite the binary or library file it is processing. This can crash the processes using code or data from the file. If the process running **strip** is affected, the binary or library being stripped can be destroyed; this can make the system completely unusable. To avoid this problem we copy some libraries and binaries into `/tmp`, strip them there, then reinstall them with the **install** command. (The related entry in Section 8.2.1, “Upgrade Issues” gives the rationale for using the **install** command here.)

Note

The ELF loader's name is ld-linux-x86-64.so.2 on 64-bit systems and ld-linux.so.2 on 32-bit systems. The construct below selects the correct name for the current architecture, excluding anything ending with `g`, in case the commands below have already been run.

Important

If there is any package whose version is different from the version specified by the book (either following a security advisory or satisfying personal preference), it may be necessary to update the library file name in `save_usrlib` or `online_usrlib`. **Failing to do so may render the system completely unusable.**

```
save_usrlib="$(cd /usr/lib; ls ld-linux*[^g])
             libc.so.6
             libthread_db.so.1
             libquadmath.so.0.0.0
             libstdc++.so.6.0.32
             libitm.so.1.0.0
             libatomic.so.1.2.0"

cd /usr/lib

for LIB in $save_usrlib; do
    objcopy --only-keep-debug --compress-debug-sections=zlib $LIB $LIB.dbg
    cp $LIB /tmp/$LIB
    strip --strip-unneeded /tmp/$LIB
    objcopy --add-gnu-debuglink=$LIB.dbg /tmp/$LIB
    install -vm755 /tmp/$LIB /usr/lib
    rm /tmp/$LIB
done

online_usrbin="bash find strip"
online_usrlib="libbfd-2.42.so
               libsframe.so.1.0.0
               libhistory.so.8.2
               libncursesw.so.6.4-20230520
               libm.so.6
               libreadline.so.8.2
               libz.so.1.3.1
               libzstd.so.1.5.5
               $(cd /usr/lib; find libnss*.so* -type f)"

for BIN in $online_usrbin; do
    cp /usr/bin/$BIN /tmp/$BIN
    strip --strip-unneeded /tmp/$BIN
    install -vm755 /tmp/$BIN /usr/bin
    rm /tmp/$BIN
done

for LIB in $online_usrlib; do
    cp /usr/lib/$LIB /tmp/$LIB
    strip --strip-unneeded /tmp/$LIB
    install -vm755 /tmp/$LIB /usr/lib
    rm /tmp/$LIB
done

for i in $(find /usr/lib -type f -name \*.so* ! -name \*dbg) \
         $(find /usr/lib -type f -name \*.a)                 \
         $(find /usr/{bin,sbin,libexec} -type f); do
    case "$online_usrbin $online_usrlib $save_usrlib" in
        *$(basename $i)* )
            ;;
        * ) strip --strip-unneeded $i
            ;;
    esac
done

unset BIN LIB save_usrlib online_usrbin online_usrlib
```

A large number of files will be flagged as errors because their file format is not recognized. These warnings can be safely ignored. They indicate that those files are scripts, not binaries.

8.84. Cleaning Up

Finally, clean up some extra files left over from running tests:

```
rm -rf /tmp/*
```

There are also several files in the /usr/lib and /usr/libexec directories with a file name extension of .la. These are "libtool archive" files. On a modern Linux system the libtool .la files are only useful for libltdl. No libraries in LFS are expected to be loaded by libltdl, and it's known that some .la files can break BLFS package builds. Remove those files now:

```
find /usr/lib /usr/libexec -name \*.la -delete
```

For more information about libtool archive files, see the *BLFS section "About Libtool Archive (.la) files"*.

The compiler built in Chapter 6 and Chapter 7 is still partially installed and not needed anymore. Remove it with:

```
find /usr -depth -name $(uname -m)-lfs-linux-gnu\* | xargs rm -rf
```

Finally, remove the temporary 'tester' user account created at the beginning of the previous chapter.

```
userdel -r tester
```

Chapter 9. System Configuration

9.1. Introduction

Booting a Linux system involves several tasks. The process must mount both virtual and real file systems, initialize devices, check file systems for integrity, mount and activate any swap partitions or files, set the system clock, bring up networking, start any daemons required by the system, and accomplish any other custom tasks specified by the user. This process must be organized to ensure the tasks are performed in the correct order and executed as quickly as possible.

9.1.1. System V

System V is the classic boot process that has been used in Unix and Unix-like systems such as Linux since about 1983. It consists of a small program, **init**, that sets up basic processes such as **login** (via getty) and runs a script. This script, usually named **rc**, controls the execution of a set of additional scripts that perform the tasks required to initialize the system.

The **init** program is controlled by the `/etc/inittab` file and is organized into run levels that can be chosen by the user. In LFS, they are used as follows:

```
0 — halt
1 — Single user mode
2 — User definable
3 — Full multiuser mode
4 — User definable
5 — Full multiuser mode with display manager
6 — reboot
```

The usual default run level is 3 or 5.

Advantages

- Established, well understood system.
- Easy to customize.

Disadvantages

- May be slower to boot. A medium speed base LFS system takes 8-12 seconds where the boot time is measured from the first kernel message to the login prompt. Network connectivity is typically established about 2 seconds after the login prompt.
- Serial processing of boot tasks. This is related to the previous point. A delay in any process, such as a file system check, will delay the entire boot process.
- Does not directly support advanced features like control groups (cgroups) and per-user fair share scheduling.
- Adding scripts requires manual, static sequencing decisions.

9.2. LFS-Bootscripts-20230728

The LFS-Bootscripts package contains a set of scripts to start/stop the LFS system at bootup/shutdown. The configuration files and procedures needed to customize the boot process are described in the following sections.

Approximate build time: less than 0.1 SBU
Required disk space: 244 KB

9.2.1. Installation of LFS-Bootscripts

Install the package:

```
make install
```

9.2.2. Contents of LFS-Bootscripts

Installed scripts: checkfs, cleanfs, console, functions, halt, ifdown, ifup, localnet, modules, mountfs, mountvirtfs, network, rc, reboot, sendsignals, setclock, ipv4-static, swap, sysctl, sysklogd, template, udev, and udev_retry
Installed directories: /etc/rc.d, /etc/init.d (symbolic link), /etc/sysconfig, /lib/services, /lib/lsb (symbolic link)

Short Descriptions

checkfs Checks the integrity of the file systems before they are mounted (with the exception of journaling and network-based file systems)

cleanfs Removes files that should not be preserved between reboots, such as those in `/run/` and `/var/lock/`; it re-creates `/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 Stops a network device

ifup Initializes a network device

localnet Sets up the system's hostname and local loopback device

modules Loads kernel modules listed in `/etc/sysconfig/modules`, using arguments that are also given there

mountfs Mounts all file systems, except those that are marked *noauto*, or are network based

mountvirtfs 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 names of the symbolic links to those other bootscripts

reboot Reboots the system

sendsignals Makes sure every process is terminated before the system reboots or halts

setclock Resets the system clock to local time if the hardware clock is not set to UTC

ipv4-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
sysctl	Loads system configuration values from `/etc/sysctl.conf`, if that file exists, into the running kernel
sysklogd	Starts and stops the system and kernel log daemons
template	A template to create custom bootscripts for other daemons
udev	Prepares the `/dev` directory and starts the udev daemon
udev_retry	Retries failed udev uevents, and copies generated rules files from `/run/udev` to `/etc/udev/rules.d` if required

9.3. Overview of Device and Module Handling

In Chapter 8, we installed the udev daemon when udev was built. Before we go into the details regarding how udev works, a brief history of previous methods of handling devices is in order.

Linux systems in general traditionally used a static device creation method, whereby a great many device nodes were created under `/dev` (sometimes literally thousands of nodes), regardless of whether the corresponding hardware devices actually existed. This was typically done via a **MAKEDEV** script, which contained 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, device nodes are only created for those devices which are detected by the kernel. These device nodes are created each time the system boots; they are stored in a `devtmpfs` 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.

9.3.1. 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 configurable, the device naming policy should be chosen by system administrators, and not imposed on them by the developer(s). The `devfs` file system also suffered from race conditions that were inherent in its design; these could not be fixed without a substantial revision of the kernel. `devfs` was marked as deprecated for a long time, and was finally removed from the kernel in June, 2006.

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 provide information about the system's hardware configuration to userspace processes. With this userspace-visible representation, it became possible to develop a userspace replacement for `devfs`.

9.3.2. Udev Implementation

9.3.2.1. 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 register their objects in `sysfs` (devtmpfs internally) as they are detected by the kernel. For drivers compiled as modules, registration happens when the module is loaded. Once the `sysfs` filesystem is mounted (on `/sys`), data which the drivers have registered with `sysfs` are available to userspace processes and to udevd for processing (including modifications to device nodes).

9.3.2.2. Device Node Creation

Device files are created by the kernel in the `devtmpfs` file system. Any driver that wishes to register a device node will use the `devtmpfs` (via the driver core) to do it. When a `devtmpfs` instance is mounted on `/dev`, the device node will initially be exposed to userspace with a fixed name, permissions, and owner.

A short time later, the kernel will send a uevent to **udevd**. Based on the rules specified in the files within the `/etc/udev/rules.d`, `/usr/lib/udev/rules.d`, and `/run/udev/rules.d` directories, **udevd** will create additional symlinks to the device node, or change its permissions, owner, or group, or modify the internal **udevd** database entry (name) for that object.

The rules in these three directories are numbered and all three directories are merged together. If **udevd** can't find a rule for the device it is creating, it will leave the permissions and ownership at whatever `devtmpfs` used initially.

9.3.2.3. 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 default rules provided with udev will cause **udevd** to call out to **/sbin/modprobe** with the contents of the `MODALIAS` uevent environment variable (which 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.

9.3.2.4. 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.

9.3.3. Problems with Loading Modules and Creating Devices

There are a few possible problems when it comes to automatically creating device nodes.

9.3.3.1. 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-6.7.4, 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-6.7.4, 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.

9.3.3.2. 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 a “softdep” line to the corresponding `/etc/modprobe.d/<filename>.conf` file. For example:

```
softdep snd-pcm post: snd-pcm-oss
```

Note that the “softdep” command also allows `pre:` dependencies, or a mixture of both `pre:` and `post:` dependencies. See the *modprobe.d(5)* manual page for more information on “softdep” syntax and capabilities.

If the module in question is not a wrapper and is useful by itself, configure the **modules** 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.

9.3.3.3. Udev Loads Some Unwanted Module

Either don't build the module, or blacklist it in a `/etc/modprobe.d/blacklist.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.

9.3.3.4. Udev Creates a Device Incorrectly, or Makes the Wrong Symlink

This usually happens if a rule unexpectedly matches a device. For example, a poorly-written 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, with the help of the **udevadm info** command.

9.3.3.5. 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 (create this file if it does not exist). Please notify the LFS Development list if you do so and it helps.

9.3.3.6. Udev Does Not Create a Device

First, be certain that the driver is built into the kernel or already loaded as a module, and that udev isn't creating a misnamed device.

If a kernel driver does not export its data to `sysfs`, udev lacks the information needed to create a device node. This is most likely to happen with third party drivers from outside the kernel tree. Create a static device node in `/usr/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 **udev**.

9.3.3.7. 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 9.4, “Managing Devices” and Section 9.5, “General Network Configuration” for examples.

9.3.4. 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*

- The sysfs Filesystem *https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf*

9.4. Managing Devices

9.4.1. Network Devices

Udev, by default, names network devices according to Firmware/BIOS data or physical characteristics like the bus, slot, or MAC address. The purpose of this naming convention is to ensure that network devices are named consistently, not based on when the network card was discovered. In older versions of Linux—on a computer with two network cards made by Intel and Realtek, for instance—the network card manufactured by Intel might have become eth0 while the Realtek card became eth1. After a reboot, the cards would sometimes get renumbered the other way around.

In the new naming scheme, typical network device names are something like enp5s0 or wlp3s0. If this naming convention is not desired, the traditional naming scheme, or a custom scheme, can be implemented.

9.4.1.1. Disabling Persistent Naming on the Kernel Command Line

The traditional naming scheme using eth0, eth1, etc. can be restored by adding **net.ifnames=0** on the kernel command line. This is most appropriate for systems that have just one ethernet device of a particular type. Laptops often have two ethernet connections named eth0 and wlan0; such laptops can also use this method. The command line is in the GRUB configuration file. See Section 10.4.4, “Creating the GRUB Configuration File”.

9.4.1.2. Creating Custom Udev Rules

The naming scheme can be customized by creating custom udev rules. A script has been included that generates the initial rules. Generate these rules by running:

```
bash /usr/lib/udev/init-net-rules.sh
```

Now, inspect the /etc/udev/rules.d/70-persistent-net.rules file, to find out which name was assigned to which network device:

```
cat /etc/udev/rules.d/70-persistent-net.rules
```

Note

In some cases, such as when MAC addresses have been assigned to a network card manually, or in a virtual environment such as Qemu or Xen, the network rules file may not be generated because addresses are not consistently assigned. In these cases, this method cannot be used.

The file begins with a comment block, followed by two lines for each NIC. The first line for each NIC is a commented description showing its hardware IDs (e.g. its PCI vendor and device IDs, if it's a PCI card), along with its driver (in parentheses, if the driver can be found). Neither the hardware ID nor the driver is used to determine which name to give an interface; this information is only for reference. The second line is the udev rule that matches this NIC and actually assigns it a name.

All udev rules are made up of several keywords, separated by commas and optional whitespace. Here are the keywords, and an explanation of each one:

- SUBSYSTEM=="net" - This tells udev to ignore devices that are not network cards.
- ACTION=="add" - This tells udev to ignore this rule for a uevent that isn't an add ("remove" and "change" uevents also happen, but don't need to rename network interfaces).

- `DRIVERS=="?*"` - This exists so that udev will ignore VLAN or bridge sub-interfaces (because these sub-interfaces do not have drivers). These sub-interfaces are skipped because the name that would be assigned would collide with the parent devices.
- `ATTR{address}` - The value of this keyword is the NIC's MAC address.
- `ATTR{type}=="1"` - This ensures the rule only matches the primary interface in the case of certain wireless drivers which create multiple virtual interfaces. The secondary interfaces are skipped for the same reason that VLAN and bridge sub-interfaces are skipped: there would be a name collision otherwise.
- `NAME` - The value of this keyword is the name that udev will assign to this interface.

The value of `NAME` is the important part. Make sure you know which name has been assigned to each of your network cards before proceeding, and be sure to use that `NAME` value when creating your network configuration files.

Even if the custom udev rule file is created, udev may still assign one or more alternative names for a NIC based on physical characteristics. If a custom udev rule would rename some NIC using a name already assigned as an alternative name of another NIC, this udev rule will fail. If this issue happens, you may create the `/etc/udev/network/99-default.link` configuration file with an empty alternative assignment policy, overriding the default configuration file `/usr/lib/udev/network/99-default.link`:

```
sed -e '/^AlternativeNamesPolicy/s/=.*$/=/'  \
    -i /usr/lib/udev/network/99-default.link \
     > /etc/udev/network/99-default.link
```

9.4.2. CD-ROM Symlinks

Some software that you may want to install later (e.g., various media players) expects the `/dev/cdrom` and `/dev/dvd` symlinks to exist, and to point to a CD-ROM or DVD-ROM device. Also, it may be convenient to put references to those symlinks into `/etc/fstab`. Udev comes with a script that will generate rules files to create these symlinks for you, depending on the capabilities of each device, but you need to decide which of two modes of operation you wish to have the script use.

First, the script can operate in “by-path” mode (used by default for USB and FireWire devices), where the rules it creates depend on the physical path to the CD or DVD device. Second, it can operate in “by-id” mode (default for IDE and SCSI devices), where the rules it creates depend on identification strings stored on the CD or DVD device itself. The path is determined by udev's **path_id** script, and the identification strings are read from the hardware by its **ata_id** or **scsi_id** programs, depending on which type of device you have.

There are advantages to each approach; the correct approach depends on what kinds of device changes may happen. If you expect the physical path to the device (that is, the ports and/or slots that it plugs into) to change, for example because you plan on moving the drive to a different IDE port or a different USB connector, then you should use the “by-id” mode. On the other hand, if you expect the device's identification to change, for example because it may die, and you intend to replace it with a different device that plugs into the same connectors, then you should use the “by-path” mode.

If either type of change is possible with your drive, then choose a mode based on the type of change you expect to happen more often.

Important

External devices (for example, a USB-connected CD drive) should not use by-path persistence, because each time the device is plugged into a new external port, its physical path will change. All externally-connected devices will have this problem if you write udev rules to recognize them by their physical path; the problem is not limited to CD and DVD drives.

If you wish to see the values that the udev scripts will use, then for the appropriate CD-ROM device, find the corresponding directory under /sys (e.g., this can be /sys/block/hdd) and run a command similar to the following:

```
udevadm test /sys/block/hdd
```

Look at the lines containing the output of various *_id programs. The “by-id” mode will use the ID_SERIAL value if it exists and is not empty, otherwise it will use a combination of ID_MODEL and ID_REVISION. The “by-path” mode will use the ID_PATH value.

If the default mode is not suitable for your situation, then the following modification can be made to the /etc/udev/rules.d/83-cdrom-symlinks.rules file, as follows (where *mode* is one of “by-id” or “by-path”):

```
sed -e 's/"write_cd_rules"/"write_cd_rules mode"/' \
    -i /etc/udev/rules.d/83-cdrom-symlinks.rules
```

Note that it is not necessary to create the rules files or symlinks at this time because you have bind-mounted the host's /dev directory into the LFS system and we assume the symlinks exist on the host. The rules and symlinks will be created the first time you boot your LFS system.

However, if you have multiple CD-ROM devices, then the symlinks generated at that time may point to different devices than they point to on your host because devices are not discovered in a predictable order. The assignments created when you first boot the LFS system will be stable, so this is only an issue if you need the symlinks on both systems to point to the same device. If you need that, then inspect (and possibly edit) the generated /etc/udev/rules.d/70-persistent-cd.rules file after booting, to make sure the assigned symlinks match your needs.

9.4.3. Dealing with Duplicate Devices

As explained in Section 9.3, “Overview of Device and Module Handling”, 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. For all classes of hardware except sound cards and network cards, this is fixable by creating udev rules to create persistent symlinks. The case of network cards is covered separately in Section 9.5, “General Network Configuration”, 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/videoX. Figure out the attributes that identify the device uniquely (usually, vendor and product IDs and/or serial numbers work):

```
udevadm info -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*", ATTRS{idProduct}=="1910", ATTRS{idVendor}=="0d81", SYMLINK+="webcam"
KERNEL=="video*", ATTRS{device}=="0x036f",  ATTRS{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.

9.5. General Network Configuration

9.5.1. Creating Network Interface Configuration Files

The files in `/etc/sysconfig/` usually determine which interfaces are brought up and down by the network script. This directory should contain a file for each interface to be configured, such as `ifconfig.xyz`, where "xyz" describes the network card. The interface name (e.g. eth0) is usually appropriate. Each file contains the attributes of one interface, such as its IP address(es), subnet masks, and so forth. The stem of the filename must be *ifconfig*.

Note

If the procedure in the previous section was not used, udev will assign network card interface names based on system physical characteristics such as enp2s1. If you are not sure what your interface name is, you can always run **ip link** or **ls /sys/class/net** after you have booted your system.

The interface names depend on the implementation and configuration of the udev daemon running on the system. The udev daemon for LFS (installed in Section 8.75, "Udev from Systemd-255") will not run until the LFS system is booted. So the interface names in the LFS system cannot always be determined by running those commands on the host distro, *even in the chroot environment*.

The following command creates a sample file for the *eth0* device with a static IP address:

```
cd /etc/sysconfig/
cat > ifconfig.eth0 << "EOF"
ONBOOT=yes
IFACE=eth0
SERVICE=ipv4-static
IP=192.168.1.2
GATEWAY=192.168.1.1
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

The values in italics must be changed in each file, to set the interfaces up correctly.

If the `ONBOOT` variable is set to `yes` the System V network script will bring up the Network Interface Card (NIC) during the system boot process. If set to anything besides `yes`, the NIC will be ignored by the network script and will not be started automatically. Interfaces can be manually started or stopped with the **ifup** and **ifdown** commands.

The `IFACE` variable defines the interface name, for example, eth0. It is required for all network device configuration files. The filename extension must match this value.

The `SERVICE` variable defines the method used for obtaining the IP address. The LFS-Bootscripts package has a modular IP assignment format, and creating additional files in the `/lib/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 specifies the number of bits used in the subnet. Each segment of an IP address is 8 bits. If the subnet's netmask is 255.255.255.0, then it is using the first three segments (24 bits) to specify the network number. If the netmask is 255.255.255.240, the subnet is 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. If omitted, the PREFIX defaults to 24.

For more information see the **ifup** man page.

9.5.2. Creating the /etc/resolv.conf File

The system will need some means of obtaining 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
```

The `domain` statement can be omitted or replaced with a `search` statement. See the man page for resolv.conf for more details.

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.

Note

The Google Public IPv4 DNS addresses are 8.8.8.8 and 8.8.4.4.

9.5.3. Configuring the System Hostname

During the boot process, the file `/etc/hostname` is used for establishing the system's hostname.

Create the `/etc/hostname` file and enter a hostname by running:

```
echo "<lfs>" > /etc/hostname
```

<lfs> needs to be replaced with the name given to the computer. Do not enter the Fully Qualified Domain Name (FQDN) here. That information goes in the `/etc/hosts` file.

9.5.4. Customizing the /etc/hosts File

Decide on a fully-qualified domain name (FQDN), and possible aliases for use in the `/etc/hosts` file. If using static IP addresses, you'll also need to decide on an IP address. The syntax for a hosts file entry 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:

```
Private Network Address Range      Normal Prefix
10.0.0.1 - 10.255.255.254           8
172.x.0.1 - 172.x.255.254           16
192.168.y.1 - 192.168.y.254         24
```

x can be any number in the range 16-31. y can be any number in the range 0-255.

A valid private IP address could be 192.168.1.1.

If the computer is to be visible to the Internet, a valid FQDN can be the domain name itself, or a string resulted by concatenating a prefix (often the hostname) and the domain name with a "." character. And, you need to contact the domain provider to resolve the FQDN to your public IP address.

Even if the computer is not visible to the Internet, a FQDN is still needed for certain programs, such as MTAs, to operate properly. A special FQDN, `localhost.localdomain`, can be used for this purpose.

Create the `/etc/hosts` file by running:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts

127.0.0.1 localhost.localdomain localhost
127.0.1.1 <FQDN> <HOSTNAME>
<192.168.1.1> <FQDN> <HOSTNAME> [alias1] [alias2 ...]
::1       localhost ip6-localhost ip6-loopback
ff02::1   ip6-allnodes
ff02::2   ip6-allrouters

# End /etc/hosts
EOF
```

The *<192.168.1.1>*, *<FQDN>*, and *<HOSTNAME>* values need to be changed for specific uses or requirements (if assigned an IP address by a network/system administrator and the machine will be connected to an existing network). The optional alias name(s) can be omitted.

