Cross-Compiled Linux From Scratch - Embedded

Version GIT-20240401-arm

Cross-Compiled Linux From Scratch - Embedded: Version GIT-20240401-arm

Table of Contents

Preface

Foreword

The Linux From Scratch Project has seen many changes in the few years of its existence. I personally became involved with the project in 2000, around the time of the 3.x releases. At that time, the build process was to create static binaries with the host system, then chroot and build the final binaries on top of the static ones.

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

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

CLFS Embedded follows the same guiding principles the LFS project has always followed, e.g., knowing your system inside and out by virtue of having built the system yourself. Additionally, during a CLFS Embedded build, you will learn advanced techniques such as cross-build toolchains, and how to create a smaller footprint system supporting architectures such as ARM and MIPS, in addition to x86.

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

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

Audience

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

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

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

Another benefit of CLFS is the ability to create a very compact Linux system. When installing a regular distribution, one is often forced to include several programs which are probably never used. These programs waste disk space or CPU cycles. It is not difficult to build an CLFS Embedded system of less than 10 megabytes (MB), which is substantially smaller than the majority of existing installations. Try that with a regular distribution!

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

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

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

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

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

Prerequisites

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

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

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

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

  This is a comprehensive guide to building and installing “generic” Unix software distributions under Linux.
- The Essential Pre-Reading Hint

http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt

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

Typography

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

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

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

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

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

Emphasis

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

http://clfs.org/

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

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

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

`[REPLACED TEXT]`

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

`passwd(5)`

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

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 Cross-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 including downloading the packages.

Part III - Make the Cross-Compile Tools

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

Part IV - Building the CLFS System

Part IV shows you how to build the core CLFS system. Compiling and installing BusyBox and other required packages, making the system bootable, setting up the bootscripts, and installing a bootloader.

Part V - Beyond CLFS Embedded

Part V provides instructions on optional packages that can be installed to enhance your CLFS system.

Part VI - Cleanup and Boot

Part VI finishes up the system by making sure the target file system is ready to be booted, copying it to the target, and finally booting the system.

Open Publication License

v1.0, 8 June 1999

I. REQUIREMENTS ON BOTH UNMODIFIED AND MODIFIED VERSIONS

The Open Publication works may be reproduced and distributed in whole or in part, in any medium physical or electronic, provided that the terms of this license are adhered to, and that this license or an incorporation of it by reference (with any options elected by the author(s) and/or publisher) is displayed in the reproduction.

Proper form for an incorporation by reference is as follows:

> Copyright © 1999-2024 by each contributor. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at *http://www.opencontent.org/openpub/*).

The reference must be immediately followed with any options elected by the author(s) and/or publisher of the document (see section VI).

Commercial redistribution of Open Publication-licensed material is permitted.

Any publication in standard (paper) book form shall require the citation of the original publisher and author. The publisher and author's names shall appear on all outer surfaces of the book. On all outer surfaces of the book the original publisher's name shall be as large as the bridgehead of the work and cited as possessive with respect to the bridgehead.

II. COPYRIGHT

The copyright to each Open Publication is owned by its author(s) or designee.

III. SCOPE OF LICENSE

The following license terms apply to all Open Publication works, unless otherwise explicitly stated in the document.

Mere aggregation of Open Publication works or a portion of an Open Publication work with other works or programs on the same media shall not cause this license to apply to those other works. The aggregate work shall contain a notice specifying the inclusion of the Open Publication material and appropriate copyright notice.

SEVERABILITY. If any part of this license is found to be unenforceable in any jurisdiction, the remaining portions of the license remain in force.

NO WARRANTY. Open Publication works are licensed and provided "as is" without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement.

IV. REQUIREMENTS ON MODIFIED WORKS

All modified versions of documents covered by this license, including translations, anthologies, compilations and partial documents, must meet the following requirements:

1. The modified version must be labeled as such.
2. The person making the modifications must be identified and the modifications dated.
3. Acknowledgement of the original author and publisher if applicable must be retained according to normal academic citation practices.
4. The location of the original unmodified document must be identified.
5. The original author's (or authors') name(s) may not be used to assert or imply endorsement of the resulting document without the original author's (or authors') permission.

V. GOOD-PRACTICE RECOMMENDATIONS

In addition to the requirements of this license, it is requested from and strongly recommended of redistributors that:

1. If you are distributing Open Publication works on hardcopy or CD-ROM, you provide email notification to the authors of your intent to redistribute at least thirty days before your manuscript or media freeze, to give the authors time to provide updated documents. This notification should describe modifications, if any, made to the document.
2. All substantive modifications (including deletions) be either clearly marked up in the document or else described in an attachment to the document.
3. Finally, while it is not mandatory under this license, it is considered good form to offer a free copy of any hardcopy and CD-ROM expression of an Open Publication-licensed work to its author(s).

VI. LICENSE OPTIONS

The author(s) and/or publisher of an Open Publication-licensed document may elect certain options by appending language to the reference to or copy of the license. These options are considered part of the license instance and must be included with the license (or its incorporation by reference) in derived works.

A. To prohibit distribution of substantively modified versions without the explicit permission of the author(s). "Substantive modification" is defined as a change to the semantic content of the document, and excludes mere changes in format or typographical corrections.

To accomplish this, add the phrase `Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.' to the license reference or copy.

B. To prohibit any publication of this work or derivative works in whole or in part in standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from the copyright holder.

To accomplish this, add the phrase 'Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.' to the license reference or copy.

Master Changelog

This is version GIT-20240401 of the Cross-Compiled Linux From Scratch book, dated April 01, 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 *http://trac.clfs.org/*.

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

Changelog Entries:

- March 25, 2024
  - [michele13] - We can now build all busybox applets with no problems.
  - [michele13] - Added symlink /init pointing to bin/busybox in ${CLFS}/targetfs/.
  - [michele13] - Fixed instructions for unpacking gmp mpc and mpfr during the toolchain build.
  - [michele13] - Fixed instructions for building dropbear.
- March 23, 2024
  - [michele13] - Fixed instructions for linux kernel build.
- March 22, 2024
  - [michele13] - Removed Iana-etc patch.
  - [michele13] - Fixed instructions for binutils, linux-headers and Iana-etc.
- March 21, 2024
  - [michele13] - Updated Busybox to version 1.36.1.
  - [michele13] - Updated Dropbear to version 2022.83.
  - [michele13] - Updated musl to version 1.2.5.
  - [michele13] - Updated Binutils to version 2.42.
  - [michele13] - Updated Linux to version 6.7.4.
  - [michele13] - Updated GCC to version 13.2.0.
  - [michele13] - Updated GMP to version 6.3.0.
  - [michele13] - Updated MPC to version 1.3.1.
  - [michele13] - Updated MPFR to version 4.2.1.
  - [michele13] - Fixed download links of Iana-etc and Wireless-Tools.
- April 19, 2019

    - [abradford] - Fix download links for MPC and bootscripts.
- October 12, 2017
    - [abradford] - Many small fixes submitted by "selk" on IRC and "akhiezer" via email.
    - [abradford] - Update Linux to v4.9.22.
    - [abradford] - Update musl to version 1.1.16.
    - [abradford] - Add new concept of "targetfs" directory parallel to the cross-tools which allows easier retaining of cross-tools across target builds.
    - [abradford] - Remove utmp, wtmp, and btmp creation.
- December 28, 2016
    - [abradford] - Add apt-get line to INSTALL document.
    - [abradford] - Update binutils to version 2.27.
    - [abradford] - Update busybox to version 1.24.2.
    - [abradford] - Update gmp to version 6.1.1.
    - [abradford] - Update Linux to version 4.4.21.
    - [abradford] - Update mpc to version 1.0.3.
    - [abradford] - Update mpfr to version 3.1.2.
    - [abradford] - Update musl to version 1.1.15.
    - [abradford] - Update GCC to version 6.2 which removes the need for patching GCC for compatibility with musl.
    - [abradford] - Update host requirements and script for m4 and ncurses5.
- June 13, 2014
    - [abradford] - Allow for easier starting over of targetfs build.
    - [abradford] - Remove e2fsprogs, busybox version of utilities are good enough.
    - [abradford] - Remove iptables, untested and largely unused.
    - [abradford] - Add netplug utility to beyond.
    - [abradford] - Update binutils to v2.24.
    - [abradford] - Update busybox to v1.22.1.
    - [abradford] - Update musl to v1.0.3.
    - [abradford] - Simplify beyond sections into one section.
- October 24, 2013
    - [abradford] - Shorten triplets.
    - [abradford] - Reorder variable tables for easier reading.
    - [abradford] - ARM and x86 are assumed little endian.
    - [abradford] - Fix musl installation symlinking.
    - [abradford] - Fix iana-etc 'make get'.
- October 21, 2013

- [abradford] - Cleanup the kernel build.
- [abradford] - Move bootloaders into bootable section, just give recommendations no exact steps.
- [abradford] - Create top level README and LICENSE.