9.6. System V Bootscript Usage and Configuration

9.6.1. How Do the System V Bootscripts Work?

This version of LFS uses a special booting facility named SysVinit, based on a series of *run levels*. The boot procedure can be quite different from one system to another; the fact that things worked one way in a particular Linux distribution does not guarantee they will work the same way in LFS. LFS has its own way of doing things, but it does respect generally accepted standards.

There is an alternative boot procedure called **systemd**. We will not discuss that boot process any further here. For a detailed description visit *https://www.linux.com/training-tutorials/understanding-and-using-systemd/*.

SysVinit (which will be referred to as "init" from now on) uses a run levels scheme. There are seven run levels, numbered 0 to 6. (Actually, there are more run levels, but the others are for special cases and are generally not used. See *init(8)* for more details.) Each one of the seven corresponds to actions the computer is supposed to perform when it starts up or shuts down. The default run level is 3. Here are the descriptions of the different run levels as they are implemented in LFS:

```
0: halt the computer
1: single-user mode
2: reserved for customization, otherwise the same as 3
3: multi-user mode with networking
4: reserved for customization, otherwise the same as 3
5: same as 4, it is usually used for GUI login (like GNOME's gdm or LXDE's lxdm)
6: reboot the computer
```

Note

Classically, run level 2 above was defined as “multi-user mode without networking,” but this was only the case many years ago when multiple users could connect to a system via serial ports. In today's environment it makes no sense, and we now say it is “reserved.”

9.6.2. Configuring Sysvinit

During kernel initialization, the first program that is run (if not overridden on the command line) is **init**. This program reads the initialization file `/etc/inittab`. Create this file with:

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab

id:3:initdefault:

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

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:S06:once:/sbin/sulogin
s1:1:respawn:/sbin/sulogin

1:2345:respawn:/sbin/agetty --noclear tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# End /etc/inittab
EOF
```

An explanation of this initialization file is in the man page for *inittab*. In LFS, the key command is **rc**. The initialization file above instructs **rc** to run all the scripts starting with an S in the `/etc/rc.d/rcS.d` directory followed by all the scripts starting with an S in the `/etc/rc.d/rc?.d` directory where the question mark is specified by the initdefault value.

As a convenience, the **rc** script reads a library of functions in `/lib/lsb/init-functions`. This library also reads an optional configuration file, `/etc/sysconfig/rc.site`. Any of the system configuration parameters described in subsequent sections can be placed in this file, allowing consolidation of all system parameters in this one file.

As a debugging convenience, the functions script also logs all output to `/run/var/bootlog`. Since the `/run` directory is a tmpfs, this file is not persistent across boots; however, it is appended to the more permanent file `/var/log/boot.log` at the end of the boot process.

9.6.2.1. Changing Run Levels

Changing run levels is done with **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 `rcS.d`, all containing a number of symbolic links. Some links 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 smaller the number, the sooner the script runs. When **init** switches to another run level, the appropriate services are either started or stopped, depending on the run level chosen.

The real scripts are in `/etc/rc.d/init.d`. They do the actual work, and the symlinks all point to them. K links and S 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.

These are descriptions of what the arguments make the scripts do:

`start`
: The service is started.

`stop`
: The service is stopped.

`restart`
: The service is stopped and then started again.

`reload`
: The configuration of the service is updated. This is used after the configuration file of a service was modified, when the service does not need to be restarted.

`status`
: Tells if the service is running and with which PIDs.

Feel free to modify the way the boot process works (after all, it is your own LFS system). The files given here are an example of how it can be done.

9.6.3. Udev Bootscripts

The `/etc/rc.d/init.d/udev` initscript starts **udevd**, triggers any "coldplug" devices that have already been created by the kernel, and waits for any rules to complete. The script also unsets the uevent handler from the default of `/sbin/hotplug` . This is done because the kernel no longer needs to call an external binary. Instead, **udevd** will listen on a netlink socket for uevents that the kernel raises.

The **/etc/rc.d/init.d/udev_retry** script takes care of re-triggering events for subsystems whose rules may rely on file systems that are not mounted until the **mountfs** script is run (in particular, `/usr` and `/var` may cause this). This script runs after the **mountfs** script, so those rules (if re-triggered) should succeed the second time around. It is configured by the `/etc/sysconfig/udev_retry` file; any words in this file other than comments are considered subsystem names to trigger at retry time. To find the subsystem of a device, use **udevadm info --attribute-walk <device>** where <device> is an absolute path in /dev or /sys, such as /dev/sr0, or /sys/class/rtc.

For information on kernel module loading and udev, see Section 9.3.2.3, “Module Loading”.

9.6.4. Configuring the System Clock

The **setclock** script reads the time from the hardware clock, also known as the BIOS or 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 time zone to use). There is no way to detect whether or not the hardware clock is set to UTC, so this must be configured manually.

The **setclock** program is run via udev when the kernel detects the hardware capability upon boot. It can also be run manually with the stop parameter to store the system time to the CMOS clock.

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 number of hours for your time zone to the time shown by **hwclock**. For example, if you are currently in the MST time zone, 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

# Set this to any options you might need to give to hwclock,
# such as machine hardware clock type for Alphas.
CLOCKPARAMS=

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

A good hint explaining how to deal with time on LFS is available at *https://www.linuxfromscratch.org/hints/downloads/files/time.txt*. It explains issues such as time zones, UTC, and the TZ environment variable.

Note

The CLOCKPARAMS and UTC parameters may also be set in the /etc/sysconfig/rc.site file.

9.6.5. Configuring the Linux Console

This section discusses how to configure the **console** bootscript that sets up the keyboard map, console font, and console kernel log level. If non-ASCII characters (e.g., the copyright sign, the British pound sign, and the Euro symbol) will not be used and the keyboard is a U.S. one, much of this section can be skipped. Without the configuration file, (or equivalent settings in rc.site), 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 HOWTOs can also help with this; see *https://tldp.org/HOWTO/HOWTO-INDEX/other-lang.html*. If still in doubt, look in the /usr/share/keymaps and /usr/share/consolefonts directories for valid keymaps and screen fonts. Read the *loadkeys(1)* and *setfont(8)* manual pages to determine the correct arguments for these programs.

The /etc/sysconfig/console file should contain lines of the form: VARIABLE=value. The following variables are recognized:

LOGLEVEL
: This variable specifies the log level for kernel messages sent to the console as set by **dmesg -n**. Valid levels are from 1 (no messages) to 8. The default level is 7, which is quite verbose.

KEYMAP
: This variable specifies the arguments for the **loadkeys** program, typically, the name of the keymap to load, e.g., it. If this variable is not set, the bootscript will not run the **loadkeys** program, and the default kernel keymap will

be used. Note that a few keymaps have multiple versions with the same name (cz and its variants in qwerty/ and qwertz/, es in olpc/ and qwerty/, and trf in fgGIod/ and qwerty/). In these cases the parent directory should also be specified (e.g. qwerty/es) to ensure the proper keymap is loaded.

KEYMAP_CORRECTIONS

This (rarely used) variable specifies the arguments for the second call to the **loadkeys** program. This is useful if the stock keymap is not completely satisfactory and a small adjustment has to be made. E.g., to include the Euro sign into a keymap that normally doesn't have it, set this variable to `euro2`.

FONT

This variable specifies the arguments for the **setfont** program. Typically, this includes the font name, `-m`, and the name of the application character map to load. E.g., in order to load the “lat1-16” font together with the “8859-1” application character map (appropriate in the USA), set this variable to `lat1-16 -m 8859-1`. In UTF-8 mode, the kernel uses the application character map to convert 8-bit key codes to UTF-8. Therefore the argument of the "-m" parameter should be set to the encoding of the composed key codes in the keymap.

UNICODE

Set this variable to `1`, `yes`, or `true` in order to put the console into UTF-8 mode. This is useful in UTF-8 based locales and harmful otherwise.

LEGACY_CHARSET

For many keyboard layouts, there is no stock Unicode keymap in the Kbd package. The **console** bootscript will convert an available keymap to UTF-8 on the fly if this variable is set to the encoding of the available non-UTF-8 keymap.

Some examples:

- We'll use `C.UTF-8` as the locale for interactive sessions in the Linux console in Section 9.7, “Configuring the System Locale,” so we should set `UNICODE` to `1`. And the console fonts shipped by the Kbd package containing the glyphs for all characters from the program messages in the `C.UTF-8` locale are `LatArCyrHeb*.psfu.gz`, `LatGrkCyr*.psfu.gz`, `Lat2-Terminus16.psfu.gz`, and `pancyrillic.f16.psfu.gz` in `/usr/share/consolefonts` (the other shipped console fonts lack glyphs of some characters like the Unicode left/right quotation marks and the Unicode English dash). So set one of them, for example `Lat2-Terminus16.psfu.gz` as the default console font:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
FONT="Lat2-Terminus16"

# End /etc/sysconfig/console
EOF
```

- For a non-Unicode setup, only the KEYMAP and FONT variables are generally needed. E.g., for a Polish setup, one would use:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="pl2"
FONT="lat2a-16 -m 8859-2"

# End /etc/sysconfig/console
EOF
```

- As mentioned above, it is sometimes necessary to adjust a stock keymap slightly. The following example adds the Euro symbol to the German keymap:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
FONT="lat0-16 -m 8859-15"
UNICODE="1"

# End /etc/sysconfig/console
EOF
```

- The following is a Unicode-enabled example for Bulgarian, where a stock UTF-8 keymap exists:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="LatArCyrHeb-16"

# End /etc/sysconfig/console
EOF
```

- Due to the use of a 512-glyph LatArCyrHeb-16 font in the previous example, bright colors are no longer available on the Linux console unless a framebuffer is used. If one wants to have bright colors without a framebuffer and can live without characters not belonging to his language, it is still possible to use a language-specific 256-glyph font, as illustrated below:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="cyr-sun16"

# End /etc/sysconfig/console
EOF
```

- The following example illustrates keymap autoconversion from ISO-8859-15 to UTF-8 and enabling dead keys in Unicode mode:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
LEGACY_CHARSET="iso-8859-15"
FONT="LatArCyrHeb-16 -m 8859-15"

# End /etc/sysconfig/console
EOF
```

- Some keymaps have dead keys (i.e., keys that don't produce a character by themselves, but put an accent on the character produced by the next key) or define composition rules (such as: “press Ctrl+. A E to get Æ” in the default keymap). Linux-6.7.4 interprets dead keys and composition rules in the keymap correctly only when the source characters to be composed together are not multibyte. This deficiency doesn't affect keymaps for European languages, because there accents are added to unaccented ASCII characters, or two ASCII characters are

composed together. However, in UTF-8 mode it is a problem; e.g., for the Greek language, where one sometimes needs to put an accent on the letter α. The solution is either to avoid the use of UTF-8, or to install the X window system, which doesn't have this limitation, in its input handling.

- For Chinese, Japanese, Korean, and some other languages, the Linux console cannot be configured to display the needed characters. Users who need such languages should install the X Window System, fonts that cover the necessary character ranges, and the proper input method (e.g., SCIM supports a wide variety of languages).

Note

The `/etc/sysconfig/console` file only controls the Linux text console localization. It has nothing to do with setting the proper keyboard layout and terminal fonts in the X Window System, with ssh sessions, or with a serial console. In such situations, limitations mentioned in the last two list items above do not apply.

9.6.6. Creating Files at Boot

At times, it is desirable to create files at boot time. For instance, the `/tmp/.ICE-unix` directory is often needed. This can be done by creating an entry in the `/etc/sysconfig/createfiles` configuration script. The format of this file is embedded in the comments of the default configuration file.

9.6.7. Configuring the Sysklogd Script

The `sysklogd` script invokes the **syslogd** program as a part of System V initialization. The *`-m 0`* 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 `/etc/sysconfig/rc.site` and define the variable SYSKLOGD_PARMS to the desired value. For instance, to remove all parameters, set the variable to a null value:

```
SYSKLOGD_PARMS=
```

See **`man syslogd`** for more options.

9.6.8. The rc.site File

The optional `/etc/sysconfig/rc.site` file contains settings that are automatically set for each SystemV boot script. It can alternatively set the values specified in the `hostname`, `console`, and `clock` files in the `/etc/sysconfig/` directory. If the associated variables are present in both these separate files and `rc.site`, the values in the script-specific files take effect.

`rc.site` also contains parameters that can customize other aspects of the boot process. Setting the IPROMPT variable will enable selective running of bootscripts. Other options are described in the file comments. The default version of the file is as follows:

```
# rc.site
# Optional parameters for boot scripts.

# Distro Information
# These values, if specified here, override the defaults
#DISTRO="Linux From Scratch" # The distro name
#DISTRO_CONTACT="lfs-dev@lists.linuxfromscratch.org" # Bug report address
#DISTRO_MINI="LFS" # Short name used in filenames for distro config

# Define custom colors used in messages printed to the screen

# Please consult `man console_codes` for more information
# under the "ECMA-48 Set Graphics Rendition" section
#
```

```
# Warning: when switching from a 8bit to a 9bit font,
# the linux console will reinterpret the bold (1;) to
# the top 256 glyphs of the 9bit font.  This does
# not affect framebuffer consoles

# These values, if specified here, override the defaults
#BRACKET="\\033[1;34m" # Blue
#FAILURE="\\033[1;31m" # Red
#INFO="\\033[1;36m"    # Cyan
#NORMAL="\\033[0;39m"  # Grey
#SUCCESS="\\033[1;32m" # Green
#WARNING="\\033[1;33m" # Yellow

# Use a colored prefix
# These values, if specified here, override the defaults
#BMPREFIX="      "
#SUCCESS_PREFIX="${SUCCESS}  *  ${NORMAL} "
#FAILURE_PREFIX="${FAILURE}*****${NORMAL} "
#WARNING_PREFIX="${WARNING} *** ${NORMAL} "

# Manually set the right edge of message output (characters)
# Useful when resetting console font during boot to override
# automatic screen width detection
#COLUMNS=120

# Interactive startup
#IPROMPT="yes" # Whether to display the interactive boot prompt
#itime="3"    # The amount of time (in seconds) to display the prompt

# The total length of the distro welcome string, without escape codes
#wlen=$(echo "Welcome to ${DISTRO}" | wc -c )
#welcome_message="Welcome to ${INFO}${DISTRO}${NORMAL}"

# The total length of the interactive string, without escape codes
#ilen=$(echo "Press 'I' to enter interactive startup" | wc -c )
#i_message="Press '${FAILURE}I${NORMAL}' to enter interactive startup"

# Set scripts to skip the file system check on reboot
#FASTBOOT=yes

# Skip reading from the console
#HEADLESS=yes

# Write out fsck progress if yes
#VERBOSE_FSCK=no

# Speed up boot without waiting for settle in udev
#OMIT_UDEV_SETTLE=y

# Speed up boot without waiting for settle in udev_retry
#OMIT_UDEV_RETRY_SETTLE=yes

# Skip cleaning /tmp if yes
#SKIPTMPCLEAN=no

# For setclock
#UTC=1
#CLOCKPARAMS=

# For consolelog (Note that the default, 7=debug, is noisy)
#LOGLEVEL=7

# For network
```

```
#HOSTNAME=mylfs

# Delay between TERM and KILL signals at shutdown
#KILLDELAY=3

# Optional sysklogd parameters
#SYSKLOGD_PARMS="-m 0"

# Console parameters
#UNICODE=1
#KEYMAP="de-latin1"
#KEYMAP_CORRECTIONS="euro2"
#FONT="lat0-16 -m 8859-15"
#LEGACY_CHARSET=
```

9.6.8.1. Customizing the Boot and Shutdown Scripts

The LFS boot scripts boot and shut down a system in a fairly efficient manner, but there are a few tweaks you can make in the rc.site file to improve speed even more, and to adjust messages according to your preferences. To do this, adjust the settings in the `/etc/sysconfig/rc.site` file above.

- During the boot script `udev`, there is a call to **udev settle** that requires some time to complete. This time may or may not be required depending on the devices in the system. If you only have simple partitions and a single ethernet card, the boot process will probably not need to wait for this command. To skip it, set the variable OMIT_UDEV_SETTLE=y.
- The boot script `udev_retry` also runs **udev settle** by default. This command is only needed if the `/var` directory is separately mounted, because the clock needs the `/var/lib/hwclock/adjtime` file. Other customizations may also need to wait for udev to complete, but in many installations it is not necessary. Skip the command by setting the variable OMIT_UDEV_RETRY_SETTLE=y.
- By default, the file system checks are silent. This can appear to be a delay during the bootup process. To turn on the **fsck** output, set the variable VERBOSE_FSCK=y.
- When rebooting, you may want to skip the filesystem check, **fsck**, completely. To do this, either create the file `/fastboot` or reboot the system with the command **/sbin/shutdown -f -r now**. On the other hand, you can force all file systems to be checked by creating `/forcefsck` or running **shutdown** with the `-F` parameter instead of `-f`.

  Setting the variable FASTBOOT=y will disable **fsck** during the boot process until it is removed. This is not recommended on a permanent basis.
- Normally, all files in the `/tmp` directory are deleted at boot time. Depending on the number of files or directories present, this can cause a noticeable delay in the boot process. To skip removing these files set the variable SKIPTMPCLEAN=y.
- During shutdown, the **init** program sends a TERM signal to each program it has started (e.g. agetty), waits for a set time (default 3 seconds), then sends each process a KILL signal and waits again. This process is repeated in the **sendsignals** script for any processes that are not shut down by their own scripts. The delay for **init** can be set by passing a parameter. For example to remove the delay in **init**, pass the -t0 parameter when shutting down or rebooting (e.g. **/sbin/shutdown -t0 -r now**). The delay for the **sendsignals** script can be skipped by setting the parameter KILLDELAY=0.

9.7. Configuring the System Locale

Some environment variables are necessary for native language support. Setting them properly results in:

- The output of programs being translated into your native language
- The 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
- The appropriate default paper size
- The correct formatting of monetary, time, and date values

Replace *<ll>* below with the two-letter code for your 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. Optional modifiers such as @euro may also be present.

The list of all locales supported by Glibc can be obtained by running the following command:

```
locale -a
```

Charmaps can have a number of aliases, e.g., ISO-8859-1 is also referred to as iso8859-1 and iso88591. Some applications cannot handle the various synonyms correctly (e.g., require that UTF-8 is written as UTF-8, not utf8), so it is the safest in most cases 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 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 8 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.

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.

The shell program **/bin/bash** (here after referred as “the shell”) uses a collection of startup files to help create the 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 equivalent files exist in the home directory, they 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.

Create the `/etc/profile` once the proper locale settings have been determined to set the desired locale, but set the `C.UTF-8` locale instead if running in the Linux console (to prevent programs from outputting characters that the Linux console is unable to render):

```
cat > /etc/profile << "EOF"
# Begin /etc/profile

for i in $(locale); do
  unset ${i%=*}
done

if [[ "$TERM" = linux ]]; then
  export LANG=C.UTF-8
else
  export LANG=<ll>_<CC>.<charmap><@modifiers>
fi

# End /etc/profile
EOF
```

The `C` (default) and `en_US` (the recommended one for United States English users) locales are different. `C` uses the US-ASCII 7-bit character set, and treats bytes with the high bit set as invalid characters. That's why, e.g., the **ls** command substitutes them with question marks in that locale. Also, an attempt to send mail with such characters from Mutt or Pine results in non-RFC-conforming messages being sent (the charset in the outgoing mail is indicated as `unknown 8-bit`). It's suggested that you use the `C` locale only if you are certain that you will never need 8-bit characters.

9.8. Creating the /etc/inputrc File

The `inputrc` file is the configuration file for the readline library, which provides editing capabilities while the user is entering a line from the terminal. It works by translating keyboard inputs into specific actions. Readline is used by bash and most other shells as well as many other applications.

Most people do not need user-specific functionality 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 8-bit 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 in 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
```

9.9. Creating the /etc/shells File

The `shells` file contains a list of login shells on the system. Applications use this file to determine whether a shell is valid. For each shell a single line should be present, consisting of the shell's path relative to the root of the directory structure (/).

For example, this file is consulted by **chsh** to determine whether an unprivileged user may change the login shell for her own account. If the command name is not listed, the user will be denied the ability to change shells.

It is a requirement for applications such as GDM which does not populate the face browser if it can't find `/etc/shells`, or FTP daemons which traditionally disallow access to users with shells not included in this file.

```
cat > /etc/shells << "EOF"
# Begin /etc/shells

/bin/sh
/bin/bash

# End /etc/shells
EOF
```

Chapter 10. Making the LFS System Bootable

10.1. Introduction

It is time to make the LFS system bootable. This chapter discusses creating the `/etc/fstab` file, building a kernel for the new LFS system, and installing the GRUB boot loader so that the LFS system can be selected for booting at startup.