- October 18, 2013
  - [abradford] - Fix the networking configuration to use eth0 and DHCP to obtain a valid IP address.
- October 17, 2013
  - [abradford] - Update dropbear to 2013.60 and fix instructions.
  - [abradford] - Assorted docbook updates for common pages.
  - [abradford] - Reorganize things people don't need to read into the preface and things that really should be read into the chapters.
  - [abradford] - Delete hostapd and lib-nl.
  - [abradford] - Delete dependencies information, most of it was wrong or outdated and so long as you follow the book in order there's no issues.
- October 16, 2013
  - [abradford] - Remove uClibc, use musl-libc.
  - [abradford] - Build cross compiler completely separate from target file system.
  - [abradford] - Build GMP, MPFR, and MPC within GCC tree.
  - [abradford] - Linewrap many configure switches for easier reading and editing.
- August 12, 2013
  - [abradford] - Don't copy libiberty.h from binutils.
  - [abradford] - Install linux headers directly without cp.
  - [abradford] - Revert install of headers to cross-tools.
- July 24, 2013
  - [abradford] - Fix binutils build issue with some Texinfo.
  - [abradford] - Install headers to cross-tools.
- June 17, 2013
  - [abradford] - Change partitioning to creating build dir.
- June 13, 2013
  - [abradford] - Remove WRT arch.
- June 11, 2013
  - [abradford] - Many package updates.
  - [abradford] - Removal of some LFS specific information.
  - [abradford] - Update host requirements to at least Debian Squeeze.
  - [kterrell] - Removal of sources from final tarball.
- September 22, 2012
  - [abradford] - Small fixes to prologue and ch 1 and 2.

- September 13, 2012
  - [ljump] - Fix System.map depmod command.
  - [ljump] - Fix console and null /dev node creation for final system.
- September 02, 2012
  - [William Harrington] - Fix uClibc patch MD5 sum.
- August 24, 2012
  - [William Harrington] - Change Beyond Net hostapd page to include bootscript and configuration sections.
- August 22, 2012
  - [William Harrington] - Change Beyond Net Dropbear page and edit the installation of bootscripts section.
  - [William Harrington] - Remove bash reference in hostreqs version script to use $SHELL variable.
- August 08, 2012
  - [William Harrington] - Update version check script to find the libc version with hosts that use paths other than /lib and /lib64, such as multiarch distro.
- May 14, 2011
  - [abradford] - Updated /etc/mdev.conf to Alpine Linux's example to provide a better baseline example.
  - [abradford] - Reverted changes that removed the mknod'ing of /dev/console and /dev/null. These nodes may be needed on some systems for proper booting.
  - [abradford] - Added Wireless Tools package to Beyond Networking section. Thanks to Ivan Castell Rovira for providing a set of instructions for the installation of this package.
  - [abradford] - Fixed some Dropbear issues with symlinks and directory naming.
- April 20, 2011
  - [abradford] - Created a Beyond section just for extra libraries and moved zlib into it.
- April 19, 2011
  - [abradford] - Created a Beyond section just for networking tools.
  - [abradford] - Created a Beyond section just for file system tools and moved e2fsprogs into it.
- April 8, 2011
  - [jciccone] - Cleaned up the acknowledgements page.
- March 31, 2011
  - [abradford] - Updated BusyBox to version 1.18.4.
  - [abradford] - Updated GCC to version 4.6.0.
  - [abradford] - Updated MPC to version 0.9.
  - [abradford] - Updated zlib to version 1.2.5.
  - [abradford] - Updated Linux kernel to version 2.6.38.2.
  - [abradford] - Require Gawk 3.1 or greater because of Iana-Etc requirements.
  - [abradford] - Updated "What Now?" section to include CBLFS link. Thanks go to Adrian Grigo.
  - [abradford] - Link ${CLFS}/etc/init.d/rcS to ${CLFS}/etc/rc.d/startup so that BusyBox's init can find the startup scripts.

    - [abradford] - Updated mdev configuration such that video output is disabled by default to ensure no errors if video is not available.
- March 23, 2011
    - [abradford] - Remove mknod commands as mdev takes care of creating all required entries in /dev automatically.
    - [abradford] - Moved change ownership section to be as late in the book as possible and only operate on a copy of the file system.
- March 18, 2011
    - [abradford] - Copy libgcc library to target as it's needed by at least e2fsprogs.
- March 7, 2011
    - [abradford] - Added optional information for building C++ compiler.
    - [abradford] - Updated bootscripts install target and package contents to match current bootscripts.
- February 24, 2011
    - [abradford] - Added --disable-multilib to GCC builds.
- February 10, 2011
    - [abradford] - Changed GCC Static and Final to build GCC for ${CLFS_ABI} for ARM and MIPS, ${CLFS_CPU} for x86. This removes the need for using the ${BUILD} variable when building packages.
    - [abradford] - Updated Resources section.
- January 30, 2011
    - [abradford] - Updated MPFR download link.
    - [abradford] - Updated GCC to version 4.5.2.
    - [abradford] - Updated Binutils to version 2.21.
    - [abradford] - Added information to GCC's configure options.
    - [abradford] - Updated Iana-Etc to use more up-to-date data.
- Janurary 10, 2011
    - [jciccone] - Changes Submitted By Andrew Bradford via the CLFS-Dev Mailing List. Updated E2fsprogs to 1.41.14 and changed some configure switches for uClibc compatibility.
- November 28, 2010
    - [jciccone] - Updated BusyBox to 1.17.3.
    - [jciccone] - Updated uClibc to 0.9.31.
- November 21, 2010
    - [jciccone] - Added MPC 0.8.2.
    - [jciccone] - Updated GCC to 4.5.1.
    - [jciccone] - Updated Binutils to 2.20.1.
    - [jciccone] - Updated MPFR to 3.0.0.
    - [jciccone] - Updated GMP to 5.0.1.
    - [jciccone] - Updated the Linux Kernel to 2.6.36.

    - [jciccone] - Make sure we change CLFS_HOST before we set it. Also make sure that CLFS_HOST gets saved to the bashrc.
- November 8, 2008
    - [jciccone] - Updated Busybox to 1.12.1.
    - [jciccone] - Updated uClibc to 0.9.30-rc3.
- November 8, 2008
    - [jciccone] - Dropped the uClibc Headers Page.
    - [jciccone] - Updated GCC to 4.3.2.
    - [jciccone] - Updated Binutils to 2.19.
- Janurary 9, 2008
    - [jciccone] - Updated Busybox to 1.8.2.
- November 5, 2006
    - [jim] - Start of CLFS Embedded Development.

Cross-LFS Acknowledgements

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

Our Leaders:

- Ryan Oliver - Build Process Developer.
- Joe Ciccone - Lead Developer.
- Jim Gifford - Lead Developer.

Our CLFS Team:

- Andrew Bradford - Embedded Developer.
- Manuel Canales Esparcia - Book XML.
- Justin Knierim - Website Architect.
- Maarten Lankhorst - MIPS/WRT Embedded Developer.
- Chris Staub - Testing and Quality Control.
- Zack Winkles - Testing and Quality Control.

Thank you all for your support.

Errata

The software used to create a CLFS system is constantly being updated and enhanced. Security warnings and bug fixes may become available after the CLFS book has been released. Once a release of the CLFS Embedded book is made, this page will provide a link to the CLFS Embedded errata page.

Part I. Introduction

Chapter 1. Introduction

1.1. How to Build a CLFS System

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

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

Build Directory, Packages, and Patches of this book describes how to create a temporary build directory and which packages and patches need to be downloaded to build a CLFS system. Final Preparations discusses the setup for an appropriate working environment. Please read Final Preparations carefully as it explains several important issues the developer should be aware of before beginning to work through Constructing Cross-Compile Tools and beyond.

Constructing Cross-Compile Tools explains the installation of cross-compile tools which will be built on the host but be able to compile programs that run on the target machine. These cross-compile tools will be used to create the final-system.

The process of building cross-compile tools first involves installing binutils into ${CLFS}/cross-tools, so that we have an asembler and a linker for our target architecture. GCC is then compiled statically and installed into ${CLFS}/cross-tools, this cross-compiler is used to build the libc for the final-system. The GCC cross-compiler is then rebuilt dynamically - this final cross-compiler is what will be used to build the final-system.

In Installing Basic System Software, the full CLFS system is cross-compiled. The system is built using a sysroot compiler. Sysroot is a parameter passed to binutils and gcc that modifies its default search paths.

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

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

1.2. Host System Requirements

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

- **Bash-4.0**
- **Binutils-2.20**
- **Bzip2-1.0.5**
- **Coreutils-8.1**
- **Diffutils-3.0**
- **Findutils-4.4.0**
- **Gawk-3.1**
- **GCC-4.4**
- **Glibc-2.11**
- **Grep-2.6**
- **Gzip-1.3**
- **M4-1.4.16**
- **Make-3.81**
- **ncurses5**
- **libelf from Elfutils**
- **libssl from OpenSSL**
- **Patch-2.6**
- **Sed-4.2.1**
- **Sudo-1.7.4p4**
- **Tar-1.23**
- **Texinfo-4.13**

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

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

# Simple script to list version numbers of critical development tools
set -e
bash --version | head -n1 | cut -d" " -f2-4
echo -n "Binutils: "; ld --version | head -n1 | cut -d" " -f3-
bzip2 --version 2>&1 < /dev/null | head -n1 | cut -d" " -f1,6-
echo -n "Coreutils: "; chown --version | head -n1 | cut -d")" -f2
diff --version | head -n1
find --version | head -n1
gawk --version | head -n1
gcc --version | head -n1
ldd $(which ${SHELL}) | grep libc.so | cut -d ' ' -f 3 | ${SHELL} | head -n 1 \
| cut -d ' ' -f 1-10
grep --version | head -n1
gzip --version | head -n1
m4 --version | head -n1
make --version | head -n1
echo "#include <ncurses.h>" | gcc -E - > /dev/null
echo "#include <openssl/ssl2.h>" | gcc -E - > /dev/null
echo "#include <libelf.h>" | gcc -E - > /dev/null
patch --version | head -n1
sed --version | head -n1
sudo -V | head -n1
tar --version | head -n1
makeinfo --version | head -n1

EOF

bash version-check.sh
```

1.3. Resources

1.3.1. FAQ

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

1.3.2. Mailing Lists

The `clfs.org` server hosts a number of mailing lists used for the development of the CLFS project. These lists include the main development and support lists, among others. If the FAQ does not solve the problem you are having, the next step would be to search the mailing lists at *http://trac.clfs.org/wiki/lists*.

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

1.3.3. IRC

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

1.3.4. Mirror Sites

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

1.3.5. Contact Information

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

1.4. Help

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

We also have a wonderful CLFS community that is willing to offer assistance through the mailing lists and IRC (see the Section 1.3, “Resources” section of this book). However, we get several support questions everyday and many of them can be easily answered by going to the FAQ and by searching the mailing lists first. So for us to offer the best assistance possible, you need to do some research on your own first. This allows us to focus on the more unusual support needs. If your searches do not produce a solution, please include all relevant information (mentioned below) in your request for help.

1.4.1. Things to Mention

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

- The version of the book being used (in this case CLFS Embedded GIT-20240401)
- The host distribution and version being used to create CLFS.
- The architecture of the host and target.
- The value of the ${CLFS_TARGET}, and ${BUILD} environment variables.
- The package or section in which the problem was encountered.
- The exact error message or symptom received. See Section 1.4.3, “Compilation Problems” below for an example.
- Note whether you have deviated from the book at all. A package version change or even a minor change to any command is considered deviation.

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

1.4.2. Configure Script Problems

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

1.4.3. Compilation Problems

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

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

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

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

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

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

Part II. Preparing for the Build

Chapter 2. Build Directory, Packages, and Patches

2.1. Introduction

This chapter creates a directory where the cross compiler toolchain wil be built and a directory where sources are kept. It includes a list of packages that need to be downloaded for building a basic Linux system.

The listed version numbers correspond to versions of the software that are known to work, and this book is based on their use. We highly recommend not using newer or older versions because the build commands for one version may not work with another version.

Download locations may not always be accessible. If a download location has changed since this book was published, Google (*http://www.google.com/*) provides a useful search engine for most packages.

2.2. Creating Build and Sources Directories

Create a directory for the CLFS build. The exact location is up to you but depending on the location you choose, you may need to do this as the root user:

```
sudo mkdir -p /mnt/clfs
```

Assign it to the `CLFS` environment variable:

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

Ensure that this new directory has permissions that are not too restrictive such that you can write to it as a non-root user. Depending on the location of the `CLFS` directory, you may need to do this as the root user:

```
sudo chmod 777 ${CLFS}
```

Downloaded packages and patches will need to be stored somewhere that is conveniently available throughout the entire build. A working directory is also required to unpack the sources and build them. `${CLFS}/sources` can be used both as the place to store the tarballs and patches and as a working directory.

Create a directory to store the sources:

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

2.3. All Packages

Download or otherwise obtain the following packages:

• Binutils (2.42) - 26922 KB:
Home page: *http://www.gnu.org/software/binutils/*
Download: *http://ftp.gnu.org/gnu/binutils/binutils-2.42.tar.xz*
MD5 sum: `a075178a9646551379bfb64040487715`

• BusyBox (1.36.1) - 2466 KB:
Home page: *http://www.busybox.net*
Download: *http://busybox.net/downloads/busybox-1.36.1.tar.bz2*
MD5 sum: `0fc591bc9f4e365dfd9ade0014f32561`

• CLFS-Bootscripts (git master HEAD) - 3 KB:
Home page: *https://github.com/cross-lfs/bootscripts-embedded*
Download: *https://github.com/cross-lfs/bootscripts-embedded/archive/master.tar.gz*
MD5 sum: `N/A`

• GCC (13.2.0) - 85800 KB:
Home page: *http://gcc.gnu.org*
Download: *http://gcc.gnu.org/pub/gcc/releases/gcc-13.2.0/gcc-13.2.0.tar.xz*
MD5 sum: `e0e48554cc6e4f261d55ddee9ab69075`

• GMP (6.3.0) - 2046 KB:
Home page: *https://www.gnu.org/software/gmp/*
Download: *http://ftp.gnu.org/gnu/gmp/gmp-6.3.0.tar.xz*
MD5 sum: `956dc04e864001a9c22429f761f2c283`

• iana-etc (20240125) - 200 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`

• Linux (6.7.4) - 138130 KB:
Home page: *http://www.kernel.org*
Download: *http://www.kernel.org/pub/linux/kernel/v6.x/linux-6.7.4.tar.xz*
MD5 sum: `370e1b6155ae63133380e421146619e0`

• MPC (1.3.1) - 655 KB:
Home page: *http://www.multiprecision.org/*
Download: *http://ftp.gnu.org/gnu/mpc/mpc-1.3.1.tar.gz*
MD5 sum: `5c9bc658c9fd0f940e8e3e0f09530c62`

• MPFR (4.2.1) - 1459 KB:
Home page: *http://www.mpfr.org/*
Download: *http://ftp.gnu.org/gnu/mpfr/mpfr-4.2.1.tar.xz*
MD5 sum: `523c50c6318dde6f9dc523bc0244690a`

• musl (1.2.5) - 1055 KB:
Home page: *https://musl.libc.org/*
Download: *https://musl.libc.org/releases/musl-1.2.5.tar.gz*
MD5 sum: `ac5cfde7718d0547e224247ccfe59f18`

Total size of these packages: about 253 MB

2.4. Needed Patches

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

There aren't any patches to download at this time.

Total size of these patches: about 0 KB

Chapter 3. Final Preparations

3.1. About $CLFS

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

```
echo ${CLFS}
```

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

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

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

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

3.2. Adding the CLFS User

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

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

The meaning of the command line options:

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

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

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

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

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

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

```
sudo passwd clfs
```

Grant `clfs` full access to `${CLFS}` by making `clfs` the directory's owner. You may need to do this as the root user:

```
sudo chown -Rv clfs ${CLFS}
```

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

```
su - clfs
```

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

3.3. Setting Up the Environment

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

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

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

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

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

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

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

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

The LC_ALL variable controls the localization of certain programs, making their messages follow the conventions of a specified country. If the host system uses a version of Glibc older than 2.2.4, having LC_ALL set to something other than “POSIX” or “C” (during this chapter) may cause issues.

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

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

```
source ~/.bash_profile
```

Part III. Make the Cross-Compile Tools

Chapter 4. Constructing Cross-Compile Tools

4.1. Introduction

This chapter shows you how to create cross platform tools.

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

4.1.1. Common Notes

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

Several of the packages are patched before compilation, but only when the patch is needed to circumvent a problem or supply a default configuration. Warning messages about *offset* or *fuzz* may also be encountered when applying a patch. Do not worry about these warnings, as the patch was still successfully applied.

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

> **Important**
>
> After installing each package, both in this and the next chapters, delete its source and build directories, unless specifically instructed otherwise.

4.2. Build CFLAGS

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

To disable `CFLAGS` use the following commands:

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

4.3. Build Variables

Setting Host and Target

During the building of the cross-compile tools, you will need to set a few variables that will be dependent on your particular needs. You will need to select the target triplet for the target architecture, the CPU architecture, the CPU floating point hardware availability, and (if available) the type of floating point hardware. If you do not know what values can be chosen for each of these, you can use the tables as a reference.

If your processor is an ARM9, good choices include: triplet of arm-linux-musleabi, ARM arch of armv5t, and float of soft. ARM9 processors do not usually have hardware floating point abilities. If your processor is a Cortex-A series, which often have hardware floating point capability, good choices include: triplet of arm-linux-musleabihf, ARM arch of armv7-a, float of hard, and fpu of vfpv3-d16.

If your target CPU has hardware floating point support (not all ARM CPUs do), then set the following CLFS_FLOAT variable to either "hard" or "softfp". Use "softfp" if you need to integrate binaries compiled with "soft". Use "hard" if you don't. If your target CPU does not have hard floating point support, set the following CLFS_FLOAT vairable to "soft".