10.2. 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     nosuid,noexec,nodev 0     0
sysfs          /sys           sysfs    nosuid,noexec,nodev 0     0
devpts         /dev/pts       devpts   gid=5,mode=620      0     0
tmpfs          /run           tmpfs    defaults            0     0
devtmpfs       /dev           devtmpfs mode=0755,nosuid    0     0
tmpfs          /dev/shm       tmpfs    nosuid,nodev        0     0
cgroup2        /sys/fs/cgroup cgroup2  nosuid,noexec,nodev 0     0

# End /etc/fstab
EOF
```

Replace *<xxx>*, *<yyy>*, and *<fff>* with the values appropriate for the system, for example, `sda2`, `sda5`, and `ext4`. For details on the six fields in this file, see *fstab(5)*.

Filesystems with MS-DOS or Windows origin (i.e. vfat, ntfs, smbfs, cifs, iso9660, udf) need a special option, utf8, in order for non-ASCII characters in file names to be interpreted properly. For non-UTF-8 locales, the value of `iocharset` should be set to be the same as the character set of the locale, adjusted in such a way that the kernel understands it. This works if the relevant character set definition (found under File systems -> Native Language Support when configuring the kernel) has been compiled into the kernel or built as a module. However, if the character set of the locale is UTF-8, the corresponding option `iocharset=utf8` would make the file system case sensitive. To fix this, use the special option `utf8` instead of `iocharset=utf8`, for UTF-8 locales. The “codepage” option is also needed for vfat and smbfs filesystems. It should be set to the codepage number used under MS-DOS in your country. For example, in order to mount USB flash drives, a ru_RU.KOI8-R user would need the following in the options portion of its mount line in `/etc/fstab`:

```
noauto,user,quiet,showexec,codepage=866,iocharset=koi8r
```

The corresponding options fragment for ru_RU.UTF-8 users is:

```
noauto,user,quiet,showexec,codepage=866,utf8
```

Note that using `iocharset` is the default for `iso8859-1` (which keeps the file system case insensitive), and the `utf8` option tells the kernel to convert the file names using UTF-8 so they can be interpreted in the UTF-8 locale.

It is also possible to specify default codepage and iocharset values for some filesystems during kernel configuration. The relevant parameters are named “Default NLS Option” (`CONFIG_NLS_DEFAULT`), “Default Remote NLS Option” (`CONFIG_SMB_NLS_DEFAULT`), “Default codepage for FAT” (`CONFIG_FAT_DEFAULT_CODEPAGE`), and “Default iocharset for FAT” (`CONFIG_FAT_DEFAULT_IOCHARSET`). There is no way to specify these settings for the ntfs filesystem at kernel compilation time.

It is possible to make the ext3 filesystem reliable across power failures for some hard disk types. To do this, add the `barrier=1` mount option to the appropriate entry in `/etc/fstab`. To check if the disk drive supports this option, run *hdparm* on the applicable disk drive. For example, if:

```
hdparm -I /dev/sda | grep NCQ
```

returns non-empty output, the option is supported.

Note: Logical Volume Management (LVM) based partitions cannot use the `barrier` option.

10.3. Linux-6.7.4

The Linux package contains the Linux kernel.

Approximate build time: 0.6 - 20.4 SBU (typically about 1.4 SBU)
Required disk space: 1.8 - 10.6 GB (typically about 2 GB)

10.3.1. 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.

Important

Building the linux kernel for the first time is one of the most challenging tasks in LFS. Getting it right depends on the specific hardware for the target system and your specific needs. There are almost 12,000 configuration items that are available for the kernel although only about a third of them are needed for most computers. The LFS editors recommend that users not familiar with this process follow the procedures below fairly closely. The objective is to get an initial system to a point where you can log in at the command line when you reboot later in Section 11.3, “Rebooting the System”. At this point optimization and customization is not a goal.

For general information on kernel configuration see *https://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt*. Additional information about configuring and building the kernel can be found at *https://anduin.linuxfromscratch.org/LFS/kernel-nutshell/*. These references are a bit dated, but still give a reasonable overview of the process.

If all else fails, you can ask for help on the *lfs-support* mailing list. Note that subscribing is required in order for the list to avoid spam.

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.

There are several ways to configure the kernel options. Usually, this is done through a menu-driven interface, for example:

```
make menuconfig
```

The meaning of optional make environment variables:

LANG=<host_LANG_value> LC_ALL=
This establishes the locale setting to the one used on the host. This may be needed for a proper menuconfig ncurses interface line drawing on a UTF-8 linux text console.
If used, be sure to replace *<host_LANG_value>* by the value of the $LANG variable from your host. You can alternatively use instead the host's value of $LC_ALL or $LC_CTYPE.

make menuconfig
This launches an ncurses menu-driven interface. For other (graphical) interfaces, type **make help**.

Note

A good starting place for setting up the kernel configuration is to run **make defconfig**. This will set the base configuration to a good state that takes your current system architecture into account.

Be sure to enable/disable/set the following features or the system might not work correctly or boot at all:

```
General setup --->
  [ ] Compile the kernel with warnings as errors                        [WERROR]
  CPU/Task time and stats accounting --->
    [*] Pressure stall information tracking                               [PSI]
    [ ]   Require boot parameter to enable pressure stall information tracking
                                                          ...  [PSI_DEFAULT_DISABLED]
  < > Enable kernel headers through /sys/kernel/kheaders.tar.xz      [IKHEADERS]
  [*] Control Group support --->                                        [CGROUPS]
    [*] Memory controller                                                 [MEMCG]
  [ ] Configure standard kernel features (expert users) --->             [EXPERT]

Processor type and features --->
  [*] Build a relocatable kernel                                    [RELOCATABLE]
  [*]   Randomize the address of the kernel image (KASLR)       [RANDOMIZE_BASE]

General architecture-dependent options --->
  [*] Stack Protector buffer overflow detection                 [STACKPROTECTOR]
  [*]   Strong Stack Protector                              [STACKPROTECTOR_STRONG]

Device Drivers --->
  Generic Driver Options --->
    [ ] Support for uevent helper                                [UEVENT_HELPER]
    [*] Maintain a devtmpfs filesystem to mount at /dev               [DEVTMPFS]
    [*]   Automount devtmpfs at /dev, after the kernel mounted the rootfs
                                                                 ...  [DEVTMPFS_MOUNT]
  Graphics support --->
    < /*/M> Direct Rendering Manager (XFree86 4.1.0 and higher DRI support) --->
                                                                      ...  [DRM]
    # If [DRM] is selected as * or M, this must be selected:
    [ /*]   Enable legacy fbdev support for your modesetting driver
                                                          ...  [DRM_FBDEV_EMULATION]
    Console display driver support --->
      # If [DRM] is selected as * or M, this must be selected:
      [ /*] Framebuffer Console support                    [FRAMEBUFFER_CONSOLE]
```

Enable some additional features if you are building a 64-bit system. If you are using menuconfig, enable them in the order of *CONFIG_PCI_MSI* first, then *CONFIG_IRQ_REMAP*, at last *CONFIG_X86_X2APIC* because an option only shows up after its dependencies are selected.

```
Processor type and features --->
  [*] Support x2apic                                                 [X86_X2APIC]

Device Drivers --->
  [*] PCI support --->                                                      [PCI]
    [*] Message Signaled Interrupts (MSI and MSI-X)                      [PCI_MSI]
  [*] IOMMU Hardware Support --->                                   [IOMMU_SUPPORT]
    [*] Support for Interrupt Remapping                                 [IRQ_REMAP]
```

If you are building a 32-bit system running on a hardware with RAM more than 4GB, adjust the configuration so the kernel will be able to use up to 64GB physical RAM:

```
Processor type and features --->
  High Memory Support --->
    (X) 64GB                                                         [HIGHMEM64G]
```

If the partition for the LFS system is in a NVME SSD (i. e. the device node for the partition is `/dev/nvme*` instead of `/dev/sd*`), enable NVME support or the LFS system won't boot:

```
Device Drivers --->
  NVME Support --->
    <*> NVM Express block device                                    [BLK_DEV_NVME]
```

There are several other options that may be desired depending on the requirements for the system. For a list of options needed for BLFS packages, see the *BLFS Index of Kernel Settings*.

Note

If your host hardware is using UEFI and you wish to boot the LFS system with it, you should adjust some kernel configuration following *the BLFS page* **even if you'll use the UEFI bootloader from the host distro**.

The rationale for the above configuration items:

`Randomize the address of the kernel image (KASLR)`
Enable ASLR for kernel image, to mitigate some attacks based on fixed addresses of sensitive data or code in the kernel.

`Compile the kernel with warnings as errors`
This may cause building failure if the compiler and/or configuration are different from those of the kernel developers.

`Enable kernel headers through /sys/kernel/kheaders.tar.xz`
This will require **cpio** building the kernel. **cpio** is not installed by LFS.

`Configure standard kernel features (expert users)`
This will make some options show up in the configuration interface but changing those options may be dangerous. Do not use this unless you know what you are doing.

`Strong Stack Protector`
Enable SSP for the kernel. We've enabled it for the entire userspace with *`--enable-default-ssp`* configuring GCC, but the kernel does not use GCC default setting for SSP. We enable it explicitly here.

`Support for uevent helper`
Having this option set may interfere with device management when using Udev.

`Maintain a devtmpfs`
This will create automated device nodes which are populated by the kernel, even without Udev running. Udev then runs on top of this, managing permissions and adding symlinks. This configuration item is required for all users of Udev.

`Automount devtmpfs at /dev`
This will mount the kernel view of the devices on /dev upon switching to root filesystem just before starting init.

`Enable legacy fbdev support for your modesetting driver` and *`Framebuffer Console support`*
These are needed to display the Linux console on a GPU driven by a DRI (Direct Rendering Infrastructure) driver. If `CONFIG_DRM` (Direct Rendering Manager) is enabled, you should enable these two options as well or you'll see a blank screen once the DRI driver is loaded.

`Support x2apic`
Support running the interrupt controller of 64-bit x86 processors in x2APIC mode. x2APIC may be enabled by firmware on 64-bit x86 systems, and a kernel without this option enabled will panic on boot if x2APIC is enabled by firmware. This option has no effect, but also does no harm if x2APIC is disabled by the firmware.

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 unpacked linux-6.7.4 directory. However, we do not recommend this option. It is often better to explore all the configuration menus and create the kernel configuration from scratch.

Compile the kernel image and modules:

```
make
```

If using kernel modules, module configuration in /etc/modprobe.d may be required. Information pertaining to modules and kernel configuration is located in Section 9.3, “Overview of Device and Module Handling” and in the kernel documentation in the linux-6.7.4/Documentation directory. Also, *modprobe.d(5)* may be of interest.

Unless module support has been disabled in the kernel configuration, install the modules with:

```
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.

Caution

If you've decided to use a separate /boot partition for the LFS system (maybe sharing a /boot partition with the host distro) , the files copied below should go there. The easiest way to do that is to create the entry for /boot in /etc/fstab first (read the previous section for details), then issue the following command as the root user in the *chroot environment*:

```
mount /boot
```

The path to the device node is omitted in the command because **mount** can read it from /etc/fstab.

The path to the kernel image may vary depending on the platform being used. The filename below can be changed to suit your taste, but the stem of the filename should be *vmlinuz* to be compatible with the automatic setup of the boot process described in the next section. The following command assumes an x86 architecture:

```
cp -iv arch/x86/boot/bzImage /boot/vmlinuz-6.7.4-lfs-12.1
```

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

```
cp -iv System.map /boot/System.map-6.7.4
```

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 -iv .config /boot/config-6.7.4
```

Install the documentation for the Linux kernel:

```
cp -r Documentation -T /usr/share/doc/linux-6.7.4
```

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.

Note

In many cases, the configuration of the kernel will need to be updated for packages that will be installed later in BLFS. Unlike other packages, it is not necessary to remove the kernel source tree after the newly built kernel is installed.

If the kernel source tree is going to be retained, run **chown -R 0:0** on the `linux-6.7.4` 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 an LFS system as it can cause problems for packages you may wish to build once your base LFS system is complete.

Warning

The headers in the system's `include` directory (`/usr/include`) should *always* be the ones against which Glibc was compiled, that is, the sanitised headers installed in Section 5.4, “Linux-6.7.4 API Headers”. Therefore, they should *never* be replaced by either the raw kernel headers or any other kernel sanitized headers.

10.3.2. Configuring Linux Module Load Order

Most of the time Linux modules are loaded automatically, but sometimes it needs some specific direction. The program that loads modules, **modprobe** or **insmod**, uses `/etc/modprobe.d/usb.conf` for this purpose. This file needs to be created so that if the USB drivers (ehci_hcd, ohci_hcd and uhci_hcd) have been built as modules, they will be loaded in the correct order; ehci_hcd needs to be loaded prior to ohci_hcd and uhci_hcd in order to avoid a warning being output at boot time.

Create a new file `/etc/modprobe.d/usb.conf` by running the following:

```
install -v -m755 -d /etc/modprobe.d
cat > /etc/modprobe.d/usb.conf << "EOF"
# Begin /etc/modprobe.d/usb.conf

install ohci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i ohci_hcd ; true
install uhci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i uhci_hcd ; true

# End /etc/modprobe.d/usb.conf
EOF
```

10.3.3. Contents of Linux

Installed files: config-6.7.4, vmlinuz-6.7.4-lfs-12.1, and System.map-6.7.4
Installed directories: /lib/modules, /usr/share/doc/linux-6.7.4

Short Descriptions

`config-6.7.4` Contains all the configuration selections for the kernel

`vmlinuz-6.7.4-lfs-12.1`	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-6.7.4`	A list of addresses and symbols; it maps the entry points and addresses of all the functions and data structures in the kernel

10.4. Using GRUB to Set Up the Boot Process

Note

If your system has UEFI support and you wish to boot LFS with UEFI, you should skip the instructions in this page but still learn the syntax of `grub.cfg` and the method to specify a partition in the file from this page, and configure GRUB with UEFI support using the instructions provided in *the BLFS page*.

10.4.1. Introduction

Warning

Configuring GRUB incorrectly can render your system inoperable without an alternate boot device such as a CD-ROM or bootable USB drive. This section is not required to boot your LFS system. You may just want to modify your current boot loader, e.g. Grub-Legacy, GRUB2, or LILO.

Ensure that an emergency boot disk is ready to “rescue” the computer if the computer becomes unusable (un-bootable). If you do not already have a boot device, you can create one. In order for the procedure below to work, you need to jump ahead to BLFS and install **`xorriso`** from the *libisoburn* package.

```
cd /tmp
grub-mkrescue --output=grub-img.iso
xorriso -as cdrecord -v dev=/dev/cdrw blank=as_needed grub-img.iso
```

10.4.2. GRUB Naming Conventions

GRUB uses its own naming structure for drives and partitions in the form of *(hdn,m)*, where *n* is the hard drive number and *m* is the partition number. The hard drive numbers start from zero, but the partition numbers start from one for normal partitions (from five for extended partitions). Note that this is different from earlier versions where both numbers started from zero. For example, partition `sda1` is *(hd0,1)* to GRUB and `sdb3` is *(hd1,3)*. In contrast to Linux, GRUB does not consider CD-ROM drives to be hard drives. For example, if using a CD on `hdb` and a second hard drive on `hdc`, that second hard drive would still be *(hd1)*.

10.4.3. Setting Up the Configuration

GRUB works by writing data to the first physical track of the hard disk. This area is not part of any file system. The programs there access GRUB modules in the boot partition. The default location is /boot/grub/.

The location of the boot partition is a choice of the user that affects the configuration. One recommendation is to have a separate small (suggested size is 200 MB) partition just for boot information. That way each build, whether LFS or some commercial distro, can access the same boot files and access can be made from any booted system. If you choose to do this, you will need to mount the separate partition, move all files in the current `/boot` directory (e.g. the Linux kernel you just built in the previous section) to the new partition. You will then need to unmount the partition and remount it as `/boot`. If you do this, be sure to update `/etc/fstab`.

Leaving `/boot` on the current LFS partition will also work, but configuration for multiple systems is more difficult.

Using the above information, determine the appropriate designator for the root partition (or boot partition, if a separate one is used). For the following example, it is assumed that the root (or separate boot) partition is `sda2`.

Install the GRUB files into `/boot/grub` and set up the boot track:

Warning

The following command will overwrite the current boot loader. Do not run the command if this is not desired, for example, if using a third party boot manager to manage the Master Boot Record (MBR).

```
grub-install /dev/sda
```

Note

If the system has been booted using UEFI, **grub-install** will try to install files for the *x86_64-efi* target, but those files have not been installed in Chapter 8. If this is the case, add `--target i386-pc` to the command above.

10.4.4. Creating the GRUB Configuration File

Generate `/boot/grub/grub.cfg`:

```
cat > /boot/grub/grub.cfg << "EOF"
# Begin /boot/grub/grub.cfg
set default=0
set timeout=5

insmod part_gpt
insmod ext2
set root=(hd0,2)

menuentry "GNU/Linux, Linux 6.7.4-lfs-12.1" {
        linux   /boot/vmlinuz-6.7.4-lfs-12.1 root=/dev/sda2 ro
}
EOF
```

The **insmod** commands load the GRUB modules named `part_gpt` and `ext2`. Despite the naming, `ext2` actually supports `ext2`, `ext3`, and `ext4` filesystems. The **grub-install** command has embedded some modules into the main GRUB image (installed into the MBR or the GRUB BIOS partition) to access the other modules (in `/boot/grub/i386-pc`) without a chicken-or-egg issue, so with a typical configuration these two modules are already embedded and those two **insmod** commands will do nothing. But they do no harm anyway, and they may be needed with some rare configurations.

Note

From GRUB's perspective, the kernel files are relative to the partition used. If you used a separate /boot partition, remove /boot from the above *linux* line. You will also need to change the *set root* line to point to the boot partition.

Note

The GRUB designator for a partition may change if you added or removed some disks (including removable disks like USB thumb devices). The change may cause boot failure because `grub.cfg` refers to some "old" designators. If you wish to avoid such a problem, you may use the UUID of a partition and the UUID of a filesystem instead of a GRUB designator to specify a device. Run **lsblk -o UUID,PARTUUID,PATH,MOUNTPOINT** to show the UUIDs of your filesystems (in the `UUID` column) and partitions (in the `PARTUUID` column). Then replace `set root=(hdx,y)` with `search --set=root --fs-uuid <UUID of the filesystem where the kernel is installed>`, and replace `root=/dev/sda2` with `root=PARTUUID=<UUID of the partition where LFS is built>`.

Note that the UUID of a partition is completely different from the UUID of the filesystem in this partition. Some online resources may instruct you to use `root=UUID=<filesystem UUID>` instead of `root=PARTUUID=<partition UUID>`, but doing so will require an initramfs, which is beyond the scope of LFS.

The name of the device node for a partition in `/dev` may also change (this is less likely than a GRUB designator change). You can also replace paths to device nodes like `/dev/sda1` with `PARTUUID=<partition UUID>`, in `/etc/fstab`, to avoid a potential boot failure in case the device node name has changed.

GRUB is an extremely powerful program and it provides a tremendous number of options for booting from a wide variety of devices, operating systems, and partition types. There are also many options for customization such as graphical splash screens, playing sounds, mouse input, etc. The details of these options are beyond the scope of this introduction.

Caution

There is a command, grub-mkconfig, that can write a configuration file automatically. It uses a set of scripts in /etc/grub.d/ and will destroy any customizations that you make. These scripts are designed primarily for non-source distributions and are not recommended for LFS. If you install a commercial Linux distribution, there is a good chance that this program will be run. Be sure to back up your grub.cfg file.

Chapter 11. The End

11.1. The End

Well done! The new LFS system is installed! We wish you much success with your shiny new custom-built Linux system.

It may be a good idea to create an `/etc/lfs-release` file. By having this file, it is very easy for you (and for us if you need to ask for help at some point) to find out which LFS version is installed on the system. Create this file by running:

```
echo 12.1 > /etc/lfs-release
```

Two files describing the installed system may be used by packages that can be installed on the system later, either in binary form or by building them.

The first one shows the status of your new system with respect to the Linux Standards Base (LSB). To create this file, run:

```
cat > /etc/lsb-release << "EOF"
DISTRIB_ID="Linux From Scratch"
DISTRIB_RELEASE="12.1"
DISTRIB_CODENAME="<your name here>"
DISTRIB_DESCRIPTION="Linux From Scratch"
EOF
```

The second one contains roughly the same information, and is used by systemd and some graphical desktop environments. To create this file, run:

```
cat > /etc/os-release << "EOF"
NAME="Linux From Scratch"
VERSION="12.1"
ID=lfs
PRETTY_NAME="Linux From Scratch 12.1"
VERSION_CODENAME="<your name here>"
HOME_URL="https://www.linuxfromscratch.org/lfs/"
EOF
```

Be sure to customize the fields 'DISTRIB_CODENAME' and 'VERSION_CODENAME' to make the system uniquely yours.

11.2. Get Counted

Now that you have finished the book, do you want to be counted as an LFS user? Head over to *https://www.linuxfromscratch.org/cgi-bin/lfscounter.php* and register as an LFS user by entering your name and the first LFS version you have used.

Let's reboot into LFS now.

11.3. Rebooting the System

Now that all of the software has been installed, it is time to reboot your computer. However, there are still a few things to check. Here are some suggestions:

- Install any *firmware* needed if the kernel driver for your hardware requires some firmware files to function properly.
- Ensure a password is set for the `root` user.

- A review of the following configuration files is also appropriate at this point.
  - /etc/bashrc
  - /etc/dircolors
  - /etc/fstab
  - /etc/hosts
  - /etc/inputrc
  - /etc/profile
  - /etc/resolv.conf
  - /etc/vimrc
  - /root/.bash_profile
  - /root/.bashrc
  - /etc/sysconfig/ifconfig.eth0

Now that we have said that, let's move on to booting our shiny new LFS installation for the first time! *First exit from the chroot environment*:

```
logout
```

Then unmount the virtual file systems:

```
umount -v $LFS/dev/pts
mountpoint -q $LFS/dev/shm && umount -v $LFS/dev/shm
umount -v $LFS/dev
umount -v $LFS/run
umount -v $LFS/proc
umount -v $LFS/sys
```

If multiple partitions were created, unmount the other partitions before unmounting the main one, like this:

```
umount -v $LFS/home
umount -v $LFS
```

Unmount the LFS file system itself:

```
umount -v $LFS
```

Now, reboot the system.

Assuming the GRUB boot loader was set up as outlined earlier, the menu is set to boot *LFS 12.1* automatically.

When the reboot is complete, the LFS system is ready for use. What you will see is a simple “login: ” prompt. At this point, you can proceed to *the BLFS Book* where you can add more software to suit your needs.

If your reboot is **not** successful, it is time to troubleshoot. For hints on solving initial booting problems, see *https://www.linuxfromscratch.org/lfs/troubleshooting.html*.

11.4. Additional Resources

Thank you for reading this LFS book. We hope that you have found this book helpful and have learned more about the system creation process.

Now that the LFS system is installed, you may be wondering “What next?” To answer that question, we have compiled a list of resources for you.

- Maintenance

  Bugs and security notices are reported regularly for all software. Since an LFS 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:

  - *LFS Security Advisories*

    This is a list of security vulnerabilities discovered in the LFS book after it's published.

  - *Open Source Security Mailing List*

    This is a mailing list for discussion of security flaws, concepts, and practices in the Open Source community.

- LFS Hints

  The LFS Hints are a collection of educational documents submitted by volunteers in the LFS community. The hints are available at *https://www.linuxfromscratch.org/hints/downloads/files/*.

- Mailing lists

  There are several LFS mailing lists you may subscribe to if you are in need of help, want to stay current with the latest developments, want to contribute to the project, and more. See Chapter 1 - Mailing Lists for more information.

- The Linux Documentation Project

  The goal of The Linux Documentation Project (TLDP) is to collaborate on all of the issues of Linux documentation. The TLDP features a large collection of HOWTOs, guides, and man pages. It is located at *https://tldp.org/*.

11.5. Getting Started After LFS

11.5.1. Deciding what to do next

Now that LFS is complete and you have a bootable system, what do you do? The next step is to decide how to use it. Generally, there are two broad categories to consider: workstation or server. Indeed, these categories are not mutually exclusive. The applications needed for each category can be combined onto a single system, but let's look at them separately for now.

A server is the simpler category. Generally this consists of a web server such as the *Apache HTTP Server* and a database server such as *MariaDB*. However other services are possible. The operating system embedded in a single use device falls into this category.

On the other hand, a workstation is much more complex. It generally requires a graphical user environment such as *LXDE*, *XFCE*, *KDE*, or *Gnome* based on a basic *graphical environment* and several graphical based applications such as the *Firefox web browser*, *Thunderbird email client*, or *LibreOffice office suite*. These applications require many (several hundred depending on desired capabilities) more packages of support applications and libraries.

In addition to the above, there is a set of applications for system management for all kinds of systems. These applications are all in the BLFS book. Not all packages are needed in every environments. For example *dhcpcd*, is not normally appropriate for a server and *wireless_tools*, are normally only useful for a laptop system.

11.5.2. Working in a basic LFS environment

When you initially boot into LFS, you have all the internal tools to build additional packages. Unfortunately, the user environment is quite sparse. There are a couple of ways to improve this:

11.5.2.1. Work from the LFS host in chroot

This method provides a complete graphical environment where a full featured browser and copy/paste capabilities are available. This method allows using applications like the host's version of wget to download package sources to a location available when working in the chroot environment.

In order to properly build packages in chroot, you will also need to remember to mount the virtual file systems if they are not already mounted. One way to do this is to create a script on the **HOST** system:

```
cat > ~/mount-virt.sh << "EOF"
#!/bin/bash

function mountbind
{
   if ! mountpoint $LFS/$1 >/dev/null; then
     $SUDO mount --bind /$1 $LFS/$1
     echo $LFS/$1 mounted
   else
     echo $LFS/$1 already mounted
   fi
}

function mounttype
{
   if ! mountpoint $LFS/$1 >/dev/null; then
     $SUDO mount -t $2 $3 $4 $5 $LFS/$1
     echo $LFS/$1 mounted
   else
     echo $LFS/$1 already mounted
   fi
}

if [ $EUID -ne 0 ]; then
  SUDO=sudo
else
  SUDO=""
fi

if [ x$LFS == x ]; then
  echo "LFS not set"
  exit 1
fi

mountbind dev
mounttype dev/pts devpts devpts -o gid=5,mode=620
mounttype proc    proc   proc
mounttype sys     sysfs  sysfs
mounttype run     tmpfs  run
if [ -h $LFS/dev/shm ]; then
  install -v -d -m 1777 $LFS$(realpath /dev/shm)
else
  mounttype dev/shm tmpfs tmpfs -o nosuid,nodev
fi

#mountbind usr/src
#mountbind boot
#mountbind home
EOF
```

Note that the last three commands in the script are commented out. These are useful if those directories are mounted as separate partitions on the host system and will be mounted when booting the completed LFS/BLFS system.

The script can be run with **bash ~/mount-virt.sh** as either a regular user (recommended) or as `root`. If run as a regular user, sudo is required on the host system.

Another issue pointed out by the script is where to store downloaded package files. This location is arbitrary. It can be in a regular user's home directory such as ~/sources or in a global location like /usr/src. Our recommendation is not to mix BLFS sources and LFS sources in (from the chroot environment) /sources. In any case, the packages must be accessible inside the chroot environment.

A last convenience feature presented here is to streamline the process of entering the chroot environment. This can be done with an alias placed in a user's ~/.bashrc file on the host system:

```
alias lfs='sudo /usr/sbin/chroot /mnt/lfs /usr/bin/env -i HOME=/root TERM="$TERM" PS1="\u:\w\\\$ "
PATH=/bin:/usr/bin:/sbin:/usr/sbin /bin/bash --login'
```

This alias is a little tricky because of the quoting and levels of backslash characters. It must be all on a single line. The above command has been split in two for presentation purposes.

11.5.2.2. Work remotely via ssh

This method also provides a full graphical environment, but first requires installing *sshd* on the LFS system, usually in chroot. It also requires a second computer. This method has the advantage of being simple by not requiring the complexity of the chroot environment. It also uses your LFS built kernel for all additional packages and still provides a complete system for installing packages.

You may use the **scp** command to upload the package sources to be built onto the LFS system. If you want to download the sources onto the LFS system directly instead, install *libtasn1*, *p11-kit*, *make-ca*, and *wget* in chroot (or upload their sources using **scp** after booting the LFS system).

11.5.2.3. Work from the LFS command line

This method requires installing *libtasn1*, *p11-kit*, *make-ca*, *wget*, *gpm*, and *links* (or *lynx*) in chroot and then rebooting into the new LFS system. At this point the default system has six virtual consoles. Switching consoles is as easy as using the **Alt**+**Fx** key combinations where **Fx** is between **F1** and **F6**. The **Alt**+$\leftarrow$ and **Alt**+$\rightarrow$ combinations also will change the console.

At this point you can log into two different virtual consoles and run the links or lynx browser in one console and bash in the other. GPM then allows copying commands from the browser with the left mouse button, switching consoles, and pasting into the other console.

Note

As a side note, switching of virtual consoles can also be done from an X Window instance with the **Ctrl**+**Alt**+**Fx** key combination, but the mouse copy operation does not work between the graphical interface and a virtual console. You can return to the X Window display with the **Ctrl**+**Alt**+**Fx** combination, where **Fx** is usually **F1** but may be **F7**.

Part V. Appendices

Appendix A. Acronyms and Terms

ABI Application Binary Interface

ALFS Automated Linux From Scratch

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BIOS Basic Input/Output System

BLFS Beyond Linux From Scratch

BSD Berkeley Software Distribution

chroot change root

CMOS Complementary Metal Oxide Semiconductor

COS Class Of Service

CPU Central Processing Unit

CRC Cyclic Redundancy Check

CVS Concurrent Versions System

DHCP Dynamic Host Configuration Protocol

DNS Domain Name Service

EGA Enhanced Graphics Adapter

ELF Executable and Linkable Format

EOF End of File

EQN equation

ext2 second extended file system

ext3 third extended file system

ext4 fourth 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 Gigabytes

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
OSS Open Sound System
PCH Pre-Compiled Headers
PCRE Perl Compatible Regular Expression
PID Process Identifier
PTY pseudo terminal
QOS Quality Of Service
RAM Random Access Memory
RPC Remote Procedure Call
RTC Real Time Clock
SBU Standard Build Unit
SCO The Santa Cruz Operation
SHA1 Secure-Hash Algorithm 1
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
- *Bruce Dubbs* <bdubbs@linuxfromscratch.org> – LFS Managing Editor
- *Jim Gifford* <jim@linuxfromscratch.org> – CLFS Project Co-Leader
- *Pierre Labastie* <pierre@linuxfromscratch.org> – BLFS Editor and ALFS Lead
- *DJ Lucas* <dj@linuxfromscratch.org> – LFS and BLFS Editor
- *Ken Moffat* <ken@linuxfromscratch.org> – BLFS Editor
- 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@macana-es.com> – Spanish LFS translation project
- *Johan Lenglet* <johan@linuxfromscratch.org> – French LFS translation project until 2008
- *Jean-Philippe Mengual* <jmengual@linuxfromscratch.org> – French LFS translation project 2008-2016
- *Julien Lepiller* <jlepiller@linuxfromscratch.org> – French LFS translation project 2017-present
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Portuguese LFS translation project historical
- *Jamenson Espindula* <jafesp@gmail.com> – Portuguese LFS translation project 2022-present
- *Thomas Reitelbach* <tr@erdfunkstelle.de> – German LFS translation project

Mirror Maintainers

North American Mirrors

- *Scott Kveton* <scott@osuosl.org> – lfs.oregonstate.edu mirror
- *William Astle* <lost@l-w.net> – ca.linuxfromscratch.org mirror
- *Eujon Sellers* <jpolen@rackspace.com> – lfs.introspeed.com mirror
- *Justin Knierim* <tim@idge.net> – lfs-matrix.net mirror

South American Mirrors

- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.info mirror
- *Luis Falcon* <Luis Falcon> – torredehanoi.org mirror

European Mirrors

- *Guido Passet* <guido@primerelay.net> – nl.linuxfromscratch.org mirror
- *Bastiaan Jacques* <baafie@planet.nl> – lfs.pagefault.net mirror
- *Sven Cranshoff* <sven.cranshoff@lineo.be> – lfs.lineo.be mirror

- Scarlet Belgium – lfs.scarlet.be mirror
- *Sebastian Faulborn* <info@aliensoft.org> – lfs.aliensoft.org mirror
- *Stuart Fox* <stuart@dontuse.ms> – lfs.dontuse.ms mirror
- *Ralf Uhlemann* <admin@realhost.de> – lfs.oss-mirror.org mirror
- *Antonin Sprinzl* <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org mirror
- *Fredrik Danerklint* <fredan-lfs@fredan.org> – se.linuxfromscratch.org mirror
- *Franck* <franck@linuxpourtous.com> – lfs.linuxpourtous.com mirror
- *Philippe Baque* <baque@cict.fr> – lfs.cict.fr mirror
- *Vitaly Chekasin* <gyouja@pilgrims.ru> – lfs.pilgrims.ru mirror
- *Benjamin Heil* <kontakt@wankoo.org> – lfs.wankoo.org mirror
- *Anton Maisak* <info@linuxfromscratch.org.ru> – linuxfromscratch.org.ru mirror

Asian Mirrors

- *Satit Phermsawang* <satit@wbac.ac.th> – lfs.phayoune.org mirror
- *Shizunet Co.,Ltd.* <info@shizu-net.jp> – lfs.mirror.shizu-net.jp 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
- Archaic <archaic@linuxfromscratch.org> – LFS Technical Writer/Editor, HLFS Project Leader, BLFS Editor, Hints and Patches Project Maintainer
- *Matthew Burgess* <matthew@linuxfromscratch.org> – LFS Project Leader, LFS Technical Writer/Editor
- *Nathan Coulson* <nathan@linuxfromscratch.org> – LFS-Bootscripts Maintainer
- Timothy Bauscher
- Robert Briggs
- Ian Chilton
- *Jeroen Coumans* <jeroen@linuxfromscratch.org> – Website Developer, FAQ Maintainer
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – LFS/BLFS/HLFS XML and XSL Maintainer
- Alex Groenewoud – LFS Technical Writer
- Marc Heerdink
- *Jeremy Huntwork* <jhuntwork@linuxfromscratch.org> – LFS Technical Writer, LFS LiveCD Maintainer
- *Bryan Kadzban* <bryan@linuxfromscratch.org> – LFS Technical Writer
- Mark Hymers
- Seth W. Klein – FAQ maintainer
- *Nicholas Leippe* <nicholas@linuxfromscratch.org> – Wiki Maintainer

- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Website Backend-Scripts Maintainer
- *Randy McMurchy* <randy@linuxfromscratch.org> – BLFS Project Leader, LFS Editor
- *Dan Nicholson* <dnicholson@linuxfromscratch.org> – LFS and BLFS Editor
- *Alexander E. Patrakov* <alexander@linuxfromscratch.org> – LFS Technical Writer, LFS Internationalization Editor, LFS Live CD Maintainer
- Simon Perreault
- *Scot Mc Pherson* <scot@linuxfromscratch.org> – LFS NNTP Gateway Maintainer
- *Douglas R. Reno* <renodr@linuxfromscratch.org> – Systemd Editor
- *Ryan Oliver* <ryan@linuxfromscratch.org> – CLFS Project Co-Leader
- *Greg Schafer* <gschafer@zip.com.au> – LFS Technical Writer and Architect of the Next Generation 64-bit-enabling Build Method
- Jesse Tie-Ten-Quee – LFS Technical Writer
- *James Robertson* <jwrober@linuxfromscratch.org> – Bugzilla Maintainer
- *Tushar Teredesai* <tushar@linuxfromscratch.org> – BLFS Book Editor, Hints and Patches Project Leader
- *Jeremy Utley* <jeremy@linuxfromscratch.org> – LFS Technical Writer, Bugzilla Maintainer, LFS-Bootscripts Maintainer
- *Zack Winkles* <zwinkles@gmail.com> – LFS Technical Writer

Appendix C. Dependencies

Every package built in LFS 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 LFS is very important. The purpose of this page is to document the dependencies of each package built in LFS.

For each package that is built, there are three, and sometimes up to five types of dependencies listed below. The first lists what other packages need to be available in order to compile and install the package in question. The second lists the packages that must be available when any programs or libraries from the package are used at runtime. The third lists what packages, in addition to those on the first list, need to be available in order to run the test suites. The fourth list of dependencies are packages that require this package to be built and installed in its final location before they are built and installed.

The last list of dependencies are optional packages that are not addressed in LFS, but could be useful to the user. These packages may have additional mandatory or optional dependencies of their own. For these dependencies, the recommended practice is to install them after completion of the LFS book and then go back and rebuild the LFS package. In several cases, re-installation is addressed in BLFS.

Acl

Installation depends on: Attr, Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, and Texinfo
Required at runtime: Attr and Glibc
Test suite depends on: Automake, Diffutils, Findutils, and Libtool
Must be installed before: Coreutils, Sed, Tar, and Vim
Optional dependencies: None

Attr

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, Sed, and Texinfo
Required at runtime: Glibc
Test suite depends on: Automake, Diffutils, Findutils, and Libtool
Must be installed before: Acl, Libcap, and Patch
Optional dependencies: None

Autoconf

Installation depends on: Bash, Coreutils, Grep, M4, Make, Perl, Sed, and Texinfo
Required at runtime: Bash, Coreutils, Grep, M4, Make, Sed, and Texinfo
Test suite depends on: Automake, Diffutils, Findutils, GCC, and Libtool
Must be installed before: Automake and Coreutils
Optional dependencies: *Emacs*

Automake

Installation depends on: Autoconf, Bash, Coreutils, Gettext, Grep, M4, Make, Perl, Sed, and Texinfo
Required at runtime: Bash, Coreutils, Grep, M4, Sed, and Texinfo
Test suite depends on: Binutils, Bison, Bzip2, DejaGNU, Diffutils, Expect, Findutils, Flex, GCC, Gettext, Gzip, Libtool, and Tar
Must be installed before: Coreutils
Optional dependencies: None

Bash

Installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Readline, Sed, and Texinfo
Required at runtime: Glibc, Ncurses, and Readline
Test suite depends on: Expect and Shadow
Must be installed before: None
Optional dependencies: *Xorg*

Bc

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, and Readline
Required at runtime: Glibc, Ncurses, and Readline
Test suite depends on: Gawk
Must be installed before: Linux
Optional dependencies: None

Binutils

Installation depends on: Bash, Binutils, Coreutils, Diffutils, File, Flex, Gawk, GCC, Glibc, Grep, Make, Perl, Pkgconf, Sed, Texinfo, Zlib, and Zstd
Required at runtime: Glibc, Zlib, and Zstd
Test suite depends on: DejaGNU and Expect
Must be installed before: None
Optional dependencies: *Elfutils* and *Jansson*

Bison

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, and Sed
Required at runtime: Glibc
Test suite depends on: Diffutils, Findutils, and Flex
Must be installed before: Kbd and Tar
Optional dependencies: *Doxygen*

Bzip2

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make, and Patch
Required at runtime: Glibc
Test suite depends on: None
Must be installed before: File and Libelf
Optional dependencies: None

Check

Installation depends on: Gawk, GCC, Grep, Make, Sed, and Texinfo
Required at runtime: Bash and Gawk
Test suite depends on: None
Must be installed before: None
Optional dependencies: *libsubunit* and *patchutils*

Coreutils

Installation depends on: Autoconf, Automake, Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Libcap, Make, OpenSSL, Patch, Perl, Sed, and Texinfo
Required at runtime: Glibc
Test suite depends on: Diffutils, E2fsprogs, Findutils, Shadow, and Util-linux
Must be installed before: Bash, Diffutils, Findutils, Man-DB, and Udev
Optional dependencies: *Expect.pm* and *IO::Tty*

DejaGNU

Installation depends on: Bash, Coreutils, Diffutils, Expect, GCC, Grep, Make, Sed, and Texinfo
Required at runtime: Expect and Bash
Test suite depends on: None
Must be installed before: None
Optional dependencies: None

Diffutils

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, and Texinfo
Required at runtime: Glibc
Test suite depends on: Perl
Must be installed before: None
Optional dependencies: None

E2fsprogs

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Gzip, Make, Pkgconf, Sed, Texinfo, and Util-linux
Required at runtime: Glibc and Util-linux
Test suite depends on: Procps-ng and Psmisc
Must be installed before: None
Optional dependencies: None

Expat

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, and Sed
Required at runtime: Glibc
Test suite depends on: None
Must be installed before: Python and XML::Parser
Optional dependencies: None

Expect

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Patch, Sed, and Tcl
Required at runtime: Glibc and Tcl
Test suite depends on: None
Must be installed before: None
Optional dependencies: *Tk*

File

Installation depends on: Bash, Binutils, Bzip2, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Xz, and Zlib
Required at runtime: Glibc, Bzip2, Xz, and Zlib
Test suite depends on: None
Must be installed before: None
Optional dependencies: *libseccomp*

Findutils

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, and Texinfo
Required at runtime: Bash and Glibc
Test suite depends on: DejaGNU, Diffutils, and Expect
Must be installed before: None
Optional dependencies: None

Flex

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Patch, Sed, and Texinfo
Required at runtime: Bash, Glibc, and M4
Test suite depends on: Bison and Gawk
Must be installed before: Binutils, IProute2, Kbd, Kmod, and Man-DB
Optional dependencies: None

Flit-Core

Installation depends on: Python
Required at runtime: Python
Test suite depends on: No test suite available
Must be installed before: Wheel
Optional dependencies: *pytest* and *testpath*

Gawk

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Make, MPFR, Patch, Readline, Sed, and Texinfo
Required at runtime: Bash, Glibc, and Mpfr
Test suite depends on: Diffutils
Must be installed before: None
Optional dependencies: *libsigsegv*

GCC

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, GMP, Grep, Libxcrypt, M4, Make, MPC, MPFR, Patch, Perl, Sed, Tar, Texinfo, and Zstd
Required at runtime: Bash, Binutils, Glibc, Mpc, and Python
Test suite depends on: DejaGNU, Expect, and Shadow
Must be installed before: None
Optional dependencies: *GDC*, *GNAT*, and *ISL*

GDBM

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Grep, Make, and Sed
Required at runtime: Bash, Glibc, and Readline
Test suite depends on: None
Must be installed before: None
Optional dependencies: None

Gettext

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Sed, and Texinfo
Required at runtime: Acl, Bash, Gcc, and Glibc
Test suite depends on: Diffutils, Perl, and Tcl
Must be installed before: Automake and Bison
Optional dependencies: *libunistring* and *libxml2*

Glibc

Installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Gzip, Linux API Headers, Make, Perl, Python, Sed, and Texinfo
Required at runtime: None
Test suite depends on: File
Must be installed before: None
Optional dependencies: None

GMP

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, M4, Make, Sed, and Texinfo
Required at runtime: GCC and Glibc
Test suite depends on: None
Must be installed before: MPFR and GCC
Optional dependencies: None

Gperf

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, and Make
Required at runtime: GCC and Glibc
Test suite depends on: Diffutils and Expect
Must be installed before: None
Optional dependencies: None

Grep

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed, and Texinfo
Required at runtime: Glibc
Test suite depends on: Gawk
Must be installed before: Man-DB
Optional dependencies: *PCRE2* and *libsigsegv*

Groff

Installation depends on: Bash, Binutils, Bison, Coreutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed, and Texinfo
Required at runtime: GCC, Glibc, and Perl
Test suite depends on: No test suite available
Must be installed before: Man-DB
Optional dependencies: *ghostscript* and *Uchardet*

GRUB

Installation depends on: Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Texinfo, and Xz
Required at runtime: Bash, GCC, Gettext, Glibc, Xz, and Sed.
Test suite depends on: None
Must be installed before: None
Optional dependencies: None

Gzip

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, and Texinfo
Required at runtime: Bash and Glibc
Test suite depends on: Diffutils and Less
Must be installed before: Man-DB
Optional dependencies: None

Iana-Etc

Installation depends on: Coreutils
Required at runtime: None
Test suite depends on: No test suite available
Must be installed before: Perl
Optional dependencies: None

Inetutils

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, Texinfo, and Zlib
Required at runtime: GCC, Glibc, Ncurses, and Readline
Test suite depends on: None
Must be installed before: Tar
Optional dependencies: None

Intltool

Installation depends on: Bash, Gawk, Glibc, Make, Perl, Sed, and XML::Parser
Required at runtime: Autoconf, Automake, Bash, Glibc, Grep, Perl, and Sed
Test suite depends on: Perl
Must be installed before: None
Optional dependencies: None

IProute2

Installation depends on: Bash, Bison, Coreutils, Flex, GCC, Glibc, Make, Libcap, Libelf, Linux API Headers, Pkgconf, and Zlib
Required at runtime: Bash, Coreutils, Glibc, Libcap, Libelf, and Zlib
Test suite depends on: No test suite available
Must be installed before: None
Optional dependencies: *Berkeley DB*, *iptables*, *libbpf*, *libmnl*, and *libtirpc*

Jinja2

Installation depends on: MarkupSafe, Python, Setuptools, and Wheel
Required at runtime: MarkupSafe and Python
Test suite depends on: No test suite available
Must be installed before: Udev
Optional dependencies: None

Kbd

Installation depends on: Bash, Binutils, Bison, Check, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch, and Sed
Required at runtime: Bash, Coreutils, and Glibc
Test suite depends on: None
Must be installed before: None
Optional dependencies: *Linux-PAM*

Kmod

Installation depends on: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, OpenSSL, Pkgconf, Sed, Xz, and Zlib
Required at runtime: Glibc, Xz, and Zlib
Test suite depends on: No test suite available
Must be installed before: Udev
Optional dependencies: None

Less

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, and Sed
Required at runtime: Glibc and Ncurses
Test suite depends on: No test suite available
Must be installed before: Gzip
Optional dependencies: *PCRE2* or *PCRE*

Libcap

Installation depends on: Attr, Bash, Binutils, Coreutils, GCC, Glibc, Perl, Make, and Sed
Required at runtime: Glibc
Test suite depends on: None
Must be installed before: IProute2 and Shadow
Optional dependencies: *Linux-PAM*

Libelf

Installation depends on: Bash, Binutils, Bzip2, Coreutils, GCC, Glibc, Make, Xz, Zlib, and Zstd
Required at runtime: Bzip2, Glibc, Xz, Zlib, and Zstd
Test suite depends on: None
Must be installed before: IProute2 and Linux
Optional dependencies: None

Libffi

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make, and Sed
Required at runtime: Glibc
Test suite depends on: DejaGnu
Must be installed before: Python
Optional dependencies: None

Libpipeline

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, and Texinfo
Required at runtime: Glibc
Test suite depends on: Check and Pkgconf
Must be installed before: Man-DB
Optional dependencies: None

Libtool

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, and Texinfo
Required at runtime: Autoconf, Automake, Bash, Binutils, Coreutils, File, GCC, Glibc, Grep, Make, and Sed
Test suite depends on: Autoconf, Automake, and Findutils
Must be installed before: None
Optional dependencies: None

Libxcrypt

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Perl, and Sed
Required at runtime: Glibc
Test suite depends on: None
Must be installed before: GCC, Perl, Python, Shadow, and Udev
Optional dependencies: None

Linux

Installation depends on: Bash, Bc, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Kmod, Libelf, Make, Ncurses, OpenSSL, Perl, and Sed
Required at runtime: None
Test suite depends on: No test suite available
Must be installed before: None
Optional dependencies: *cpio* and *LLVM* (with Clang)

Linux API Headers

Installation depends on: Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Perl, and Sed
Required at runtime: None
Test suite depends on: No test suite available
Must be installed before: None
Optional dependencies: None

M4

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, and Texinfo
Required at runtime: Bash and Glibc
Test suite depends on: Diffutils
Must be installed before: Autoconf and Bison
Optional dependencies: *libsigsegv*

Make

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, and Texinfo
Required at runtime: Glibc
Test suite depends on: Perl and Procps-ng
Must be installed before: None
Optional dependencies: *Guile*

Man-DB

Installation depends on: Bash, Binutils, Bzip2, Coreutils, Flex, GCC, GDBM, Gettext, Glibc, Grep, Groff, Gzip, Less, Libpipeline, Make, Pkgconf, Sed, and Xz
Required at runtime: Bash, GDBM, Groff, Glibc, Gzip, Less, Libpipeline, and Zlib
Test suite depends on: Util-linux
Must be installed before: None
Optional dependencies: *libseccomp* and *po4a*

Man-Pages

Installation depends on: Bash, Coreutils, and Make
Required at runtime: None
Test suite depends on: No test suite available
Must be installed before: None
Optional dependencies: None

MarkupSafe

Installation depends on: Python, Setuptools, and Wheel
Required at runtime: Python
Test suite depends on: No test suite available
Must be installed before: Jinja2
Optional dependencies: None

Meson

Installation depends on: Ninja, Python, Setuptools, and Wheel
Required at runtime: Python
Test suite depends on: No test suite available
Must be installed before: Udev
Optional dependencies: None

MPC

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, MPFR, Sed, and Texinfo
Required at runtime: Glibc, GMP, and MPFR
Test suite depends on: None
Must be installed before: GCC
Optional dependencies: None

MPFR

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, Sed, and Texinfo
Required at runtime: Glibc and GMP
Test suite depends on: None
Must be installed before: Gawk and GCC
Optional dependencies: None

Ncurses

Installation depends on: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Patch, and Sed
Required at runtime: Glibc
Test suite depends on: No test suite available
Must be installed before: Bash, GRUB, Inetutils, Less, Procps-ng, Psmisc, Readline, Texinfo, Util-linux, and Vim
Optional dependencies: None

Ninja

Installation depends on: Binutils, Coreutils, GCC, and Python
Required at runtime: GCC and Glibc
Test suite depends on: None
Must be installed before: Meson
Optional dependencies: *Asciidoc*, *Doxygen*, *Emacs*, and *re2c*

OpenSSL

Installation depends on: Binutils, Coreutils, GCC, Make, and Perl
Required at runtime: Glibc and Perl
Test suite depends on: None
Must be installed before: Coreutils, Kmod, Linux, and Udev
Optional dependencies: None

Patch

Installation depends on: Attr, Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, and Sed
Required at runtime: Attr and Glibc
Test suite depends on: Diffutils
Must be installed before: None
Optional dependencies: *Ed*

Perl

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, GDBM, Glibc, Grep, Libxcrypt, Make, Sed, and Zlib
Required at runtime: GDBM, Glibc, and Libxcrypt
Test suite depends on: Iana-Etc, Less, and Procps-ng
Must be installed before: Autoconf
Optional dependencies: *Berkeley DB*

Pkgconf

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, and Sed
Required at runtime: Glibc
Test suite depends on: None
Must be installed before: Binutils, E2fsprogs, IProute2, Kmod, Man-DB, Procps-ng, Python, Udev, and Util-linux
Optional dependencies: None

Procps-ng

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses, and Pkgconf
Required at runtime: Glibc
Test suite depends on: DejaGNU
Must be installed before: None
Optional dependencies: *elogind*

Psmisc

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, and Sed
Required at runtime: Glibc and Ncurses
Test suite depends on: No test suite available
Must be installed before: None
Optional dependencies: None

Python

Installation depends on: Bash, Binutils, Coreutils, Expat, GCC, Gdbm, Gettext, Glibc, Grep, Libffi, Libxcrypt, Make, Ncurses, OpenSSL, Pkgconf, Sed, and Util-linux
Required at runtime: Bzip2, Expat, Gdbm, Glibc, Libffi, Libxcrypt, Ncurses, OpenSSL, and Zlib
Test suite depends on: GDB and Valgrind
Must be installed before: Ninja
Optional dependencies: *Berkeley DB*, *libnsl*, *SQLite*, and *Tk*

Readline

Installation depends on: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, and Texinfo
Required at runtime: Glibc and Ncurses
Test suite depends on: No test suite available
Must be installed before: Bash, Bc, and Gawk
Optional dependencies: None

Sed

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, and Texinfo
Required at runtime: Acl, Attr, and Glibc
Test suite depends on: Diffutils and Gawk
Must be installed before: E2fsprogs, File, Libtool, and Shadow
Optional dependencies: None

Setuptools

Installation depends on: Python and Wheel
Required at runtime: Python
Test suite depends on: No test suite available
Must be installed before: Jinja2, MarkupSafe, and Meson
Optional dependencies: None

Shadow

Installation depends on: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Libcap, Libxcrypt, Make, and Sed
Required at runtime: Glibc and Libxcrypt
Test suite depends on: No test suite available
Must be installed before: Coreutils
Optional dependencies: *CrackLib* and *Linux-PAM*

Sysklogd

Installation depends on: Binutils, Coreutils, GCC, Glibc, Make, and Patch
Required at runtime: Glibc
Test suite depends on: No test suite available
Must be installed before: None
Optional dependencies: None

Sysvinit

Installation depends on: Binutils, Coreutils, GCC, Glibc, Make, and Sed
Required at runtime: Glibc
Test suite depends on: No test suite available
Must be installed before: None
Optional dependencies: None

Tar

Installation depends on: Acl, Attr, Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Sed, and Texinfo
Required at runtime: Acl, Attr, Bzip2, Glibc, Gzip, and Xz
Test suite depends on: Autoconf, Diffutils, Findutils, Gawk, and Gzip
Must be installed before: None
Optional dependencies: None

Tcl

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, and Sed
Required at runtime: Glibc and Zlib
Test suite depends on: None
Must be installed before: None
Optional dependencies: None

Texinfo

Installation depends on: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch, and Sed
Required at runtime: Glibc and Ncurses
Test suite depends on: None
Must be installed before: None
Optional dependencies: None

Udev

Installation depends on: Acl, Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Gperf, Grep, Jinja2, Libcap, Libxcrypt, Meson, OpenSSL, Pkgconf, Sed, Util-linux, and Zstd
Required at runtime: Acl, Glibc, Libcap, OpenSSL, and Util-linux
Test suite depends on: None
Must be installed before: Util-linux
Optional dependencies: None

Util-linux

Installation depends on: Bash, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Ncurses, Pkgconf, Sed, Udev, and Zlib
Required at runtime: Glibc, Ncurses, Readline, Udev, and Zlib
Test suite depends on: None
Must be installed before: None
Optional dependencies: *Asciidoctor*, *Libcap-NG*, *libeconf*, *libuser*, *libutempter*, *Linux-PAM*, *smartmontools*, *po4a*, and *slang*

Vim

Installation depends on: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, and Sed
Required at runtime: Acl, Attr, Glibc, Python, Ncurses, and Tcl
Test suite depends on: None
Must be installed before: None
Optional dependencies: *Xorg*, *GTK+2*, *LessTif*, *Ruby*, and *GPM*

Wheel

Installation depends on: Python and Flit-core
Required at runtime: Python
Test suite depends on: No test suite available
Must be installed before: Jinja2, MarkupSafe, Meson, and Setuptools
Optional dependencies: None

XML::Parser

Installation depends on: Bash, Binutils, Coreutils, Expat, GCC, Glibc, Make, and Perl
Required at runtime: Expat, Glibc, and Perl
Test suite depends on: Perl
Must be installed before: Intltool
Optional dependencies: None

Xz

Installation depends on: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, and Make
Required at runtime: Glibc
Test suite depends on: None
Must be installed before: File, GRUB, Kmod, Libelf, Man-DB, and Udev
Optional dependencies: None

Zlib

Installation depends on: Bash, Binutils, Coreutils, GCC, Glibc, Make, and Sed
Required at runtime: Glibc
Test suite depends on: None
Must be installed before: File, Kmod, Libelf, Perl, and Util-linux
Optional dependencies: None

Zstd

Installation depends on: Binutils, Coreutils, GCC, Glibc, Gzip, Make, Xz, and Zlib
Required at runtime: Glibc
Test suite depends on: None
Must be installed before: Binutils, GCC, Libelf, and Udev
Optional dependencies: *LZ4*

Appendix D. Boot and sysconfig scripts version-20230728

The scripts in this appendix are listed by the directory where they normally reside. The order is `/etc/rc.d/init.d`, `/etc/sysconfig`, `/etc/sysconfig/network-devices`, and `/etc/sysconfig/network-devices/services`. Within each section, the files are listed in the order they are normally called.

D.1. /etc/rc.d/init.d/rc

The `rc` script is the first script called by init and initiates the boot process.