```
export CLFS_FLOAT="[hard, softfp, or soft]"
```

If you chose either "hard" or "softfp" for CLFS_FLOAT, you now need to set which floating point hardware is actually included (see table below) with your ARM CPU:

```
export CLFS_FPU="[fpu version]"
```

Table 4.1. ARM Hard Floating Point Versions

fpa	fpe2	fpe3	maverick
vfp	vfpv3	vfpv3-fp16	vfpv3-d16
vfpv3-d16-fp16	vfpv3xd	vfpv3xd-fp16	neon
neon-fp16	vfpv4	vfpv4-d16	fpv4-sp-d16
neon-vfpv4			

Then, set the host and target triplets:

```
export CLFS_HOST=$(echo ${MACHTYPE} | sed "s/-[^-]*/-cross/")
export CLFS_TARGET="[target triplet]"
```

Table 4.2. Target Triplets

Float Type	**Triplet**
soft or softfp	arm-linux-musleabi
hard	arm-linux-musleabihf

Now set the architecture of the CPU:

```
export CLFS_ARCH=arm
```

Choose the ARM architecture (see table below):

```
export CLFS_ARM_ARCH="[architecture]"
```

Table 4.3. ARM Archiecture Choices

armv4t	armv5	armv5t	armv5te
armv6	armv6j	armv6t2	armv6z
armv6zk	armv6-m	armv7	armv7-a
armv7-r	armv7-m		

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

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

4.4. Create the Cross Tools Sysroot Directory

Create a sysroot directory which will be used when building the cross compiler and link its `usr` directory to itself such that everything installs to the sysroot:

```
mkdir -p ${CLFS}/cross-tools/${CLFS_TARGET}
ln -sfv . ${CLFS}/cross-tools/${CLFS_TARGET}/usr
```

4.5. Linux-Headers-6.7.4

The Linux Kernel contains a make target that installs “sanitized” kernel headers.

4.5.1. Installation of Linux Headers

For this step you will need the kernel tarball.

Install the header files that are common to all architectures:

```
make mrproper
make ARCH=${CLFS_ARCH} headers
find usr/include -type f ! -name '*.h' -delete
cp -rv usr/include ${CLFS}/cross-tools/${CLFS_TARGET}
```

4.5.2. Contents of Linux-Headers

Installed headers: ${CLFS}/cross-tools/${CLFS_TARGET}/include/{asm,asm-generic,drm,linux,mtd,rdma,scsi,sound,video,xen}/*.h

Short Descriptions

`${CLFS}/cross-tools/${CLFS_TARGET}/include/{asm,asm-generic,drm,linux,mtd,rdma,scsi,sound,video,xen}/*.h`	The Linux API headers

4.6. Binutils-2.42

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

4.6.1. Installation of Cross Binutils

It is important that Binutils be the first package compiled because both the C library 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 outside of the source directory in a dedicated build directory:

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

Prepare Binutils for compilation:

```
../binutils-2.42/configure \
   --prefix=${CLFS}/cross-tools \
   --target=${CLFS_TARGET} \
   --with-sysroot=${CLFS}/cross-tools/${CLFS_TARGET} \
   --disable-nls \
   --enable-gprofng=no \
   --disable-multilib
```

The meaning of the configure options:

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

`--target=${CLFS_TARGET}`
This creates a cross-architecture executable that creates files for ${CLFS_TARGET} but runs on the host system.

`--with-sysroot=${CLFS}/cross-tools/${CLFS_TARGET}`
This tells configure that ${CLFS} is going to be the root of our system. It will now use the specified sysroot, ${CLFS}, as a prefix of the default search paths.

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

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

Compile the package:

```
make configure-host
make
```

The meaning of the make options:

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

Install the package:

```
make install
```

4.6.2. Contents of Binutils

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

Short Descriptions

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

ar Creates, modifies, and extracts from archives

as An assembler that assembles the output of **gcc** into object files

c++filt Used by the linker to de-mangle C++ and Java symbols and to keep overloaded functions from clashing

elfedit Examine and modify ELF metadata within an ELF object

gprof Displays call graph profile data

ld A linker that combines a number of object and archive files into a single file, relocating their data and tying up symbol references

nm Lists the symbols occurring in a given object file

objcopy Copy the contents of one object file to another

objdump Displays information about the given object file, with options controlling the particular information to display; the information shown is useful to programmers who are working on the compilation tools

ranlib Generates an index of the contents of an archive and stores it in the archive; the index lists all of the symbols defined by archive members that are relocatable object files

readelf Displays information about ELF type binaries

size Lists the section sizes and the total size for the given object files

strings Outputs, for each given file, the sequences of printable characters that are of at least the specified length (defaulting to four); for object files, it prints, by default, only the strings from the initializing and loading sections while for other types of files, it scans the entire file

strip Discards symbols from object files

`libiberty` Contains routines used by various GNU programs, including **getopt**, **obstack**, **strerror**, **strtol**, and **strtoul**

`libbfd` The Binary File Descriptor library

`libopcodes` A library for dealing with opcodes—the “readable text” versions of instructions for the processor; it is used for building utilities like **objdump**.

4.7. GCC-13.2.0 - Static

The GCC package contains the GNU compiler collection, which includes the C compiler. This build of GCC is mainly done so that the C library can be built next.

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

GCC requires the GMP, MPFR, and MPC packages to either be present on the host or to be present in source form within the gcc source tree. Unpack these into the GCC directory after unpacking GCC:

```
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
```

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

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

Prepare GCC for compilation:

```
../gcc-13.2.0/configure \
  --prefix=${CLFS}/cross-tools \
  --build=${CLFS_HOST} \
  --host=${CLFS_HOST} \
  --target=${CLFS_TARGET} \
  --with-sysroot=${CLFS}/cross-tools/${CLFS_TARGET} \
  --disable-nls \
  --disable-shared \
  --without-headers \
  --with-newlib \
  --disable-decimal-float \
  --disable-libgomp \
  --disable-libmudflap \
  --disable-libssp \
  --disable-libatomic \
  --disable-libquadmath \
  --disable-threads \
  --enable-languages=c \
  --disable-multilib \
  --with-mpfr-include=$(pwd)/../gcc-13.2.0/mpfr/src \
  --with-mpfr-lib=$(pwd)/mpfr/src/.libs \
  --with-arch=${CLFS_ARM_ARCH} \
  --with-float=${CLFS_FLOAT} \
  --with-fpu=${CLFS_FPU}
```

The meaning of the configure options:

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

`--build=${CLFS_HOST}`
This tells the configure script the triplet to use to build GCC. It will use ${CLFS_HOST} as that's where it's being built.

`--host=${CLFS_HOST}`
This tells the configure script the triplet of the machine GCC will be executed on when actually cross compiling. It will use ${CLFS_HOST} as that's where GCC will execute when cross compiling software later.

`--target=${CLFS_TARGET}`
This tells the configure script the triplet of the machine GCC will build executables for. It will use ${CLFS_TARGET} so that software compiled with this version of GCC can be executed on the embedded machine target.

`--with-sysroot=${CLFS}/cross-tools/${CLFS_TARGET}`
This tells configure that ${CLFS}/cross-tools/${CLFS_TARGET} is going to be the temporary root of our system. It will now use the specified sysroot as a prefix of the default search paths.

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

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

`--without-headers`
Tells configure to not use any headers from any C libraries. This is needed as we haven't yet built the C library and to prevent influence from the host environment.

`--with-newlib`
Tells configure to build libgcc without needing any C libraries.

`--disable-decimal-float`
Tells configure to disable IEEE 754-2008 decimal floating point support. Decimal floating point support isn't needed yet.

`--disable-libgomp`
Tells configure to not build the GOMP run-time libraries. GOMP is the GNU implementation of OpenMP, a API for shared-memory parallel programming.

`--disable-libmudflap`
Tells configure to not build libmudflap. Mudflap is a library that can be used to help check for proper pointer usage.

`--disable-libssp`
Tells configure not to build run-time libraries for stack smashing detection.

`--disable-libatomic`
Tells configure not to build atomic operations.

`--disable-libquadmath`
Tells configure not to build quad math operations.

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

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

`--disable-multilib`
This option specifies that multiple target libraries should not be built.

`--with-mpfr-include=$(pwd)/../gcc-13.2.0/mpfr/src`
Tells configure how to find the mpfr headers.

`--with-mpfr-lib=$(pwd)/mpfr/src/.libs`
Tells configure to use the mpfr libraries built within the GCC build directory. This happens automatically but is needed to prevent GCC from searching the host's normal library paths.

`--with-arch=${CLFS_ARM_ARCH}`
This option sets the ARM architecture selected earlier.

`--with-float=${CLFS_FLOAT}`
This option sets the floating point mode selected earlier.

`--with-fpu=${CLFS_FPU}`
This option sets the hardware floating point type selected earlier. If soft floating point was selected, this value is ignored.

Continue with compiling the package:

```
make all-gcc all-target-libgcc
```

Install the package:

```
make install-gcc install-target-libgcc
```

Details on this package are located in Section 4.9.2, “Contents of GCC.”

4.8. musl-1.2.5

The musl 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.

4.8.1. Installation of musl

Configure the package:

```
./configure \
  CROSS_COMPILE=${CLFS_TARGET}- \
  --prefix=/ \
  --target=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install the package:

```
DESTDIR=${CLFS}/cross-tools/${CLFS_TARGET} make install
```

4.8.2. Contents of musl

Installed Programs: ld-musl.so.0
Installed Libraries: libc.so.0, libcrypt.so.0, libdl.so.0, libm.so.0, libpthread.so.0, librt.so.0
Installed Headers: To be written...

Short Descriptions

ld-musl	The musl dynamic linker / loader
`libc`	The C library
`libcrypt`	The cryptographic library
`libdl`	The musl dynamic linker / loader library
`libm`	The math library
`libpthread`	The POSIX thread library
`librt`	The clock and timer library

4.9. GCC-13.2.0 - Final

The GCC package contains the GNU compiler collection, which includes the C compiler. This second build of GCC will produce the final cross compiler which will use the previously built C library.

4.9.1. Installation of GCC Cross Compiler

GCC requires the GMP, MPFR, and MPC packages to either be present on the host or to be present in source form within the gcc source tree. Unpack these into the GCC directory after unpacking GCC:

```
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
```

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

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

Prepare GCC for compilation:

> **Note**
>
> If you would like to build a C++ compiler in addition to the C compiler, change the following --enable-languages=c option to be --enable-languages=c,c++ instead. A C++ compiler is not required for any of the software included in this book.

```
../gcc-13.2.0/configure \
  --prefix=${CLFS}/cross-tools \
  --build=${CLFS_HOST} \
  --host=${CLFS_HOST} \
  --target=${CLFS_TARGET} \
  --with-sysroot=${CLFS}/cross-tools/${CLFS_TARGET} \
  --disable-nls \
  --enable-languages=c \
  --enable-c99 \
  --enable-long-long \
  --disable-libmudflap \
  --disable-multilib \
  --with-mpfr-include=$(pwd)/../gcc-13.2.0/mpfr/src \
  --with-mpfr-lib=$(pwd)/mpfr/src/.libs \
  --with-arch=${CLFS_ARM_ARCH} \
  --with-float=${CLFS_FLOAT} \
  --with-fpu=${CLFS_FPU}
```

The meaning of the configure options not used previously:

`--enable-c99`
Enable C99 support for C programs.

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

Continue with compiling the package:

```
make
```

Install the package:

```
make install
```

4.9.2. Contents of GCC

Installed programs: gcc, and gcov
Installed libraries: libgcc.a, libgcc_eh.a, and libgcc_s.so

Short Descriptions

gcc The C compiler

gcov A coverage testing tool; it is used to analyze programs to determine where optimizations will have the most effect

`libgcc` Contains run-time support for **gcc**

4.10. Create the Target File System Directory

Create a target file system directory which will be used when building the programs and libraries which will deploy to the target system:

```
mkdir -pv ${CLFS}/targetfs
```

4.11. ToolChain Variables

Setup target-specific variables for the compiler, linker, and other compile-time tools. For the compiler and linker, use the target file system directory as a sysroot rather than the sysroot we setup before for the cross-tools. This time we write them to `~/.bashrc` so that they persist even if the clfs user logs out:

```
echo export CC=\""${CLFS_TARGET}-gcc --sysroot=${CLFS}/targetfs\"" >> ~/.bashrc
echo export CXX=\""${CLFS_TARGET}-g++ --sysroot=${CLFS}/targetfs\"" >> ~/.bashrc
echo export AR=\""${CLFS_TARGET}-ar\"" >> ~/.bashrc
echo export AS=\""${CLFS_TARGET}-as\"" >> ~/.bashrc
echo export LD=\""${CLFS_TARGET}-ld --sysroot=${CLFS}/targetfs\"" >> ~/.bashrc
echo export RANLIB=\""${CLFS_TARGET}-ranlib\"" >> ~/.bashrc
echo export READELF=\""${CLFS_TARGET}-readelf\"" >> ~/.bashrc
echo export STRIP=\""${CLFS_TARGET}-strip\"" >> ~/.bashrc
source ~/.bashrc
```

Part IV. Building the CLFS System

Chapter 5. Installing Basic System Software

5.1. Introduction

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

The order that packages are installed in this chapter needs to be strictly followed to ensure that no program accidentally acquires a path referring to `${CLFS}/cross-tools` hard-wired into it. For the same reason, do not compile packages in parallel. Compiling more than one package at a time may save time, but it could result in a program containing a hard-wired path to `${CLFS}/cross-tools`, which will cause the program to stop working when that directory is removed.

Compiling any single package using make's parallel job execution option, "-j" is OK if you want to speed up compiling any one package.

5.2. Creating Directories

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

```
mkdir -pv ${CLFS}/targetfs/{bin,boot,dev,etc,home,lib/{firmware,modules}}
mkdir -pv ${CLFS}/targetfs/{mnt,opt,proc,sbin,srv,sys}
mkdir -pv ${CLFS}/targetfs/var/{cache,lib,local,lock,log,opt,run,spool}
install -dv -m 0750 ${CLFS}/targetfs/root
install -dv -m 1777 ${CLFS}/targetfs/{var/,}tmp
mkdir -pv ${CLFS}/targetfs/usr/{,local/}{bin,include,lib,sbin,share,src}
```

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

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

5.2.1. FHS Compliance Note

The directory tree is based on the Filesystem Hierarchy Standard (FHS) (available at *http://refspecs.linuxfoundation.org/fhs.shtml*).

5.3. Creating the passwd, group, and lastlog Files

A proper Linux system maintains a list of the mounted file systems in the file `/etc/mtab`. With the way our embedded system is designed, we will be using a symlink to `/proc/mounts`:

```
ln -svf ../proc/mounts ${CLFS}/targetfs/etc/mtab
```

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

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

Additional optional users you may want to add:

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

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

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

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

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

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

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

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

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

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

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

```
cat > ${CLFS}/targetfs/etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
EOF
```

Additional optional groups you may want to add

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

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

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

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

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

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

`users:x:100:`
: The default GID used by shadow for new users

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

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

The created groups are not part of any standard—they are groups decided on in part by the requirements of BusyBox later in this chapter, and in part by common convention employed by a number of existing Linux distributions. The Linux Standard Base (LSB, available at *http://www.linuxbase.org*) recommends only that, besides the group `root`

with a Group ID (GID) of 0, a group `bin` with a GID of 1 be present. All other group names and GIDs can be chosen freely by the system administrator since well-written programs do not depend on GID numbers, but rather use the group's name.

The **login**, **agetty**, and **init** programs (and others) use the lastlog file to record information such as who was logged into the system and when. However, these programs will not write to the lastlog file if it does not already exist. Initialize the lastlog file and give it proper permissions:

```
touch ${CLFS}/targetfs/var/log/lastlog
chmod -v 664 ${CLFS}/targetfs/var/log/lastlog
```

5.4. libgcc-13.2.0

When compiling dynamically linked software using GCC, GCC requires that libgcc be able to be loaded during runtime when executing the software. Hence, we must be sure to provide the final system with a copy of the libgcc we previously built for our cross tools.

5.4.1. Installation of libgcc

Install the package:

```
cp -v ${CLFS}/cross-tools/${CLFS_TARGET}/lib/libgcc_s.so.1 ${CLFS}/targetfs/lib/
```

Strip libgcc to make it smaller:

```
${CLFS_TARGET}-strip ${CLFS}/targetfs/lib/libgcc_s.so.1
```

5.5. musl-1.2.5

The musl 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. We build it again here so that it can be installed into our targetfs sysroot but this time we only build the shared object version.