```
#!/bin/bash
########################################################################
# Begin rc
#
# Description : Main Run Level Control Script
#
# Authors     : Gerard Beekmans  - gerard@linuxfromscratch.org
#             : DJ Lucas - dj@linuxfromscratch.org
# Updates     : Bruce Dubbs - bdubbs@linuxfromscratch.org
#             : Pierre Labastie - pierre@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Notes       : Updates March 24th, 2022: new semantics of S/K files
#               - Instead of testing that S scripts were K scripts in the
#                 previous runlevel, test that they were not S scripts
#               - Instead of testing that K scripts were S scripts in the
#                 previous runlevel, test that they were not K scripts
#               - S scripts in runlevel 0 or 6 are now run with
#                "script start" (was "script stop" previously).
########################################################################

. /lib/lsb/init-functions

print_error_msg()
{
   log_failure_msg
   # $i is set when called
   MSG="FAILURE:\n\nYou should not be reading this error message.\n\n"
   MSG="${MSG}It means that an unforeseen error took place in\n"
   MSG="${MSG}${i},\n"
   MSG="${MSG}which exited with a return value of ${error_value}.\n"

   MSG="${MSG}If you're able to track this error down to a bug in one of\n"
   MSG="${MSG}the files provided by the ${DISTRO_MINI} book,\n"
   MSG="${MSG}please be so kind to inform us at ${DISTRO_CONTACT}.\n"
   log_failure_msg "${MSG}"

   log_info_msg "Press Enter to continue..."
   wait_for_user
}

check_script_status()
{
   # $i is set when called
   if [ ! -f ${i} ]; then
      log_warning_msg "${i} is not a valid symlink."
      SCRIPT_STAT="1"
```

```
   fi

   if [ ! -x ${i} ]; then
      log_warning_msg "${i} is not executable, skipping."
      SCRIPT_STAT="1"
   fi
}

run()
{
   if [ -z $interactive ]; then
      ${1} ${2}
      return $?
   fi

   while true; do
      read -p "Run ${1} ${2} (Yes/no/continue)? " -n 1 runit
      echo

      case ${runit} in
         c | C)
            interactive=""
            ${i} ${2}
            ret=${?}
            break;
            ;;

         n | N)
            return 0
            ;;

         y | Y)
            ${i} ${2}
            ret=${?}
            break
            ;;
      esac
   done

   return $ret
}

# Read any local settings/overrides
[ -r /etc/sysconfig/rc.site ] && source /etc/sysconfig/rc.site

DISTRO=${DISTRO:-"Linux From Scratch"}
DISTRO_CONTACT=${DISTRO_CONTACT:-"lfs-dev@lists.linuxfromscratch.org (Registration required)"}
DISTRO_MINI=${DISTRO_MINI:-"LFS"}
IPROMPT=${IPROMPT:-"no"}

# These 3 signals will not cause our script to exit
trap "" INT QUIT TSTP

[ "${1}" != "" ] && runlevel=${1}

if [ "${runlevel}" == "" ]; then
   echo "Usage: ${0} <runlevel>" >&2
   exit 1
fi

previous=${PREVLEVEL}
[ "${previous}" == "" ] && previous=N
```

```
if [ ! -d /etc/rc.d/rc${runlevel}.d ]; then
   log_info_msg "/etc/rc.d/rc${runlevel}.d does not exist.\n"
   exit 1
fi

if [ "$runlevel" == "6" -o "$runlevel" == "0" ]; then IPROMPT="no"; fi

# Note: In ${LOGLEVEL:-7}, it is ':' 'dash' '7', not minus 7
if [ "$runlevel" == "S" ]; then
   [ -r /etc/sysconfig/console ] && source /etc/sysconfig/console
   dmesg -n "${LOGLEVEL:-7}"
fi

if [ "${IPROMPT}" == "yes" -a "${runlevel}" == "S" ]; then
   # The total length of the distro welcome string, without escape codes
   wlen=${wlen:-$(echo "Welcome to ${DISTRO}" | wc -c )}
   welcome_message=${welcome_message:-"Welcome to ${INFO}${DISTRO}${NORMAL}"}

   # The total length of the interactive string, without escape codes
   ilen=${ilen:-$(echo "Press 'I' to enter interactive startup" | wc -c )}
   i_message=${i_message:-"Press '${FAILURE}I${NORMAL}' to enter interactive startup"}


   # dcol and icol are spaces before the message to center the message
   # on screen. itime is the amount of wait time for the user to press a key
   wcol=$(( ( ${COLUMNS} - ${wlen} ) / 2 ))
   icol=$(( ( ${COLUMNS} - ${ilen} ) / 2 ))
   itime=${itime:-"3"}

   echo -e "\n\n"
   echo -e "\\033[${wcol}G${welcome_message}"
   echo -e "\\033[${icol}G${i_message}${NORMAL}"
   echo ""
   read -t "${itime}" -n 1 interactive 2>&1 > /dev/null
fi

# Make lower case
[ "${interactive}" == "I" ] && interactive="i"
[ "${interactive}" != "i" ] && interactive=""

# Read the state file if it exists from runlevel S
[ -r /run/interactive ] && source /run/interactive

# Stop all services marked as K, except if marked as K in the previous
# runlevel: it is the responsibility of the script to not try to kill
# a non running service
if [ "${previous}" != "N" ]; then
   for i in $(ls -v /etc/rc.d/rc${runlevel}.d/K* 2> /dev/null)
   do
      check_script_status
      if [ "${SCRIPT_STAT}" == "1" ]; then
         SCRIPT_STAT="0"
         continue
      fi

      suffix=${i#/etc/rc.d/rc${runlevel}.d/K[0-9][0-9]}
      [ -e /etc/rc.d/rc${previous}.d/K[0-9][0-9]$suffix ] && continue

      run ${i} stop
      error_value=${?}

      if [ "${error_value}" != "0" ]; then print_error_msg; fi
   done
```

```
fi

if [ "${previous}" == "N" ]; then export IN_BOOT=1; fi

if [ "$runlevel" == "6" -a -n "${FASTBOOT}" ]; then
   touch /fastboot
fi


# Start all services marked as S in this runlevel, except if marked as
# S in the previous runlevel
# it is the responsibility of the script to not try to start an already running
# service
for i in $( ls -v /etc/rc.d/rc${runlevel}.d/S* 2> /dev/null)
do

   if [ "${previous}" != "N" ]; then
      suffix=${i#/etc/rc.d/rc${runlevel}.d/S[0-9][0-9]}
      [ -e /etc/rc.d/rc${previous}.d/S[0-9][0-9]$suffix ] && continue
   fi

   check_script_status
   if [ "${SCRIPT_STAT}" == "1" ]; then
      SCRIPT_STAT="0"
      continue
   fi

   run ${i} start

   error_value=${?}

   if [ "${error_value}" != "0" ]; then print_error_msg; fi
done

# Store interactive variable on switch from runlevel S and remove if not
if [ "${runlevel}" == "S" -a "${interactive}" == "i" ]; then
    echo "interactive=\"i\"" > /run/interactive
else
    rm -f /run/interactive 2> /dev/null
fi

# Copy the boot log on initial boot only
if [ "${previous}" == "N" -a  "${runlevel}" != "S" ]; then
   cat $BOOTLOG >> /var/log/boot.log

   # Mark the end of boot
   echo "--------" >> /var/log/boot.log

   # Remove the temporary file
   rm -f $BOOTLOG 2> /dev/null
fi

# End rc
```

D.2. /lib/lsb/init-functions

```
#!/bin/sh
########################################################################
#
# Begin /lib/lsb/init-funtions
#
# Description : Run Level Control Functions
```

```
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#             : DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Notes       : With code based on Matthias Benkmann's simpleinit-msb
#               http://winterdrache.de/linux/newboot/index.html
#
#               The file should be located in /lib/lsb
#
########################################################################

## Environmental setup
# Setup default values for environment
umask 022
export PATH="/bin:/usr/bin:/sbin:/usr/sbin"

## Set color commands, used via echo
# Please consult `man console_codes for more information
# under the "ECMA-48 Set Graphics Rendition" section
#
# Warning: when switching from a 8bit to a 9bit font,
# the linux console will reinterpret the bold (1;) to
# the top 256 glyphs of the 9bit font.  This does
# not affect framebuffer consoles

NORMAL="\\033[0;39m"         # Standard console grey
SUCCESS="\\033[1;32m"        # Success is green
WARNING="\\033[1;33m"        # Warnings are yellow
FAILURE="\\033[1;31m"        # Failures are red
INFO="\\033[1;36m"           # Information is light cyan
BRACKET="\\033[1;34m"        # Brackets are blue

# Use a colored prefix
BMPREFIX="      "
SUCCESS_PREFIX="${SUCCESS}  *  ${NORMAL} "
FAILURE_PREFIX="${FAILURE}*****${NORMAL} "
WARNING_PREFIX="${WARNING} *** ${NORMAL} "
SKIP_PREFIX="${INFO}  S   ${NORMAL}"

SUCCESS_SUFFIX="${BRACKET}[${SUCCESS}  OK  ${BRACKET}]${NORMAL}"
FAILURE_SUFFIX="${BRACKET}[${FAILURE} FAIL ${BRACKET}]${NORMAL}"
WARNING_SUFFIX="${BRACKET}[${WARNING} WARN ${BRACKET}]${NORMAL}"
SKIP_SUFFIX="${BRACKET}[${INFO} SKIP ${BRACKET}]${NORMAL}"

BOOTLOG=/run/bootlog
KILLDELAY=3
SCRIPT_STAT="0"

# Set any user specified environment variables e.g. HEADLESS
[ -r /etc/sysconfig/rc.site ]  && . /etc/sysconfig/rc.site

## Screen Dimensions
# Find current screen size
if [ -z "${COLUMNS}" ]; then
   COLUMNS=$(stty size)
   COLUMNS=${COLUMNS##* }
fi

# When using remote connections, such as a serial port, stty size returns 0
if [ "${COLUMNS}" = "0" ]; then
```

```
   COLUMNS=80
fi

## Measurements for positioning result messages
COL=$((${COLUMNS} - 8))
WCOL=$((${COL} - 2))

## Set Cursor Position Commands, used via echo
SET_COL="\\033[${COL}G"      # at the $COL char
SET_WCOL="\\033[${WCOL}G"    # at the $WCOL char
CURS_UP="\\033[1A\\033[0G"   # Up one line, at the 0'th char
CURS_ZERO="\\033[0G"

#############################################################################
# start_daemon()                                                            #
# Usage: start_daemon [-f] [-n nicelevel] [-p pidfile] pathname [args...]   #
#                                                                           #
# Purpose: This runs the specified program as a daemon                      #
#                                                                           #
# Inputs: -f: (force) run the program even if it is already running.        #
#         -n nicelevel: specify a nice level. See 'man nice(1)'.            #
#         -p pidfile: use the specified file to determine PIDs.             #
#         pathname: the complete path to the specified program              #
#         args: additional arguments passed to the program (pathname)       #
#                                                                           #
# Return values (as defined by LSB exit codes):                             #
#       0 - program is running or service is OK                             #
#       1 - generic or unspecified error                                    #
#       2 - invalid or excessive argument(s)                                #
#       5 - program is not installed                                        #
#############################################################################
start_daemon()
{
    local force=""
    local nice="0"
    local pidfile=""
    local pidlist=""
    local retval=""

    # Process arguments
    while true
    do
        case "${1}" in

            -f)
                force="1"
                shift 1
                ;;

            -n)
                nice="${2}"
                shift 2
                ;;

            -p)
                pidfile="${2}"
                shift 2
                ;;

            -*)
                return 2
                ;;
```

```
            *)
                program="${1}"
                break
                ;;
        esac
    done

    # Check for a valid program
    if [ ! -e "${program}" ]; then return 5; fi

    # Execute
    if [ -z "${force}" ]; then
        if [ -z "${pidfile}" ]; then
            # Determine the pid by discovery
            pidlist=`pidofproc "${1}"`
            retval="${?}"
        else
            # The PID file contains the needed PIDs
            # Note that by LSB requirement, the path must be given to pidofproc,
            # however, it is not used by the current implementation or standard.
            pidlist=`pidofproc -p "${pidfile}" "${1}"`
            retval="${?}"
        fi

        # Return a value ONLY
        # It is the init script's (or distribution's functions) responsibility
        # to log messages!
        case "${retval}" in

            0)
                # Program is already running correctly, this is a
                # successful start.
                return 0
                ;;

            1)
                # Program is not running, but an invalid pid file exists
                # remove the pid file and continue
                rm -f "${pidfile}"
                ;;

            3)
                # Program is not running and no pidfile exists
                # do nothing here, let start_deamon continue.
                ;;

            *)
                # Others as returned by status values shall not be interpreted
                # and returned as an unspecified error.
                return 1
                ;;
        esac
    fi

    # Do the start!
    nice -n "${nice}" "${@}"
}

################################################################################
# killproc()                                                                   #
# Usage: killproc [-p pidfile] pathname [signal]                               #
#                                                                              #
# Purpose: Send control signals to running processes                           #
```

```
#                                                                              #
# Inputs: -p pidfile, uses the specified pidfile                               #
#         pathname, pathname to the specified program                          #
#         signal, send this signal to pathname                                 #
#                                                                              #
# Return values (as defined by LSB exit codes):                               #
#       0 - program (pathname) has stopped/is already stopped or a             #
#           running program has been sent specified signal and stopped         #
#           successfully                                                       #
#       1 - generic or unspecified error                                       #
#       2 - invalid or excessive argument(s)                                   #
#       5 - program is not installed                                           #
#       7 - program is not running and a signal was supplied                   #
################################################################################
killproc()
{
    local pidfile
    local program
    local prefix
    local progname
    local signal="-TERM"
    local fallback="-KILL"
    local nosig
    local pidlist
    local retval
    local pid
    local delay="30"
    local piddead
    local dtime

    # Process arguments
    while true; do
        case "${1}" in
            -p)
                pidfile="${2}"
                shift 2
                ;;

             *)
                 program="${1}"
                 if [ -n "${2}" ]; then
                     signal="${2}"
                     fallback=""
                 else
                     nosig=1
                 fi

                 # Error on additional arguments
                 if [ -n "${3}" ]; then
                     return 2
                 else
                     break
                 fi
                 ;;
        esac
    done

    # Check for a valid program
    if [ ! -e "${program}" ]; then return 5; fi

    # Check for a valid signal
    check_signal "${signal}"
    if [ "${?}" -ne "0" ]; then return 2; fi
```

```
    # Get a list of pids
    if [ -z "${pidfile}" ]; then
        # determine the pid by discovery
        pidlist=`pidofproc "${1}"`
        retval="${?}"
    else
        # The PID file contains the needed PIDs
        # Note that by LSB requirement, the path must be given to pidofproc,
        # however, it is not used by the current implementation or standard.
        pidlist=`pidofproc -p "${pidfile}" "${1}"`
        retval="${?}"
    fi

    # Return a value ONLY
    # It is the init script's (or distribution's functions) responsibility
    # to log messages!
    case "${retval}" in

        0)
            # Program is running correctly
            # Do nothing here, let killproc continue.
            ;;

        1)
            # Program is not running, but an invalid pid file exists
            # Remove the pid file.

            progname=${program##*/}

            if [[ -e "/run/${progname}.pid" ]]; then
                pidfile="/run/${progname}.pid"
                rm -f "${pidfile}"
            fi

            # This is only a success if no signal was passed.
            if [ -n "${nosig}" ]; then
                return 0
            else
                return 7
            fi
            ;;

        3)
            # Program is not running and no pidfile exists
            # This is only a success if no signal was passed.
            if [ -n "${nosig}" ]; then
                return 0
            else
                return 7
            fi
            ;;

        *)
            # Others as returned by status values shall not be interpreted
            # and returned as an unspecified error.
            return 1
            ;;
    esac

    # Perform different actions for exit signals and control signals
    check_sig_type "${signal}"
```

```
    if [ "${?}" -eq "0" ]; then # Signal is used to terminate the program

        # Account for empty pidlist (pid file still exists and no
        # signal was given)
        if [ "${pidlist}" != "" ]; then

            # Kill the list of pids
            for pid in ${pidlist}; do

                kill -0 "${pid}" 2> /dev/null

                if [ "${?}" -ne "0" ]; then
                    # Process is dead, continue to next and assume all is well
                    continue
                else
                    kill "${signal}" "${pid}" 2> /dev/null

                    # Wait up to ${delay}/10 seconds to for "${pid}" to
                    # terminate in 10ths of a second

                    while [ "${delay}" -ne "0" ]; do
                        kill -0 "${pid}" 2> /dev/null || piddead="1"
                        if [ "${piddead}" = "1" ]; then break; fi
                        sleep 0.1
                        delay="$(( ${delay} - 1 ))"
                    done

                    # If a fallback is set, and program is still running, then
                    # use the fallback
                    if [ -n "${fallback}" -a "${piddead}" != "1" ]; then
                        kill "${fallback}" "${pid}" 2> /dev/null
                        sleep 1
                        # Check again, and fail if still running
                        kill -0 "${pid}" 2> /dev/null && return 1
                    fi
                fi
            done
        fi

        # Check for and remove stale PID files.
        if [ -z "${pidfile}" ]; then
            # Find the basename of $program
            prefix=`echo "${program}" | sed 's/[^/]*$//'`
            progname=`echo "${program}" | sed "s@${prefix}@@"`

            if [ -e "/run/${progname}.pid" ]; then
                rm -f "/run/${progname}.pid" 2> /dev/null
            fi
        else
            if [ -e "${pidfile}" ]; then rm -f "${pidfile}" 2> /dev/null; fi
        fi

    # For signals that do not expect a program to exit, simply
    # let kill do its job, and evaluate kill's return for value

    else # check_sig_type - signal is not used to terminate program
        for pid in ${pidlist}; do
            kill "${signal}" "${pid}"
            if [ "${?}" -ne "0" ]; then return 1; fi
        done
    fi
}
```

```
##############################################################################
# pidofproc()                                                                #
# Usage: pidofproc [-p pidfile] pathname                                     #
#                                                                            #
# Purpose: This function returns one or more pid(s) for a particular daemon  #
#                                                                            #
# Inputs: -p pidfile, use the specified pidfile instead of pidof             #
#         pathname, path to the specified program                            #
#                                                                            #
# Return values (as defined by LSB status codes):                            #
#       0 - Success (PIDs to stdout)                                         #
#       1 - Program is dead, PID file still exists (remaining PIDs output)   #
#       3 - Program is not running (no output)                               #
##############################################################################
pidofproc()
{
    local pidfile
    local program
    local prefix
    local progname
    local pidlist
    local lpids
    local exitstatus="0"

    # Process arguments
    while true; do
        case "${1}" in

            -p)
                pidfile="${2}"
                shift 2
                ;;

            *)
                program="${1}"
                if [ -n "${2}" ]; then
                    # Too many arguments
                    # Since this is status, return unknown
                    return 4
                else
                    break
                fi
                ;;
        esac
    done

    # If a PID file is not specified, try and find one.
    if [ -z "${pidfile}" ]; then
        # Get the program's basename
        prefix=`echo "${program}" | sed 's/[^/]*$//'`

        if [ -z "${prefix}" ]; then
           progname="${program}"
        else
           progname=`echo "${program}" | sed "s@${prefix}@@"`
        fi

        # If a PID file exists with that name, assume that is it.
        if [ -e "/run/${progname}.pid" ]; then
            pidfile="/run/${progname}.pid"
        fi
    fi
```

```
    # If a PID file is set and exists, use it.
    if [ -n "${pidfile}" -a -e "${pidfile}" ]; then

        # Use the value in the first line of the pidfile
        pidlist=`/bin/head -n1 "${pidfile}"`
        # This can optionally be written as 'sed 1q' to repalce 'head -n1'
        # should LFS move /bin/head to /usr/bin/head
    else
        # Use pidof
        pidlist=`pidof "${program}"`
    fi

    # Figure out if all listed PIDs are running.
    for pid in ${pidlist}; do
        kill -0 ${pid} 2> /dev/null

        if [ "${?}" -eq "0" ]; then
            lpids="${lpids}${pid} "
        else
            exitstatus="1"
        fi
    done

    if [ -z "${lpids}" -a ! -f "${pidfile}" ]; then
        return 3
    else
        echo "${lpids}"
        return "${exitstatus}"
    fi
}

################################################################################
# statusproc()                                                                 #
# Usage: statusproc [-p pidfile] pathname                                      #
#                                                                              #
# Purpose: This function prints the status of a particular daemon to stdout    #
#                                                                              #
# Inputs: -p pidfile, use the specified pidfile instead of pidof               #
#         pathname, path to the specified program                              #
#                                                                              #
# Return values:                                                               #
#       0 - Status printed                                                     #
#       1 - Input error. The daemon to check was not specified.                #
################################################################################
statusproc()
{
   local pidfile
   local pidlist

   if [ "${#}" = "0" ]; then
      echo "Usage: statusproc [-p pidfle] {program}"
      exit 1
   fi

   # Process arguments
   while true; do
       case "${1}" in

           -p)
               pidfile="${2}"
               shift 2
               ;;
```

```
           *)
               if [ -n "${2}" ]; then
                   echo "Too many arguments"
                   return 1
               else
                   break
               fi
               ;;
       esac
   done

   if [ -n "${pidfile}" ]; then
      pidlist=`pidofproc -p "${pidfile}" $@`
   else
      pidlist=`pidofproc $@`
   fi

   # Trim trailing blanks
   pidlist=`echo "${pidlist}" | sed -r 's/ +$//'`

   base="${1##*/}"

   if [ -n "${pidlist}" ]; then
      /bin/echo -e "${INFO}${base} is running with Process" \
         "ID(s) ${pidlist}.${NORMAL}"
   else
      if [ -n "${base}" -a -e "/run/${base}.pid" ]; then
         /bin/echo -e "${WARNING}${1} is not running but" \
            "/run/${base}.pid exists.${NORMAL}"
      else
         if [ -n "${pidfile}" -a -e "${pidfile}" ]; then
            /bin/echo -e "${WARNING}${1} is not running" \
               "but ${pidfile} exists.${NORMAL}"
         else
            /bin/echo -e "${INFO}${1} is not running.${NORMAL}"
         fi
      fi
   fi
}

##############################################################################
# timespec()                                                                 #
#                                                                            #
# Purpose: An internal utility function to format a timestamp                #
#          a boot log file.  Sets the STAMP variable.                        #
#                                                                            #
# Return value: Not used                                                     #
##############################################################################
timespec()
{
   STAMP="$(echo `date +"%b %d %T %:z"` `hostname`) "
   return 0
}

##############################################################################
# log_success_msg()                                                          #
# Usage: log_success_msg ["message"]                                         #
#                                                                            #
# Purpose: Print a successful status message to the screen and               #
#          a boot log file.                                                  #
#                                                                            #
# Inputs: $@ - Message                                                       #
#                                                                            #
```

```
# Return values: Not used                                                      #
################################################################################
log_success_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${SUCCESS_PREFIX}${SET_COL}${SUCCESS_SUFFIX}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\\033[^a-zA-Z]*.//g'`

    timespec
    /bin/echo -e "${STAMP} ${logmessage} OK" >> ${BOOTLOG}

    return 0
}

log_success_msg2()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${SUCCESS_PREFIX}${SET_COL}${SUCCESS_SUFFIX}"

    echo " OK" >> ${BOOTLOG}

    return 0
}

################################################################################
# log_failure_msg()                                                            #
# Usage: log_failure_msg ["message"]                                           #
#                                                                              #
# Purpose: Print a failure status message to the screen and                    #
#          a boot log file.                                                    #
#                                                                              #
# Inputs: $@ - Message                                                         #
#                                                                              #
# Return values: Not used                                                      #
################################################################################
log_failure_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${FAILURE_PREFIX}${SET_COL}${FAILURE_SUFFIX}"

    # Strip non-printable characters from log file

    timespec
    logmessage=`echo "${@}" | sed 's/\\\033[^a-zA-Z]*.//g'`
    /bin/echo -e "${STAMP} ${logmessage} FAIL" >> ${BOOTLOG}

    return 0
}

log_failure_msg2()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${FAILURE_PREFIX}${SET_COL}${FAILURE_SUFFIX}"

    echo "FAIL" >> ${BOOTLOG}

    return 0
}

################################################################################
# log_warning_msg()                                                            #
```

```
# Usage: log_warning_msg ["message"]                                          #
#                                                                             #
# Purpose: Print a warning status message to the screen and                   #
#          a boot log file.                                                   #
#                                                                             #
# Return values: Not used                                                     #
###############################################################################
log_warning_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${WARNING_PREFIX}${SET_COL}${WARNING_SUFFIX}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\\033[^a-zA-Z]*.//g'`
    timespec
    /bin/echo -e "${STAMP} ${logmessage} WARN" >> ${BOOTLOG}

    return 0
}

log_skip_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"
    /bin/echo -e "${CURS_ZERO}${SKIP_PREFIX}${SET_COL}${SKIP_SUFFIX}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\\033[^a-zA-Z]*.//g'`
    /bin/echo "SKIP" >> ${BOOTLOG}

    return 0
}

###############################################################################
# log_info_msg()                                                              #
# Usage: log_info_msg message                                                 #
#                                                                             #
# Purpose: Print an information message to the screen and                     #
#          a boot log file.  Does not print a trailing newline character.     #
#                                                                             #
# Return values: Not used                                                     #
###############################################################################
log_info_msg()
{
    /bin/echo -n -e "${BMPREFIX}${@}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\\033[^a-zA-Z]*.//g'`
    timespec
    /bin/echo -n -e "${STAMP} ${logmessage}" >> ${BOOTLOG}

    return 0
}

log_info_msg2()
{
    /bin/echo -n -e "${@}"

    # Strip non-printable characters from log file
    logmessage=`echo "${@}" | sed 's/\\\033[^a-zA-Z]*.//g'`
    /bin/echo -n -e "${logmessage}" >> ${BOOTLOG}

    return 0
}
```

```
########################################################################################
# evaluate_retval()                                                                    #
# Usage: Evaluate a return value and print success or failure as appropriate  #
#                                                                                      #
# Purpose: Convenience function to terminate an info message                           #
#                                                                                      #
# Return values: Not used                                                              #
########################################################################################
evaluate_retval()
{
   local error_value="${?}"

   if [ ${error_value} = 0 ]; then
      log_success_msg2
   else
      log_failure_msg2
   fi
}

########################################################################################
# check_signal()                                                                       #
# Usage: check_signal [ -{signal} ]                                                    #
#                                                                                      #
# Purpose: Check for a valid signal.  This is not defined by any LSB draft,    #
#          however, it is required to check the signals to determine if the    #
#          signals chosen are invalid arguments to the other functions.        #
#                                                                                      #
# Inputs: Accepts a single string value in the form of -{signal}               #
#                                                                                      #
# Return values:                                                                       #
#       0 - Success (signal is valid                                                   #
#       1 - Signal is not valid                                                        #
########################################################################################
check_signal()
{
    local valsig

    # Add error handling for invalid signals
    valsig=" -ALRM -HUP -INT -KILL -PIPE -POLL -PROF -TERM -USR1 -USR2"
    valsig="${valsig} -VTALRM -STKFLT -PWR -WINCH -CHLD -URG -TSTP -TTIN"
    valsig="${valsig} -TTOU -STOP -CONT -ABRT -FPE -ILL -QUIT -SEGV -TRAP"
    valsig="${valsig} -SYS -EMT -BUS -XCPU -XFSZ -0 -1 -2 -3 -4 -5 -6 -8 -9"
    valsig="${valsig} -11 -13 -14 -15 "

    echo "${valsig}" | grep -- " ${1} " > /dev/null

    if [ "${?}" -eq "0" ]; then
        return 0
    else
        return 1
    fi
}

########################################################################################
# check_sig_type()                                                                     #
# Usage: check_signal [ -{signal} | {signal} ]                                         #
#                                                                                      #
# Purpose: Check if signal is a program termination signal or a control signal #
#          This is not defined by any LSB draft, however, it is required to    #
#          check the signals to determine if they are intended to end a        #
#          program or simply to control it.                                    #
#                                                                                      #
```

```
# Inputs: Accepts a single string value in the form or -{signal} or {signal}   #
#                                                                              #
# Return values:                                                               #
#       0 - Signal is used for program termination                             #
#       1 - Signal is used for program control                                 #
################################################################################
check_sig_type()
{
    local valsig

    # The list of termination signals (limited to generally used items)
    valsig=" -ALRM -INT -KILL -TERM -PWR -STOP -ABRT -QUIT -2 -3 -6 -9 -14 -15 "

    echo "${valsig}" | grep -- " ${1} " > /dev/null

    if [ "${?}" -eq "0" ]; then
        return 0
    else
        return 1
    fi
}

################################################################################
# wait_for_user()                                                              #
#                                                                              #
# Purpose: Wait for the user to respond if not a headless system               #
#                                                                              #
################################################################################
wait_for_user()
{
   # Wait for the user by default
   [ "${HEADLESS=0}" = "0" ] && read ENTER
   return 0
}

################################################################################
# is_true()                                                                    #
#                                                                              #
# Purpose: Utility to test if a variable is true | yes | 1                     #
#                                                                              #
################################################################################
is_true()
{
   [ "$1" = "1" ] || [ "$1" = "yes" ] || [ "$1" = "true" ] ||  [ "$1" = "y" ] ||
   [ "$1" = "t" ]
}

# End /lib/lsb/init-functions
```

D.3. /etc/rc.d/init.d/mountvirtfs

```
#!/bin/sh
########################################################################
# Begin mountvirtfs
#
# Description : Ensure proc, sysfs, run, and dev are mounted
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#               Xi Ruoyao - xry111@xry111.site
#
```

```
# Version     : LFS 12.0
#
###############################################################################

### BEGIN INIT INFO
# Provides:            mountvirtfs
# Required-Start:      $first
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Mounts various special fs needed at start
# Description:         Mounts /sys and /proc virtual (kernel) filesystems.
#                      Mounts /run (tmpfs) and /dev (devtmpfs).
#                      This is done only if they are not already mounted.
#                      with the kernel config proposed in the book, dev
#                      should be automatically mounted by the kernel.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      # Make sure /run is available before logging any messages
      if ! mountpoint /run >/dev/null; then
         mount /run || failed=1
      fi

      mkdir -p   /run/lock
      chmod 1777 /run/lock

      log_info_msg "Mounting virtual file systems: ${INFO}/run"

      if ! mountpoint /proc >/dev/null; then
         log_info_msg2 " ${INFO}/proc"
         mount -o nosuid,noexec,nodev /proc || failed=1
      fi

      if ! mountpoint /sys >/dev/null; then
         log_info_msg2 " ${INFO}/sys"
         mount -o nosuid,noexec,nodev /sys || failed=1
      fi

      if ! mountpoint /dev >/dev/null; then
         log_info_msg2 " ${INFO}/dev"
         mount -o mode=0755,nosuid /dev  || failed=1
      fi

      mkdir -p /dev/shm
      log_info_msg2 " ${INFO}/dev/shm"
      mount -o nosuid,nodev /dev/shm || failed=1

      mkdir -p /sys/fs/cgroup
      log_info_msg2 " ${INFO}/sys/fs/cgroup"
      mount -o nosuid,noexec,nodev /sys/fs/cgroup || failed=1

      (exit ${failed})
      evaluate_retval
      if [ "${failed}" = 1 ]; then
         exit 1
      fi
```

```
      log_info_msg "Create symlinks in /dev targeting /proc: ${INFO}/dev/stdin"
      ln -sf /proc/self/fd/0 /dev/stdin  || failed=1

      log_info_msg2 " ${INFO}/dev/stdout"
      ln -sf /proc/self/fd/1 /dev/stdout || failed=1

      log_info_msg2 " ${INFO}/dev/stderr"
      ln -sf /proc/self/fd/2 /dev/stderr || failed=1

      log_info_msg2 " ${INFO}/dev/fd"
      ln -sf /proc/self/fd   /dev/fd     || failed=1

      if [ -e /proc/kcore ]; then
         log_info_msg2 " ${INFO}/dev/core"
         ln -sf /proc/kcore  /dev/core   || failed=1
      fi

      (exit ${failed})
      evaluate_retval
      exit $failed
      ;;