5.5.1. Installation of musl

Configure the package:

```
./configure \
  CROSS_COMPILE=${CLFS_TARGET}- \
  --prefix=/ \
  --disable-static \
  --target=${CLFS_TARGET}
```

Compile the package:

```
make
```

Install just the shared library:

```
DESTDIR=${CLFS}/targetfs make install-libs
```

Details on this package are located in Section 4.8.2, “Contents of musl.”

5.6. BusyBox-1.36.1

BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc. The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts. BusyBox provides a fairly complete environment for any small or embedded system.

5.6.1. Installation of BusyBox

First ensure the BusyBox source is completely clean:

```
make distclean
```

> **Note**
>
> We tell BusyBox to use the generic defconfig. For those who are more adventurous, you can use **make menuconfig**, and create a custom or modified configuration for your build.

The following tells BusyBox to use the default configuration:

```
make ARCH="${CLFS_ARCH}" defconfig
```

Disable the use of utmp/wtmp as musl does not support them:

```
sed -i 's/\(CONFIG_FEATURE_WTMP\)=y/# \1 is not set/' .config
sed -i 's/\(CONFIG_FEATURE_UTMP\)=y/# \1 is not set/' .config
```

Compile the package:

```
make ARCH="${CLFS_ARCH}" CROSS_COMPILE="${CLFS_TARGET}-"
```

Install the package:

```
make ARCH="${CLFS_ARCH}" CROSS_COMPILE="${CLFS_TARGET}-"\
  CONFIG_PREFIX="${CLFS}/targetfs" install
ln -sv bin/busybox ${CLFS}/targetfs/init
```

If you're going to build your kernel with modules, you will need to make sure **depmod.pl** is available for your host to execute:

```
cp -v examples/depmod.pl ${CLFS}/cross-tools/bin
chmod -v 755 ${CLFS}/cross-tools/bin/depmod.pl
```

5.6.2. Contents of BusyBox

Installed programs: To be Written

5.7. iana-etc-20240125

The iana-etc package provides data for network services and protocols.

5.7.1. Installation of Iana-Etc

For this package, we only need to copy the files into place:

```
cp services protocols ${CLFS}/targetfs/etc
```

5.7.2. 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

Chapter 6. Making the CLFS System Bootable

6.1. Introduction

It is time to make the CLFS system bootable. This chapter discusses creating an `fstab` file, building a kernel for the new CLFS system, and installing the boot loader so that the CLFS system can be selected for booting at startup.

6.2. 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. For our embedded system, the bootloader will tell Linux where to find the root file system and we will not mount any additional file systems, so we can create an empty file system table like this:

```
cat > ${CLFS}/targetfs/etc/fstab << "EOF"
# file-system  mount-point  type   options          dump  fsck
EOF
```

You may add any other file systems, such as swap or other partitions or network file systems, to this fstab if you wish.

6.3. Linux-6.7.4

The Linux package contains the Linux kernel.

6.3.1. 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.

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.

A good starting place for setting up the kernel configuration is to use a *defconfig*. This will set the base configuration to a good state that takes your target system architecture into account.

To list the all available default configuratinons available use this command:

```
ARCH=${CLFS_ARCH} make help | grep defconfig
```

Then run **make [your default configuration]** to generate the `.config` file.

Configure the kernel via a menu-driven interface. Be sure to enable DEVTMPFS so that `/dev` will be populated automatically.

> **Note**
>
> Since you are building for an embedded system make sure all key components are built into the kernel and not as modules. The key components are console/video, disk, and network. Without these built in, the system will not function properly. It is recommended to configure the kernel without modules in order to conserve disk space and to reduce the complexity of booting.

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

Alternatively, the **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 an example system (assuming it is available) to the root directory of the unpacked kernel sources.

> **Note**
>
> If you are using the u-boot bootloader, creating a `uImage` type kernel is recommended. Install the u-boot tools (not documented here) and append "uImage" when compiling the kernel.

Compile the configured kernel image and modules:

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

If using kernel modules, an `/etc/modprobe.conf` file may be needed. Information pertaining to modules and kernel configuration is located in the kernel documentation in the `Documentation` directory of the kernel sources tree. Also, `modprobe.conf(5)` may be of interest.

If building modules, install the modules:

```
make ARCH=${CLFS_ARCH} CROSS_COMPILE=${CLFS_TARGET}- \
    INSTALL_MOD_PATH=${CLFS}/targetfs modules_install
```

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.

The resulting kernel will be located within the `arch/${CLFS_ARCH}/boot`. There may be more than one version of the same kernel, simply with different compression or bootloader helpers added. Follow your bootloader's instructions on how to copy the kernel to the final system.

6.3.2. Contents of Linux

Installed files: .config, Image files, and System.map

Short Descriptions

`.config`	Contains all the configuration selections for the kernel
`zImage, uImage, bzImage, vmlinux`	The compiled Linux kernel.
`System.map`	A list of addresses and symbols; it maps the entry points and addresses of all the functions and data structures in the kernel. Useful for debugging, sometimes.

6.4. Bootloaders

There are many different bootloaders. Creating detailed instructions on how to use even the more popular ones is difficult as each architecture and board often has different levels of support or configuration options within each bootloader. Often, development kits will ship with a bootloader already configured, if your board comes with a bootloader pre-configured, it's best to start using that and then venture on to building the bootloader yourself later.

A few of the popular bootloaders include:

Barebox
http://barebox.org/

CoLo
http://www.colonel-panic.org/cobalt-mips/

GNU GRUB
https://www.gnu.org/software/grub/

Syslinux
http://www.syslinux.org/wiki/index.php/The_Syslinux_Project

u-boot
http://www.denx.de/wiki/U-Boot

Chapter 7. Setting Up System Bootscripts

7.1. Introduction

This chapter details how to install and configure the CLFS-Bootscripts package. Most of these scripts will work without modification, but a few require additional configuration files because they deal with hardware-dependent information.

System-V style init scripts are employed in this book because they are widely used. For additional options, a hint detailing the BSD style init setup is available at *http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt*. Searching the LFS mailing lists for “depinit” will also offer additional choices.

If using an alternative style of init scripts, skip this chapter and move on to Making the CLFS System Bootable.

7.2. CLFS-Bootscripts-git master HEAD

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

7.2.1. Installation of CLFS-Bootscripts

> **Important**
>
> Unlike other chapters, after installing the bootscripts, do not delete the bootscripts source directory. The bootscripts source directory may be needed later.

Install the package:

```
make DESTDIR=${CLFS}/targetfs install-bootscripts
```

7.2.2. Contents of CLFS-Bootscripts

Installed scripts: functions, startup, shutdown, and syslog.

Short Descriptions

functions	Contains common functions, such as error and status checking, that are used by several bootscripts
startup	Performs all startup script operations
shutdown	Performs all shutdown operations
syslog	Starts and stops the system log daemons

7.3. Configure mdev

Is a BusyBox replacement of udev. With a different rule base.

7.3.1. Creating /etc/mdev.conf

Now we will create the mdev.conf file for use with our system:

```
cat > ${CLFS}/targetfs/etc/mdev.conf<< "EOF"
# /etc/mdev/conf

# Devices:
# Syntax: %s %d:%d %s
# devices user:group mode

# null does already exist; therefore ownership has to be changed with command
null root:root 0666 @chmod 666 $MDEV
zero root:root 0666
grsec root:root 0660
full root:root 0666

random root:root 0666
urandom root:root 0444
hwrandom root:root 0660

# console does already exist; therefore ownership has to be changed with command
#console root:tty 0600 @chmod 600 $MDEV && mkdir -p vc && ln -sf ../$MDEV vc/0
console root:tty 0600 @mkdir -pm 755 fd && cd fd && for x in 0 1 2 3 ; do ln -sf

fd0 root:floppy 0660
kmem root:root 0640
mem root:root 0640
port root:root 0640
ptmx root:tty 0666

# ram.*
ram([0-9]*) root:disk 0660 >rd/%1
loop([0-9]+) root:disk 0660 >loop/%1
sd[a-z].* root:disk 0660 */lib/mdev/usbdisk_link
hd[a-z][0-9]* root:disk 0660 */lib/mdev/ide_links
md[0-9]  root:disk 0660

tty  root:tty 0666
tty[0-9] root:root 0600
tty[0-9][0-9] root:tty 0660
ttyS[0-9]* root:tty 0660
pty.*  root:tty 0660
vcs[0-9]* root:tty 0660
```

```
vcsa[0-9]* root:tty 0660

ttyLTM[0-9] root:dialout 0660 @ln -sf $MDEV modem
ttySHSF[0-9] root:dialout 0660 @ln -sf $MDEV modem
slamr  root:dialout 0660 @ln -sf $MDEV slamr0
slusb  root:dialout 0660 @ln -sf $MDEV slusb0
fuse  root:root  0666

# dri device
card[0-9] root:video 0660 =dri/

# alsa sound devices and audio stuff
pcm.*  root:audio 0660 =snd/
control.* root:audio 0660 =snd/
midi.*  root:audio 0660 =snd/
seq  root:audio 0660 =snd/
timer  root:audio 0660 =snd/

adsp  root:audio 0660 >sound/
audio  root:audio 0660 >sound/
dsp  root:audio 0660 >sound/
mixer  root:audio 0660 >sound/
sequencer.* root:audio 0660 >sound/

# misc stuff
agpgart  root:root 0660  >misc/
psaux  root:root 0660  >misc/
rtc  root:root 0664  >misc/

# input stuff
event[0-9]+ root:root 0640 =input/
mice  root:root 0640 =input/
mouse[0-9] root:root 0640 =input/
ts[0-9]  root:root 0600 =input/

# v4l stuff
vbi[0-9] root:video 0660 >v4l/
video[0-9] root:video 0660 >v4l/

# dvb stuff
dvb.*  root:video 0660 */lib/mdev/dvbdev

# load drivers for usb devices
usbdev[0-9].[0-9] root:root 0660 */lib/mdev/usbdev
usbdev[0-9].[0-9]_.* root:root 0660

# net devices
```

```
tun[0-9]* root:root 0600 =net/
tap[0-9]* root:root 0600 =net/

# zaptel devices
zap(.*)  root:dialout 0660 =zap/%1
dahdi!(.*) root:dialout 0660 =dahdi/%1

# raid controllers
cciss!(.*) root:disk 0660 =cciss/%1
ida!(.*) root:disk 0660 =ida/%1
rd!(.*)  root:disk 0660 =rd/%1

sr[0-9]  root:cdrom 0660 @ln -sf $MDEV cdrom

# hpilo
hpilo!(.*) root:root 0660 =hpilo/%1

# xen stuff
xvd[a-z] root:root 0660 */lib/mdev/xvd_links
EOF
```

7.4. Creating /etc/profile

Is the file that specifices how your environment will function.

Now we will create the profile file for use with our system:

```
cat > ${CLFS}/targetfs/etc/profile << "EOF"
# /etc/profile

# Set the initial path
export PATH=/bin:/usr/bin

if [ `id -u` -eq 0 ] ; then
        PATH=/bin:/sbin:/usr/bin:/usr/sbin
        unset HISTFILE
fi

# Setup some environment variables.
export USER=`id -un`
export LOGNAME=$USER
export HOSTNAME=`/bin/hostname`
export HISTSIZE=1000
export HISTFILESIZE=1000
export PAGER='/bin/more '
export EDITOR='/bin/vi'

# End /etc/profile
EOF
```

7.5. Creating /etc/inittab

Is the file that specifices how to boot and shutdown a system.

Now we will create the inittab file for use with our system, please note that if you'd like a login prompt on a serial console to uncomment enabling of this and verify the serial port device name as many embedded boards have serial ports which are not named 'ttyS':

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

::sysinit:/etc/rc.d/startup

tty1::respawn:/sbin/getty 38400 tty1
tty2::respawn:/sbin/getty 38400 tty2
tty3::respawn:/sbin/getty 38400 tty3
tty4::respawn:/sbin/getty 38400 tty4
tty5::respawn:/sbin/getty 38400 tty5
tty6::respawn:/sbin/getty 38400 tty6

# Put a getty on the serial line (for a terminal).  Uncomment this line if
# you're using a serial console on ttyS0, or uncomment and adjust it if using a
# serial console on a different serial port.
#::respawn:/sbin/getty -L ttyS0 115200 vt100

::shutdown:/etc/rc.d/shutdown
::ctrlaltdel:/sbin/reboot
EOF
```

7.6. Setting Hostname

Part of the job of the bootscripts is setting the system's hostname. This needs to be configured in the `/etc/HOSTNAME` file.

Create the `HOSTNAME` file and enter a hostname by running:

```
echo "[clfs]" > ${CLFS}/targetfs/etc/HOSTNAME
```

`[clfs]` needs to be replaced with the name given to the computer. Do not enter the Fully Qualified Domain Name (FQDN) here. That information will be put in the `/etc/hosts` file in the next section.

7.7. Customizing the /etc/hosts File

If a network card is to be configured, decide on the IP address, FQDN, and possible aliases for use in the `/etc/hosts` file. The syntax is:

```
<IP address> myhost.example.org aliases
```

Unless the computer is to be visible to the Internet (i.e., there is a registered domain and a valid block of assigned IP addresses—most users do not have this), make sure that the IP address is in the private network IP address range. Valid ranges are:

```
  Class Networks
  A     10.0.0.0
  B     172.16.0.0 through 172.31.0.255
  C     192.168.0.0 through 192.168.255.255
```

A valid IP address could be 192.168.1.1. A valid FQDN for this IP could be `www.linuxfromscratch.org` (not recommended because this is a valid registered domain address and could cause domain name server issues).

Even if not using a network card, an FQDN is still required. This is necessary for certain programs to operate correctly.

Create the `/etc/hosts` file by running:

```
cat > ${CLFS}/targetfs/etc/hosts << "EOF"
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
[192.168.1.1] [<HOSTNAME>.example.org] [HOSTNAME]

# End /etc/hosts (network card version)
EOF
```

The *`[192.168.1.1]`* and *`[<HOSTNAME>.example.org]`* values need to be changed for specific users or requirements (if assigned an IP address by a network/system administrator and the machine will be connected to an existing network).

If a network card is not going to be configured, create the `/etc/hosts` file by running:

```
cat > ${CLFS}/targetfs/etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)

127.0.0.1 [<HOSTNAME>.example.org] [HOSTNAME] localhost

# End /etc/hosts (no network card version)
EOF
```

7.8. Configuring the network Script

7.8.1. Creating Network Interface Configuration Files

Which interfaces are brought up and down by the network utilities depends on the files and directories in the `/etc/network` directory. The `interfaces` file should contain a description of the interfaces, as done by Debian, and each of the directories contain scripts for actions to perform on each type of network event.

The following command creates the required directories and the `interfaces` file, assuming DHCP will be used for eth0:

```
mkdir -pv ${CLFS}/targetfs/etc/network/if-{post-{up,down},pre-{up,down},up,down}.
mkdir -pv ${CLFS}/targetfs/usr/share/udhcpc

cat > ${CLFS}/targetfs/etc/network/interfaces << "EOF"
auto eth0
iface eth0 inet dhcp
EOF
```

For DHCP to work properly with udhcpc, the BusyBox dhcp client, a configuration script is needed. Create a simple script to assign the provided DHCP address and update `/etc/resolv.conf`:

```
cat > ${CLFS}/targetfs/usr/share/udhcpc/default.script << "EOF"
#!/bin/sh
# udhcpc Interface Configuration
# Based on http://lists.debian.org/debian-boot/2002/11/msg00500.html
# udhcpc script edited by Tim Riker <Tim@Rikers.org>

[ -z "$1" ] && echo "Error: should be called from udhcpc" && exit 1

RESOLV_CONF="/etc/resolv.conf"
[ -n "$broadcast" ] && BROADCAST="broadcast $broadcast"
[ -n "$subnet" ] && NETMASK="netmask $subnet"

case "$1" in
 deconfig)
  /sbin/ifconfig $interface 0.0.0.0
  ;;

 renew|bound)
  /sbin/ifconfig $interface $ip $BROADCAST $NETMASK

  if [ -n "$router" ] ; then
   while route del default gw 0.0.0.0 dev $interface ; do
    true
   done

   for i in $router ; do
    route add default gw $i dev $interface
   done
  fi

  echo -n > $RESOLV_CONF
  [ -n "$domain" ] && echo search $domain >> $RESOLV_CONF
  for i in $dns ; do
   echo nameserver $i >> $RESOLV_CONF
  done
  ;;
esac

exit 0
EOF

chmod +x ${CLFS}/targetfs/usr/share/udhcpc/default.script
```

Part V. Beyond CLFS Embedded

Chapter 8. Beyond CLFS Embedded

8.1. Introduction

Now that we have our base system built, we may add some optional additional networking software.

8.2. Beyond CLFS Packages

Download or otherwise obtain the following packages:

Dropbear (2022.83) - 2268 KB:
Home page: *http://matt.ucc.asn.au/dropbear/dropbear.html*
Download: *http://matt.ucc.asn.au/dropbear/releases/dropbear-2022.83.tar.bz2*
MD5 sum: `a75a34bcc03cacf71a2db9da3b7c94a5`

Netplug (1.2.9.2) - 21 KB:
Home page: *http://www.red-bean.com/~bos/*
Download: *http://www.red-bean.com/~bos/netplug/netplug-1.2.9.2.tar.bz2*
MD5 sum: `1d6db99536bdf875ce441f2c0e45ebf2`

Wireless Tools (29) - 288 KB:
Home page: *https://hewlettpackard.github.io/wireless-tools/Tools.html*
Download: *https://hewlettpackard.github.io/wireless-tools/wireless_tools.29.tar.gz*
MD5 sum: `e06c222e186f7cc013fd272d023710cb`

Zlib (1.3.1) - 1478 KB:
Home page: *http://www.zlib.net*
Download: *https://zlib.net/fossils/zlib-1.3.1.tar.gz*
MD5 sum: `9855b6d802d7fe5b7bd5b196a2271655`

Total size of these packages: about 4,055 KB

8.3. Beyond CLFS 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 in order to build the associated packages:

Netplug Patch - 1 KB:
Download: *http://patches.clfs.org/embedded-dev/netplug-1.2.9.2-fixes-1.patch*
MD5 sum:

Total size of these patches: about 1 KB

8.4. Zlib-1.3.1

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

8.4.1. Installation of Zlib

Prepare Zlib for compilation and set to optimize for size:

```
CFLAGS="-Os" ./configure --shared
```

The meaning of the configure options:

`--shared`
Tells Zlib to build its shared library.

Compile the package:

```
make
```

Install the package into the cross-tools:

```
make prefix=${CLFS}/cross-tools/${CLFS_TARGET} install
```

Copy only the shared library into the target file system and ensure that its symlink is present so the runtime loader can find it:

```
cp -v ${CLFS}/cross-tools/${CLFS_TARGET}/lib/libz.so.1.3.1 ${CLFS}/targetfs/lib/
ln -sv libz.so.1.3.1 ${CLFS}/targetfs/lib/libz.so.1
```

8.4.2. Contents of Zlib

Installed libraries: libz.[a,so]

Short Descriptions

`libz` Contains compression and decompression functions used by some programs

8.5. Netplug-1.2.9.2

Netplug is a daemon that detects insertion and removal of network cables and reacts to bring up or take down the network interface.

8.5.1. Installation of Netplug

Patch netplug to fix issues:

```
patch -Np1 -i ../netplug-1.2.9.2-fixes-1.patch
```

Compile the package:

```
make
```

Install the package:

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

8.5.2. Installation of Netplug Bootscripts

From the clfs-bootscripts package, install the Netplug bootscripts:

```
make install-netplug DESTDIR=${CLFS}/targetfs
```

8.5.3. Contents of Netplug

Installed programs: /sbin/netplugd

8.6. Dropbear-2022.83

Dropbear is a relatively small SSH server and client. Dropbear has a small memory footprint suitable for memory-constrained environments, while still having the same features as OpenSSH. It does not depend on OpenSSL and it has a MIT style license.

Dropbear depends on zlib.

8.6.1. Installation of Dropbear

Fix dropbear so it doesn't install man pages:

```
sed -i 's/.*mandir.*//g' Makefile.in
```

Configure dropbear:

```
CC="${CC} -Os" ./configure --prefix=/usr --host=${CLFS_TARGET}
```

> **Note**
>
> We are just telling dropbear to use the default configuration plus scp. For those who are more adventurous, edit options.h to further configure dropbear.

> **Important**
>
> If you get a compile error regarding an *undefined reference to `__stack_chk_fail_local'* run **`make distclean`** and add **`--disable-harden`** to the configure command before proceding.

Compile the package:

```
make MULTI=1 \
  PROGRAMS="dropbear dbclient dropbearkey dropbearconvert scp"
```

Install the package:

```
make MULTI=1 \
  PROGRAMS="dropbear dbclient dropbearkey dropbearconvert scp" \
  install DESTDIR=${CLFS}/targetfs
```

Create the directory for the dropbear key files:

```
install -dv ${CLFS}/targetfs/etc/dropbear
```

8.6.2. Installation of Dropbear Bootscripts

From the clfs-bootscripts package, install the Dropbear bootscripts:

```
make install-dropbear DESTDIR=${CLFS}/targetfs
```

8.6.3. Contents of Dropbear

Installed programs: /usr/bin/dropbearmulti

8.7. Wireless Tools-29

Wireless Tools is the reference implementation of tools supporting the ability to manipulate the Wireless Extensions API supported by most wireless LAN networking drivers.

8.7.1. Installation of Wireless Tools

Wireless Tools' `Makefile` contains explicitly which **gcc**, **ar**, and **ranlib** to use. Make sure the `${CLFS}/cross-tools` versions of these programs are used instead of the host's:

```
sed -i s/gcc/\$\{CLFS\_TARGET\}\-gcc/g Makefile
sed -i s/\ ar/\ \$\{CLFS\_TARGET\}\-ar/g Makefile
sed -i s/ranlib/\$\{CLFS\_TARGET\}\-ranlib/g Makefile
```

Compile the package:

> **Note**
>
> There are options that can be passed to **make** and **make install** that will reduce the size and functionality of the Wireless Tools. See the Wireless Tools `INSTALL` file for more information.

```
make PREFIX=${CLFS}/targetfs/usr
```

Install the package:

```
make install PREFIX=${CLFS}/targetfs/usr
```

8.7.2. Contents of Wireless Tools

Installed programs: iwconfig, iwevent, iwgetid, iwlist, iwpriv, and iwspy
Installed libraries: libiw.so

Part VI. Cleanup and Boot

Chapter 9. Backup and Cleanup

9.1. Changing the Ownership of the CLFS System

Throughout the book, every package has been compiled and installed as the `clfs` user. The final system should be owned by `root`.

> **Important**
>
> The commands on this page of the book must be performed while logged in as `root`. Check that ${CLFS} is still set:
>
> ```
> echo ${CLFS}
> ```

Make `root` the owner of the entire CLFS system:

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

The following file should not to belong to the `root` group, it should belong to the `utmp` group (group 13):

```
chgrp -v 13 ${CLFS}/targetfs/var/log/lastlog
```

9.2. Copy to Target

We just created a cleaned-up version of our build, now compress it so it can be archived and transferred to the target.

Create a tarball of the build:

```
install -dv ${CLFS}/build
cd ${CLFS}/targetfs
tar jcfv ${CLFS}/build/clfs-embedded.tar.bz2 *
```

Now you can move your compressed tarball to your target system. This will be different on every embedded device.

When uncompressing the tarball, make sure to pass tar the "-p" switch to ensure permissions are preserved.

Chapter 10. The End

10.1. The End

Well done! The new CLFS system is installed! We wish you much success with your shiny new custom-built Linux system.

Assuming the boot loader was set up properly, *CLFS GIT-20240401* will boot automatically when you power on the target.

10.2. What Now?

Thank you for reading this CLFS book. We hope that you have found this book helpful and have learned more about the system creation process.

Now that the CLFS system is installed, you may be wondering “What next?” To answer that question, we have compiled a list of resources for you.

- Maintenance

  Bugs and security notices are reported regularly for all software. Since an CLFS system is compiled from source, it is up to you to keep abreast of such reports. There are several online resources that track such reports, some of which are shown below:

  - *CERT* (Computer Emergency Response Team)

    CERT has a mailing list that publishes security alerts concerning various operating systems and applications. Subscription information is available at *http://www.us-cert.gov/cas/signup.html*.

  - Bugtraq

    Bugtraq is a full-disclosure computer security mailing list. It publishes newly discovered security issues, and occasionally potential fixes for them. Subscription information is available at *http://www.securityfocus.com/archive*.

- Community Driven Beyond Linux From Scratch

  The Community Driven Beyond Linux From Scratch wiki provides information, installation procedures, and other tips for a large variety of software that can be installed on a CLFS system. The CBLFS wiki is located at *http://cblfs.clfs.org/*.

- Beyond Linux From Scratch

  The Beyond Linux From Scratch book covers installation procedures for a wide range of software beyond the scope of the LFS Book. The BLFS project is located at *http://www.linuxfromscratch.org/blfs/*.

- LFS Hints

  The LFS Hints are a collection of educational documents submitted by volunteers in the LFS community. The hints are available at *http://www.linuxfromscratch.org/hints/list.html*.

- Mailing lists

  There are several LFS mailing lists you may subscribe to if you are in need of help, want to stay current with the latest developments, want to contribute to the project, and more. See Chapter 1 - Mailing Lists for more information.

- The Linux Documentation Project

  The goal of The Linux Documentation Project (TLDP) is to collaborate on all of the issues of Linux documentation. The TLDP features a large collection of HOWTOs, guides, and man pages. It is located at *http://www.tldp.org/*.

Index

Packages

Programs

Libraries

Scripts

Others