   *)
      echo "Usage: ${0} {start}"
      exit 1
      ;;
esac

# End mountvirtfs
```

D.4. /etc/rc.d/init.d/modules

```
#!/bin/sh
########################################################################
# Begin modules
#
# Description : Module auto-loading script
#
# Authors     : Zack Winkles
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            modules
# Required-Start:      mountvirtfs
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Loads required modules.
# Description:         Loads modules listed in /etc/sysconfig/modules.
# X-LFS-Provided-By:   LFS
### END INIT INFO

# Assure that the kernel has module support.
[ -e /proc/modules ] || exit 0
```

```
. /lib/lsb/init-functions

case "${1}" in
   start)
      # Exit if there's no modules file or there are no
      # valid entries
      [ -r /etc/sysconfig/modules ]             || exit 0
      grep -E -qv '^($|#)' /etc/sysconfig/modules || exit 0

      log_info_msg "Loading modules:"

      # Only try to load modules if the user has actually given us
      # some modules to load.

      while read module args; do

         # Ignore comments and blank lines.
         case "$module" in
            ""|"#"*) continue ;;
         esac

         # Attempt to load the module, passing any arguments provided.
         modprobe ${module} ${args} >/dev/null

         # Print the module name if successful, otherwise take note.
         if [ $? -eq 0 ]; then
            log_info_msg2 " ${module}"
         else
            failedmod="${failedmod} ${module}"
         fi
      done < /etc/sysconfig/modules

      # Print a message about successfully loaded modules on the correct line.
      log_success_msg2

      # Print a failure message with a list of any modules that
      # may have failed to load.
      if [ -n "${failedmod}" ]; then
         log_failure_msg "Failed to load modules:${failedmod}"
         exit 1
      fi
      ;;

   *)
      echo "Usage: ${0} {start}"
      exit 1
      ;;
esac

exit 0

# End modules
```

D.5. /etc/rc.d/init.d/udev

```
#!/bin/sh
########################################################################
# Begin udev
#
# Description : Udev cold-plugging script
#
```

```
# Authors     : Zack Winkles, Alexander E. Patrakov
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#               Xi Ruoyao - xry111@xry111.site
#
# Version     : LFS 12.0
#
###############################################################################

### BEGIN INIT INFO
# Provides:            udev $time
# Required-Start:      localnet
# Should-Start:        modules
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Populates /dev with device nodes.
# Description:         Mounts a tempfs on /dev and starts the udevd daemon.
#                      Device nodes are created as defined by udev.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      log_info_msg "Populating /dev with device nodes... "
      if ! grep -q '[[:space:]]sysfs' /proc/mounts; then
         log_failure_msg2
         msg="FAILURE:\n\nUnable to create "
         msg="${msg}devices without a SysFS filesystem\n\n"
         msg="${msg}After you press Enter, this system "
         msg="${msg}will be halted and powered off.\n\n"
         log_info_msg "$msg"
         log_info_msg "Press Enter to continue..."
         wait_for_user
         /etc/rc.d/init.d/halt start
      fi

      # Start the udev daemon to continually watch for, and act on,
      # uevents
      SYSTEMD_LOG_TARGET=kmsg /sbin/udevd --daemon

      # Now traverse /sys in order to "coldplug" devices that have
      # already been discovered
      /bin/udevadm trigger --action=add    --type=subsystems
      /bin/udevadm trigger --action=add    --type=devices
      /bin/udevadm trigger --action=change --type=devices

      # Now wait for udevd to process the uevents we triggered
      if ! is_true "$OMIT_UDEV_SETTLE"; then
         /bin/udevadm settle
      fi

      # If any LVM based partitions are on the system, ensure they
      # are activated so they can be used.
      if [ -x /sbin/vgchange ]; then /sbin/vgchange -a y >/dev/null; fi

      log_success_msg2
      ;;

   *)
```

```
      echo "Usage ${0} {start}"
      exit 1
      ;;
esac

exit 0

# End udev
```

D.6. /etc/rc.d/init.d/swap

```
#!/bin/sh
########################################################################
# Begin swap
#
# Description : Swap Control Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            swap
# Required-Start:      udev
# Should-Start:        modules
# Required-Stop:       localnet
# Should-Stop:         $local_fs
# Default-Start:       S
# Default-Stop:        0 6
# Short-Description:   Activates and deactivates swap partitions.
# Description:         Activates and deactivates swap partitions defined in
#                      /etc/fstab.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      log_info_msg "Activating all swap files/partitions..."
      swapon -a
      evaluate_retval
      ;;

   stop)
      log_info_msg "Deactivating all swap files/partitions..."
      swapoff -a
      evaluate_retval
      ;;

   restart)
      ${0} stop
      sleep 1
      ${0} start
      ;;

   status)
      log_success_msg "Retrieving swap status."
```

```
      swapon -s
      ;;

   *)
      echo "Usage: ${0} {start|stop|restart|status}"
      exit 1
      ;;
esac

exit 0

# End swap
```

D.7. /etc/rc.d/init.d/setclock

```
#!/bin/sh
########################################################################
# Begin setclock
#
# Description : Setting Linux Clock
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:
# Required-Start:
# Should-Start:        modules
# Required-Stop:
# Should-Stop:         $syslog
# Default-Start:       S
# Default-Stop:
# Short-Description:   Stores and restores time from the hardware clock
# Description:         On boot, system time is obtained from hwclock.  The
#                      hardware clock can also be set on shutdown.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

[ -r /etc/sysconfig/clock ] && . /etc/sysconfig/clock


case "${UTC}" in
   yes|true|1)
      CLOCKPARAMS="${CLOCKPARAMS} --utc"
      ;;

   no|false|0)
      CLOCKPARAMS="${CLOCKPARAMS} --localtime"
      ;;

esac

case ${1} in
   start)
      hwclock --hctosys ${CLOCKPARAMS} >/dev/null
```

```
      ;;

   stop)
      log_info_msg "Setting hardware clock..."
      hwclock --systohc ${CLOCKPARAMS} >/dev/null
      evaluate_retval
      ;;

   *)
      echo "Usage: ${0} {start|stop}"
      exit 1
      ;;

esac

exit 0
```

D.8. /etc/rc.d/init.d/checkfs

```
#!/bin/sh
########################################################################
# Begin checkfs
#
# Description : File System Check
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               A. Luebke - luebke@users.sourceforge.net
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Based on checkfs script from LFS-3.1 and earlier.
#
# From man fsck
# 0    - No errors
# 1    - File system errors corrected
# 2    - System should be rebooted
# 4    - File system errors left uncorrected
# 8    - Operational error
# 16   - Usage or syntax error
# 32   - Fsck canceled by user request
# 128  - Shared library error
#
########################################################################

### BEGIN INIT INFO
# Provides:            checkfs
# Required-Start:      udev swap
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Checks local filesystems before mounting.
# Description:         Checks local filesystems before mounting.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
```

```
   start)
      if [ -f /fastboot ]; then
         msg="/fastboot found, will omit "
         msg="${msg} file system checks as requested.\n"
         log_info_msg "${msg}"
         exit 0
      fi

      log_info_msg "Mounting root file system in read-only mode... "
      mount -n -o remount,ro / >/dev/null

      if [ ${?} != 0 ]; then
         log_failure_msg2
         msg="\n\nCannot check root "
         msg="${msg}filesystem because it could not be mounted "
         msg="${msg}in read-only mode.\n\n"
         msg="${msg}After you press Enter, this system will be "
         msg="${msg}halted and powered off.\n\n"
         log_failure_msg "${msg}"

         log_info_msg "Press Enter to continue..."
         wait_for_user
         /etc/rc.d/init.d/halt start
      else
         log_success_msg2
      fi

      if [ -f /forcefsck ]; then
         msg="/forcefsck found, forcing file"
         msg="${msg} system checks as requested."
         log_success_msg "$msg"
         options="-f"
      else
         options=""
      fi

      log_info_msg "Checking file systems..."
      # Note: -a option used to be -p; but this fails e.g. on fsck.minix
      if is_true "$VERBOSE_FSCK"; then
        fsck ${options} -a -A -C -T
      else
        fsck ${options} -a -A -C -T >/dev/null
      fi

      error_value=${?}

      if [ "${error_value}" = 0 ]; then
         log_success_msg2
      fi

      if [ "${error_value}" = 1 ]; then
         msg="\nWARNING:\n\nFile system errors "
         msg="${msg}were found and have been corrected.\n"
         msg="${msg}      You may want to double-check that "
         msg="${msg}everything was fixed properly."
         log_warning_msg "$msg"
      fi

      if [ "${error_value}" = 2 -o "${error_value}" = 3 ]; then
         msg="\nWARNING:\n\nFile system errors "
         msg="${msg}were found and have been "
         msg="${msg}corrected, but the nature of the "
         msg="${msg}errors require this system to be rebooted.\n\n"
```

```
         msg="${msg}After you press enter, "
         msg="${msg}this system will be rebooted\n\n"
         log_failure_msg "$msg"

         log_info_msg "Press Enter to continue..."
         wait_for_user
         reboot -f
      fi

      if [ "${error_value}" -gt 3 -a "${error_value}" -lt 16 ]; then
         msg="\nFAILURE:\n\nFile system errors "
         msg="${msg}were encountered that could not be "
         msg="${msg}fixed automatically.\nThis system "
         msg="${msg}cannot continue to boot and will "
         msg="${msg}therefore be halted until those "
         msg="${msg}errors are fixed manually by a "
         msg="${msg}System Administrator.\n\n"
         msg="${msg}After you press Enter, this system will be "
         msg="${msg}halted and powered off.\n\n"
         log_failure_msg "$msg"

         log_info_msg "Press Enter to continue..."
         wait_for_user
         /etc/rc.d/init.d/halt start
      fi

      if [ "${error_value}" -ge 16 ]; then
         msg="FAILURE:\n\nUnexpected failure "
         msg="${msg}running fsck.  Exited with error "
         msg="${msg} code: ${error_value}.\n"
         log_info_msg $msg
         exit ${error_value}
      fi

      exit 0
      ;;
   *)
      echo "Usage: ${0} {start}"
      exit 1
      ;;
esac

# End checkfs
```

D.9. /etc/rc.d/init.d/mountfs

```
#!/bin/sh
########################################################################
# Begin mountfs
#
# Description : File System Mount Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            $local_fs
```

```
# Required-Start:      udev checkfs
# Should-Start:        modules
# Required-Stop:       localnet
# Should-Stop:
# Default-Start:       S
# Default-Stop:        0 6
# Short-Description:   Mounts/unmounts local filesystems defined in /etc/fstab.
# Description:         Remounts root filesystem read/write and mounts all
#                      remaining local filesystems defined in /etc/fstab on
#                      start.  Remounts root filesystem read-only and unmounts
#                      remaining filesystems on stop.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      log_info_msg "Remounting root file system in read-write mode..."
      mount --options remount,rw / >/dev/null
      evaluate_retval

      # Remove fsck-related file system watermarks.
      rm -f /fastboot /forcefsck

      # Make sure /dev/pts exists
      mkdir -p /dev/pts

      # This will mount all filesystems that do not have _netdev in
      # their option list.  _netdev denotes a network filesystem.

      log_info_msg "Mounting remaining file systems..."
      failed=0
      mount --all --test-opts no_netdev >/dev/null || failed=1
      evaluate_retval
      exit $failed
      ;;

   stop)
      # Don't unmount virtual file systems like /run
      log_info_msg "Unmounting all other currently mounted file systems..."
      # Ensure any loop devices are removed
      losetup -D
      umount --all --detach-loop --read-only \
             --types notmpfs,nosysfs,nodevtmpfs,noproc,nodevpts >/dev/null
      evaluate_retval

      # Make sure / is mounted read only (umount bug)
      mount --options remount,ro /

      # Make all LVM volume groups unavailable, if appropriate
      # This fails if swap or / are on an LVM partition
      #if [ -x /sbin/vgchange ]; then /sbin/vgchange -an > /dev/null; fi
      if [ -r /etc/mdadm.conf ]; then
         log_info_msg "Mark arrays as clean..."
         mdadm --wait-clean --scan
         evaluate_retval
      fi
      ;;

   *)
      echo "Usage: ${0} {start|stop}"
      exit 1
```

```
      ;;
esac

# End mountfs
```

D.10. /etc/rc.d/init.d/udev_retry

```
#!/bin/sh
########################################################################
# Begin udev_retry
#
# Description : Udev cold-plugging script (retry)
#
# Authors     : Alexander E. Patrakov
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#               Bryan Kadzban -
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            udev_retry
# Required-Start:      udev
# Should-Start:        $local_fs cleanfs
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Replays failed uevents and creates additional devices.
# Description:         Replays any failed uevents that were skipped due to
#                      slow hardware initialization, and creates those needed
#                      device nodes
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      log_info_msg "Retrying failed uevents, if any..."

      rundir=/run/udev
      # From Debian: "copy the rules generated before / was mounted
      # read-write":

      for file in ${rundir}/tmp-rules--*; do
         dest=${file##*tmp-rules--}
         [ "$dest" = '*' ] && break
         cat $file >> /etc/udev/rules.d/$dest
         rm -f $file
      done

      # Re-trigger the uevents that may have failed,
      # in hope they will succeed now
      /bin/sed -e 's/#.*$//' /etc/sysconfig/udev_retry | /bin/grep -v '^$' | \
      while read line ; do
         for subsystem in $line ; do
            /bin/udevadm trigger --subsystem-match=$subsystem --action=add
         done
      done
```

```
      # Now wait for udevd to process the uevents we triggered
      if ! is_true "$OMIT_UDEV_RETRY_SETTLE"; then
         /bin/udevadm settle
      fi

      evaluate_retval
      ;;

   *)
      echo "Usage ${0} {start}"
      exit 1
      ;;
esac

exit 0

# End udev_retry
```

D.11. /etc/rc.d/init.d/cleanfs

```
#!/bin/sh
########################################################################
# Begin cleanfs
#
# Description : Clean file system
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            cleanfs
# Required-Start:      $local_fs
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Cleans temporary directories early in the boot process.
# Description:         Cleans temporary directories /run, /var/lock, and
#                      optionally, /tmp.  cleanfs also creates /run/utmp
#                      and any files defined in /etc/sysconfig/createfiles.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

# Function to create files/directory on boot.
create_files()
{
   # Input to file descriptor 9 and output to stdin (redirection)
   exec 9>&0 < /etc/sysconfig/createfiles

   while read name type perm usr grp dtype maj min junk
   do
      # Ignore comments and blank lines.
      case "${name}" in
```

```
         ""|\#*) continue ;;
      esac

      # Ignore existing files.
      if [ ! -e "${name}" ]; then
         # Create stuff based on its type.
         case "${type}" in
            dir)
               mkdir "${name}"
               ;;
            file)
               :> "${name}"
               ;;
            dev)
               case "${dtype}" in
                  char)
                     mknod "${name}" c ${maj} ${min}
                     ;;
                  block)
                     mknod "${name}" b ${maj} ${min}
                     ;;
                  pipe)
                     mknod "${name}" p
                     ;;
                  *)
                     log_warning_msg "\nUnknown device type: ${dtype}"
                     ;;
               esac
               ;;
            *)
               log_warning_msg "\nUnknown type: ${type}"
               continue
               ;;
         esac

         # Set up the permissions, too.
         chown ${usr}:${grp} "${name}"
         chmod ${perm} "${name}"
      fi
   done

   # Close file descriptor 9 (end redirection)
   exec 0>&9 9>&-
   return 0
}

case "${1}" in
   start)
      log_info_msg "Cleaning file systems:"

      if [ "${SKIPTMPCLEAN}" = "" ]; then
         log_info_msg2 " /tmp"
         cd /tmp &&
         find . -xdev -mindepth 1 ! -name lost+found -delete || failed=1
      fi

      > /run/utmp

      if grep -q '^utmp:' /etc/group ; then
         chmod 664 /run/utmp
         chgrp utmp /run/utmp
      fi
```

```
      (exit ${failed})
      evaluate_retval

      if grep -E -qv '^(#|$)' /etc/sysconfig/createfiles 2>/dev/null; then
         log_info_msg "Creating files and directories... "
         create_files      # Always returns 0
         evaluate_retval
      fi

      exit $failed
      ;;
   *)
      echo "Usage: ${0} {start}"
      exit 1
      ;;
esac

# End cleanfs
```

D.12. /etc/rc.d/init.d/console

```
#!/bin/sh
########################################################################
# Begin console
#
# Description : Sets keymap and screen font
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               Alexander E. Patrakov
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            console
# Required-Start:      $local_fs
# Should-Start:        udev_retry
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Sets up a localised console.
# Description:         Sets up fonts and language settings for the user's
#                      local as defined by /etc/sysconfig/console.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

# Native English speakers probably don't have /etc/sysconfig/console at all
[ -r /etc/sysconfig/console ] && . /etc/sysconfig/console

failed=0

case "${1}" in
   start)
      # See if we need to do anything
      if [ -z "${KEYMAP}"         ] && [ -z "${KEYMAP_CORRECTIONS}" ] &&
         [ -z "${FONT}"           ] && [ -z "${LEGACY_CHARSET}"     ] &&
```

```
        ! is_true "${UNICODE}"; then
        exit 0
    fi

    # There should be no bogus failures below this line!
    log_info_msg "Setting up Linux console..."

    # Figure out if a framebuffer console is used
    [ -d /sys/class/graphics/fb0 ] && use_fb=1 || use_fb=0

    # Figure out the command to set the console into the
    # desired mode
    is_true "${UNICODE}" &&
        MODE_COMMAND="echo -en '\033%G' && kbd_mode -u" ||
        MODE_COMMAND="echo -en '\033%@\033(K' && kbd_mode -a"

    # On framebuffer consoles, font has to be set for each vt in
    # UTF-8 mode. This doesn't hurt in non-UTF-8 mode also.

    ! is_true "${use_fb}" || [ -z "${FONT}" ] ||
        MODE_COMMAND="${MODE_COMMAND} && setfont ${FONT}"

    # Apply that command to all consoles mentioned in
    # /etc/inittab. Important: in the UTF-8 mode this should
    # happen before setfont, otherwise a kernel bug will
    # show up and the unicode map of the font will not be
    # used.

    for TTY in `grep '^[^#].*respawn:/sbin/agetty' /etc/inittab |
        grep -o '\btty[[:digit:]]*\b'`
    do
        openvt -f -w -c ${TTY#tty} -- \
            /bin/sh -c "${MODE_COMMAND}" || failed=1
    done

    # Set the font (if not already set above) and the keymap
    [ "${use_fb}" == "1" ] || [ -z "${FONT}" ] || setfont $FONT || failed=1

    [ -z "${KEYMAP}" ] ||
        loadkeys ${KEYMAP} >/dev/null 2>&1 ||
        failed=1

    [ -z "${KEYMAP_CORRECTIONS}" ] ||
        loadkeys ${KEYMAP_CORRECTIONS} >/dev/null 2>&1 ||
        failed=1

    # Convert the keymap from $LEGACY_CHARSET to UTF-8
    [ -z "$LEGACY_CHARSET" ] ||
        dumpkeys -c "$LEGACY_CHARSET" | loadkeys -u >/dev/null 2>&1 ||
        failed=1

    # If any of the commands above failed, the trap at the
    # top would set $failed to 1
    ( exit $failed )
    evaluate_retval

    exit $failed
    ;;

*)
    echo "Usage:  ${0} {start}"
    exit 1
    ;;
```

```
esac

# End console
```

D.13. /etc/rc.d/init.d/localnet

```
#!/bin/sh
########################################################################
# Begin localnet
#
# Description : Loopback device
#
# Authors     : Gerard Beekmans  - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            localnet
# Required-Start:      mountvirtfs
# Should-Start:        modules
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:        0 6
# Short-Description:   Starts the local network.
# Description:         Sets the hostname of the machine and starts the
#                      loopback interface.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions
[ -r /etc/sysconfig/network ] && . /etc/sysconfig/network
[ -r /etc/hostname ] && HOSTNAME=`cat /etc/hostname`

case "${1}" in
   start)
      log_info_msg "Bringing up the loopback interface..."
      ip addr add 127.0.0.1/8 label lo dev lo
      ip link set lo up
      evaluate_retval

      log_info_msg "Setting hostname to ${HOSTNAME}..."
      hostname ${HOSTNAME}
      evaluate_retval
      ;;

   stop)
      log_info_msg "Bringing down the loopback interface..."
      ip link set lo down
      evaluate_retval
      ;;

   restart)
      ${0} stop
      sleep 1
      ${0} start
      ;;
```

```
   status)
      echo "Hostname is: $(hostname)"
      ip link show lo
      ;;

   *)
      echo "Usage: ${0} {start|stop|restart|status}"
      exit 1
      ;;
esac

exit 0

# End localnet
```

D.14. /etc/rc.d/init.d/sysctl

```
#!/bin/sh
########################################################################
# Begin sysctl
#
# Description : File uses /etc/sysctl.conf to set kernel runtime
#               parameters
#
# Authors     : Nathan Coulson (nathan@linuxfromscratch.org)
#               Matthew Burgress (matthew@linuxfromscratch.org)
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            sysctl
# Required-Start:      mountvirtfs
# Should-Start:        console
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Makes changes to the proc filesystem
# Description:         Makes changes to the proc filesystem as defined in
#                      /etc/sysctl.conf.  See 'man sysctl(8)'.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      if [ -f "/etc/sysctl.conf" ]; then
         log_info_msg "Setting kernel runtime parameters..."
         sysctl -q -p
         evaluate_retval
      fi
      ;;

   status)
      sysctl -a
      ;;
```

```
   *)
      echo "Usage: ${0} {start|status}"
      exit 1
      ;;
esac

exit 0

# End sysctl
```

D.15. /etc/rc.d/init.d/sysklogd

```
#!/bin/sh
########################################################################
# Begin sysklogd
#
# Description : Sysklogd loader
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            $syslog
# Required-Start:      $first localnet
# Should-Start:
# Required-Stop:       $local_fs
# Should-Stop:         sendsignals
# Default-Start:       2 3 4 5
# Default-Stop:        0 1 6
# Short-Description:   Starts kernel and system log daemons.
# Description:         Starts kernel and system log daemons.
#                      /etc/fstab.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      log_info_msg "Starting system log daemon..."
      parms=${SYSKLOGD_PARMS-'-m 0'}
      start_daemon /sbin/syslogd $parms
      evaluate_retval

      log_info_msg "Starting kernel log daemon..."
      start_daemon /sbin/klogd
      evaluate_retval
      ;;

   stop)
      log_info_msg "Stopping kernel log daemon..."
      killproc /sbin/klogd
      evaluate_retval

      log_info_msg "Stopping system log daemon..."
      killproc /sbin/syslogd
      evaluate_retval
```

```
      ;;

   reload)
      log_info_msg "Reloading system log daemon config file..."
      pid=`pidofproc syslogd`
      kill -HUP "${pid}"
      evaluate_retval
      ;;

   restart)
      ${0} stop
      sleep 1
      ${0} start
      ;;

   status)
      statusproc /sbin/syslogd
      statusproc klogd
      ;;

   *)
      echo "Usage: ${0} {start|stop|reload|restart|status}"
      exit 1
      ;;
esac

exit 0

# End sysklogd
```

D.16. /etc/rc.d/init.d/network

```
#!/bin/sh
########################################################################
# Begin network
#
# Description : Network Control Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            $network
# Required-Start:      $local_fs localnet swap
# Should-Start:        $syslog firewalld iptables nftables
# Required-Stop:       $local_fs localnet swap
# Should-Stop:         $syslog firewalld iptables nftables
# Default-Start:       2 3 4 5
# Default-Stop:        0 1 6
# Short-Description:   Starts and configures network interfaces.
# Description:         Starts and configures network interfaces.
# X-LFS-Provided-By:   LFS
### END INIT INFO

case "${1}" in
```

```
   start)
      # if the default route exists, network is already configured
      if ip route | grep -q "^default"; then return 0; fi
      # Start all network interfaces
      for file in /etc/sysconfig/ifconfig.*
      do
         interface=${file##*/ifconfig.}

         # Skip if $file is * (because nothing was found)
         if [ "${interface}" = "*" ]; then continue; fi

         /sbin/ifup ${interface}
      done
      ;;

   stop)
      # Unmount any network mounted file systems
       umount --all --force --types nfs,cifs,nfs4

      # Reverse list
      net_files=""
      for file in  /etc/sysconfig/ifconfig.*
      do
         net_files="${file} ${net_files}"
      done

      # Stop all network interfaces
      for file in ${net_files}
      do
         interface=${file##*/ifconfig.}

         # Skip if $file is * (because nothing was found)
         if [ "${interface}" = "*" ]; then continue; fi

         # See if interface exists
         if [ ! -e /sys/class/net/$interface ]; then continue; fi

         # Is interface UP?
         ip link show $interface 2>/dev/null | grep -q "state UP"
         if [ $? -ne 0 ];  then continue; fi

         /sbin/ifdown ${interface}
      done
      ;;

   restart)
      ${0} stop
      sleep 1
      ${0} start
      ;;

   *)
      echo "Usage: ${0} {start|stop|restart}"
      exit 1
      ;;
esac

exit 0

# End network
```

D.17. /etc/rc.d/init.d/sendsignals

```
#!/bin/sh
########################################################################
# Begin sendsignals
#
# Description : Sendsignals Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            sendsignals
# Required-Start:
# Should-Start:
# Required-Stop:       $local_fs swap localnet
# Should-Stop:
# Default-Start:
# Default-Stop:        0 6
# Short-Description:   Attempts to kill remaining processes.
# Description:         Attempts to kill remaining processes.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   stop)
      omit=$(pidof mdmon)
      [ -n "$omit" ] && omit="-o $omit"

      log_info_msg "Sending all processes the TERM signal..."
      killall5 -15 $omit
      error_value=${?}

      sleep ${KILLDELAY}

      if [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then
         log_success_msg
      else
         log_failure_msg
      fi

      log_info_msg "Sending all processes the KILL signal..."
      killall5 -9 $omit
      error_value=${?}

      sleep ${KILLDELAY}

      if [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then
         log_success_msg
      else
         log_failure_msg
      fi
      ;;

   *)
```

```
      echo "Usage: ${0} {stop}"
      exit 1
      ;;

esac

exit 0

# End sendsignals
```

D.18. /etc/rc.d/init.d/reboot

```
#!/bin/sh
########################################################################
# Begin reboot
#
# Description : Reboot Scripts
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Updates     : Bruce Dubbs - bdubbs@linuxfromscratch.org
#             : Pierre Labastie - pierre@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Notes       : Update March 24th, 2022: change "stop" to "start".
#               Add the $last facility to Required-start
#
########################################################################

### BEGIN INIT INFO
# Provides:            reboot
# Required-Start:      $last
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:       6
# Default-Stop:
# Short-Description:   Reboots the system.
# Description:         Reboots the System.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      log_info_msg "Restarting system..."
      reboot -d -f -i
      ;;

   *)
      echo "Usage: ${0} {start}"
      exit 1
      ;;

esac

# End reboot
```

D.19. /etc/rc.d/init.d/halt

```
#!/bin/sh
########################################################################
# Begin halt
#
# Description : Halt Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#             : Pierre Labastie - pierre@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Notes       : Update March 24th, 2022: change "stop" to "start".
#               Add the $last facility to Required-start
#
########################################################################

### BEGIN INIT INFO
# Provides:            halt
# Required-Start:      $last
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:       0
# Default-Stop:
# Short-Description:   Halts the system.
# Description:         Halts the System.
# X-LFS-Provided-By:   LFS
### END INIT INFO

case "${1}" in
   start)
      halt -d -f -i -p
      ;;

   *)
      echo "Usage: {start}"
      exit 1
      ;;
esac

# End halt
```

D.20. /etc/rc.d/init.d/template

```
#!/bin/sh
########################################################################
# Begin scriptname
#
# Description :
#
# Authors     :
#
# Version     : LFS x.x
#
# Notes       :
#
########################################################################
```

```
### BEGIN INIT INFO
# Provides:            template
# Required-Start:
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:
# Default-Stop:
# Short-Description:
# Description:
# X-LFS-Provided-By:
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
   start)
      log_info_msg "Starting..."
    # if it is possible to use start_daemon
      start_daemon fully_qualified_path
    # if it is not possible to use start_daemon
    # (command to start the daemon is not simple enough)
      if ! pidofproc daemon_name_as_reported_by_ps >/dev/null; then
         command_to_start_the_service
      fi
      evaluate_retval
      ;;

   stop)
      log_info_msg "Stopping..."
    # if it is possible to use killproc
      killproc fully_qualified_path
    # if it is not possible to use killproc
    # (the daemon shouldn't be stopped by killing it)
      if pidofproc daemon_name_as_reported_by_ps >/dev/null; then
         command_to_stop_the_service
      fi
      evaluate_retval
      ;;

   restart)
      ${0} stop
      sleep 1
      ${0} start
      ;;

   *)
      echo "Usage: ${0} {start|stop|restart}"
      exit 1
      ;;
esac

exit 0

# End scriptname
```

D.21. /etc/sysconfig/modules

```
################################################################################
# Begin /etc/sysconfig/modules
#
```

```
# Description : Module auto-loading configuration
#
# Authors     :
#
# Version     : 00.00
#
# Notes       : The syntax of this file is as follows:
#               <module> [<arg1> <arg2> ...]
#
# Each module should be on its own line, and any options that you want
# passed to the module should follow it.  The line deliminator is either
# a space or a tab.
#########################################################################

# End /etc/sysconfig/modules
```

D.22. /etc/sysconfig/createfiles

```
#########################################################################
# Begin /etc/sysconfig/createfiles
#
# Description : Createfiles script config file
#
# Authors     :
#
# Version     : 00.00
#
# Notes       : The syntax of this file is as follows:
#               if type is equal to "file" or "dir"
#                <filename> <type> <permissions> <user> <group>
#               if type is equal to "dev"
#                <filename> <type> <permissions> <user> <group> <devtype>
#             <major> <minor>
#
#               <filename> is the name of the file which is to be created
#               <type> is either file, dir, or dev.
#                       file creates a new file
#                       dir creates a new directory
#                       dev creates a new device
#               <devtype> is either block, char or pipe
#                       block creates a block device
#                       char creates a character device
#                       pipe creates a pipe, this will ignore the <major> and
#           <minor> fields
#               <major> and <minor> are the major and minor numbers used for
#     the device.
#########################################################################

# End /etc/sysconfig/createfiles
```

D.23. /etc/sysconfig/udev-retry

```
#########################################################################
# Begin /etc/sysconfig/udev_retry
#
# Description : udev_retry script configuration
#
# Authors     :
#
# Version     : 00.00
#
```

```
# Notes       : Each subsystem that may need to be re-triggered after mountfs
#               runs should be listed in this file.  Probable subsystems to be
#               listed here are rtc (due to /var/lib/hwclock/adjtime) and sound
#               (due to both /var/lib/alsa/asound.state and /usr/sbin/alsactl).
#               Entries are whitespace-separated.
########################################################################

rtc

# End /etc/sysconfig/udev_retry
```

D.24. /sbin/ifup

```
#!/bin/sh
########################################################################
# Begin /sbin/ifup
#
# Description : Interface Up
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
#
# Version     : LFS 7.7
#
# Notes       : The IFCONFIG variable is passed to the SERVICE script
#               in the /lib/services directory, to indicate what file the
#               service should source to get interface specifications.
#
########################################################################

up()
{
  log_info_msg "Bringing up the ${1} interface..."

  if ip link show $1 > /dev/null 2>&1; then
     link_status=`ip link show $1`

     if [ -n "${link_status}" ]; then
        if ! echo "${link_status}" | grep -q UP; then
           ip link set $1 up
        fi
     fi

  else
     log_failure_msg "Interface ${IFACE} doesn't exist."
     exit 1
  fi

  evaluate_retval
}

RELEASE="7.7"

USAGE="Usage: $0 [ -hV ] [--help] [--version] interface"
VERSTR="LFS ifup, version ${RELEASE}"

while [ $# -gt 0 ]; do
   case "$1" in
      --help | -h)     help="y"; break ;;
```

```
      --version | -V)  echo "${VERSTR}"; exit 0 ;;

      -*)              echo "ifup: ${1}: invalid option" >&2
                       echo "${USAGE}" >& 2
                       exit 2 ;;

      *)               break ;;
   esac
done

if [ -n "$help" ]; then
   echo "${VERSTR}"
   echo "${USAGE}"
   echo
   cat << HERE_EOF
ifup is used to bring up a network interface.  The interface
parameter, e.g. eth0 or eth0:2, must match the trailing part of the
interface specifications file, e.g. /etc/sysconfig/ifconfig.eth0:2.

HERE_EOF
   exit 0
fi

file=/etc/sysconfig/ifconfig.${1}

# Skip backup files
[ "${file}" = "${file%""~""}" ] || exit 0

. /lib/lsb/init-functions

if [ ! -r "${file}" ]; then
   log_failure_msg "Unable to bring up ${1} interface! ${file} is missing or cannot be accessed."
   exit 1
fi

.  $file

if [ "$IFACE" = "" ]; then
   log_failure_msg "Unable to bring up ${1} interface! ${file} does not define an interface [IFACE]."
   exit 1
fi

# Do not process this service if started by boot, and ONBOOT
# is not set to yes
if [ "${IN_BOOT}" = "1" -a "${ONBOOT}" != "yes" ]; then
   exit 0
fi

# Bring up the interface
if [ "$VIRTINT" != "yes" ]; then
   up ${IFACE}
fi

for S in ${SERVICE}; do
  if [ ! -x "/lib/services/${S}" ]; then
    MSG="\nUnable to process ${file}.  Either "
    MSG="${MSG}the SERVICE '${S} was not present "
    MSG="${MSG}or cannot be executed."
    log_failure_msg "$MSG"
    exit 1
  fi
done
```

```
if [ "${SERVICE}" = "wpa" ]; then log_success_msg; fi

# Create/configure the interface
for S in ${SERVICE}; do
  IFCONFIG=${file} /lib/services/${S} ${IFACE} up
done

# Set link up virtual interfaces
if [ "${VIRTINT}" == "yes" ]; then
   up ${IFACE}
fi

# Bring up any additional interface components
for I in $INTERFACE_COMPONENTS; do up $I; done

# Set MTU if requested. Check if MTU has a "good" value.
if test -n "${MTU}"; then
   if [[ ${MTU} =~ ^[0-9]+$ ]] && [[ $MTU -ge 68 ]] ; then
      for I in $IFACE $INTERFACE_COMPONENTS; do
         ip link set dev $I mtu $MTU;
      done
   else
      log_info_msg2 "Invalid MTU $MTU"
   fi
fi

# Set the route default gateway if requested
if [ -n "${GATEWAY}" ]; then
   if ip route | grep -q default; then
      log_warning_msg "Gateway already setup; skipping."
   else
      log_info_msg "Adding default gateway ${GATEWAY} to the ${IFACE} interface..."
      ip route add default via ${GATEWAY} dev ${IFACE}
      evaluate_retval
   fi
fi

# End /sbin/ifup
```

D.25. /sbin/ifdown

```
#!/bin/bash
########################################################################
# Begin /sbin/ifdown
#
# Description : Interface Down
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Notes       : the IFCONFIG variable is passed to the scripts found
#               in the /lib/services directory, to indicate what file the
#               service should source to get interface specifications.
#
########################################################################

RELEASE="7.0"

USAGE="Usage: $0 [ -hV ] [--help] [--version] interface"
```

```
VERSTR="LFS ifdown, version ${RELEASE}"

while [ $# -gt 0 ]; do
   case "$1" in
      --help | -h)     help="y"; break ;;

      --version | -V)  echo "${VERSTR}"; exit 0 ;;

      -*)              echo "ifup: ${1}: invalid option" >&2
                       echo "${USAGE}" >& 2
                       exit 2 ;;

      *)               break ;;
   esac
done

if [ -n "$help" ]; then
   echo "${VERSTR}"
   echo "${USAGE}"
   echo
   cat << HERE_EOF
ifdown is used to bring down a network interface.  The interface
parameter, e.g. eth0 or eth0:2, must match the trailing part of the
interface specifications file, e.g. /etc/sysconfig/ifconfig.eth0:2.

HERE_EOF
   exit 0
fi

file=/etc/sysconfig/ifconfig.${1}

# Skip backup files
[ "${file}" = "${file%""~""}" ] || exit 0

. /lib/lsb/init-functions

if [ ! -r "${file}" ]; then
   log_warning_msg "${file} is missing or cannot be accessed."
   exit 1
fi

. ${file}

if [ "$IFACE" = "" ]; then
   log_failure_msg "${file} does not define an interface [IFACE]."
   exit 1
fi

# We only need to first service to bring down the interface
S=`echo ${SERVICE} | cut -f1 -d" "`

if ip link show ${IFACE} > /dev/null 2>&1; then
   if [ -n "${S}" -a -x "/lib/services/${S}" ]; then
     IFCONFIG=${file} /lib/services/${S} ${IFACE} down
   else
     MSG="Unable to process ${file}.  Either "
     MSG="${MSG}the SERVICE variable was not set "
     MSG="${MSG}or the specified service cannot be executed."
     log_failure_msg "$MSG"
     exit 1
  fi
else
   log_warning_msg "Interface ${1} doesn't exist."
```

```
fi

# Leave the interface up if there are additional interfaces in the device
link_status=`ip link show ${IFACE} 2>/dev/null`

if [ -n "${link_status}" ]; then
   if [ "$(echo "${link_status}" | grep UP)" != "" ]; then
      if [ "$(ip addr show ${IFACE} | grep 'inet ')" == ""  ]; then
         log_info_msg "Bringing down the ${IFACE} interface..."
         ip link set ${IFACE} down
         evaluate_retval
      fi
   fi
fi

# End /sbin/ifdown
```

D.26. /lib/services/ipv4-static

```
#!/bin/sh
########################################################################
# Begin /lib/services/ipv4-static
#
# Description : IPV4 Static Boot Script
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

. /lib/lsb/init-functions
. ${IFCONFIG}

if [ -z "${IP}" ]; then
   log_failure_msg "\nIP variable missing from ${IFCONFIG}, cannot continue."
   exit 1
fi

if [ -z "${PREFIX}" -a -z "${PEER}" ]; then
   log_warning_msg "\nPREFIX variable missing from ${IFCONFIG}, assuming 24."
   PREFIX=24
   args="${args} ${IP}/${PREFIX}"

elif [ -n "${PREFIX}" -a -n "${PEER}" ]; then
   log_failure_msg "\nPREFIX and PEER both specified in ${IFCONFIG}, cannot continue."
   exit 1

elif [ -n "${PREFIX}" ]; then
   args="${args} ${IP}/${PREFIX}"

elif [ -n "${PEER}" ]; then
   args="${args} ${IP} peer ${PEER}"
fi

if [ -n "${LABEL}" ]; then
   args="${args} label ${LABEL}"
fi

if [ -n "${BROADCAST}" ]; then
```

```
   args="${args} broadcast ${BROADCAST}"
fi

case "${2}" in
   up)
      if [ "$(ip addr show ${1} 2>/dev/null | grep ${IP}/)" = "" ]; then
         log_info_msg "Adding IPv4 address ${IP} to the ${1} interface..."
         ip addr add ${args} dev ${1}
         evaluate_retval
      else
         log_warning_msg "Cannot add IPv4 address ${IP} to ${1}.  Already present."
      fi
   ;;

   down)
      if [ "$(ip addr show ${1} 2>/dev/null | grep ${IP}/)" != "" ]; then
         log_info_msg "Removing IPv4 address ${IP} from the ${1} interface..."
         ip addr del ${args} dev ${1}
         evaluate_retval
      fi

      if [ -n "${GATEWAY}" ]; then
         # Only remove the gateway if there are no remaining ipv4 addresses
         if [ "$(ip addr show ${1} 2>/dev/null | grep 'inet ')" != "" ]; then
            log_info_msg "Removing default gateway..."
            ip route del default
            evaluate_retval
         fi
      fi
   ;;

   *)
      echo "Usage: ${0} [interface] {up|down}"
      exit 1
   ;;
esac

# End /lib/services/ipv4-static
```

D.27. /lib/services/ipv4-static-route

```
#!/bin/sh
########################################################################
# Begin /lib/services/ipv4-static-route
#
# Description : IPV4 Static Route Script
#
# Authors     : Kevin P. Fleming - kpfleming@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

. /lib/lsb/init-functions
. ${IFCONFIG}

case "${TYPE}" in
   ("" | "network")
      need_ip=1
      need_gateway=1
```

```
   ;;

   ("default")
      need_gateway=1
      args="${args} default"
      desc="default"
   ;;

   ("host")
      need_ip=1
   ;;

   ("unreachable")
      need_ip=1
      args="${args} unreachable"
      desc="unreachable "
   ;;

   (*)
      log_failure_msg "Unknown route type (${TYPE}) in ${IFCONFIG}, cannot continue."
      exit 1
   ;;
esac

if [ -n "${GATEWAY}" ]; then
   MSG="The GATEWAY variable cannot be set in ${IFCONFIG} for static routes.\n"
   log_failure_msg "$MSG Use STATIC_GATEWAY only, cannot continue"
   exit 1
fi

if [ -n "${need_ip}" ]; then
   if [ -z "${IP}" ]; then
      log_failure_msg "IP variable missing from ${IFCONFIG}, cannot continue."
      exit 1
   fi

   if [ -z "${PREFIX}" ]; then
      log_failure_msg "PREFIX variable missing from ${IFCONFIG}, cannot continue."
      exit 1
   fi

   args="${args} ${IP}/${PREFIX}"
   desc="${desc}${IP}/${PREFIX}"
fi

if [ -n "${need_gateway}" ]; then
   if [ -z "${STATIC_GATEWAY}" ]; then
      log_failure_msg "STATIC_GATEWAY variable missing from ${IFCONFIG}, cannot continue."
      exit 1
   fi
   args="${args} via ${STATIC_GATEWAY}"
fi

if [ -n "${SOURCE}" ]; then
        args="${args} src ${SOURCE}"
fi

case "${2}" in
   up)
      log_info_msg "Adding '${desc}' route to the ${1} interface..."
      ip route add ${args} dev ${1}
      evaluate_retval
   ;;
```

```
   down)
      log_info_msg "Removing '${desc}' route from the ${1} interface..."
      ip route del ${args} dev ${1}
      evaluate_retval
   ;;

   *)
      echo "Usage: ${0} [interface] {up|down}"
      exit 1
   ;;
esac

# End /lib/services/ipv4-static-route
```

Appendix E. Udev configuration rules

The rules in this appendix are listed for convenience. Installation is normally done via instructions in Section 8.75, “Udev from Systemd-255”.

E.1. 55-lfs.rules

```
# /etc/udev/rules.d/55-lfs.rules: Rule definitions for LFS.

# Core kernel devices

# This causes the system clock to be set as soon as /dev/rtc becomes available.
SUBSYSTEM=="rtc", ACTION=="add", MODE="0644", RUN+="/etc/rc.d/init.d/setclock start"
KERNEL=="rtc", ACTION=="add", MODE="0644", RUN+="/etc/rc.d/init.d/setclock start"
```

Appendix F. LFS Licenses

This book is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 2.0 License.

Computer instructions may be extracted from the book under the MIT License.

F.1. Creative Commons License

Creative Commons Legal Code

Attribution-NonCommercial-ShareAlike 2.0

Important

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions
   a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
   b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
   c. "Licensor" means the individual or entity that offers the Work under the terms of this License.
   d. "Original Author" means the individual or entity who created the Work.
   e. "Work" means the copyrightable work of authorship offered under the terms of this License.
   f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

g. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

   b. to create and reproduce Derivative Works;

   c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

   d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works;

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(e) and 4(f).

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.

   b. You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with the same License Elements as this License, or a Creative Commons iCommons license that contains the same License Elements as this License (e.g. Attribution-NonCommercial-ShareAlike 2.0 Japan). You must include a copy of, or the Uniform Resource Identifier for, this License or other license specified in the previous sentence with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner

inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.

c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

d. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

e. For the avoidance of doubt, where the Work is a musical composition:

   i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

   ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation. 6. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Important

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at *http://creativecommons.org/*.

F.2. The MIT License

Index

Packages

Programs

Libraries

Scripts

Others