Linux From Scratch

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Linux From Scratch

Gerard Beekmans − Main document

Michael Peters − Apple PowerPC additions

This book describes the process of creating your own Linux system from scratch from an already installed Linux distribution, using nothing but the sources of software that are needed.

Dedication

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

Table of Contents

Preface

Who would want to read this book

This book is intended for Linux users who want to learn more about the inner workings of Linux and how the various pieces of the Operating System fit together. This book will guide you step−by−step in creating your own custom build Linux system from scratch, using nothing but the sources of software that are needed.

This book is also intended for Linux users who want to get away from the existing commercial and free distributions that are often too bloated. Using existing distributions also forces you to use the file system structure, boot script structure, etc. that they choose to use. With this book you can create your own structures and methods in exactly the way you like them (which can be based on the ones this book provides)

Also, if you have security concerns, you don't want to rely on pre−compiled packages. So instead, you want to compile all programs yourself and install them. That could be another reason why you would want to build a custom made Linux system.

For those and numerous other reasons somebody might want to build his or her own Linux system from the ground up. If you are one of those people, this book is meant for you.

Who would not want to read this book

Users who don't want to build an entire Linux system from scratch probably don't want to read this book. If you, however, do want to learn more about what happens behind the scenes, in particulair what happens between turning on your computer and seeing the command prompt, you want to read the "From Power Up To Bash Prompt" (P2B) HOWTO. This HOWTO builds a bare system, in a similar way as this book does, but it focusses more on just installing a bootable system instead of a complete system.

To decide whether you want to read this book or the P2B HOWTO, you could ask yourself this question: Is my main objective to get a working Linux system that I'm going to build myself and along the way learn and learn what every component of a system is for, or is just the learning part your main objective. If you want to build and learn, read this book. If you just want to learn, then the P2B HOWTO is probably better material to read.

The "From Power Up To Bash Prompt" HOWTO can be downloaded from http://learning.taslug.org.au/power2bash

Organization

This book is devided into the following parts. Although there is a lot of duplicate information in certain parts, it's the easiest way to read it and not to mention the easiest way for me to maintain the book.

Part I – Introduction

Part One gives you general information about this book (versions, where to get it, changelog, mailinglists and how to get in touch with me). It also explains a few important aspects you really want and need to read before you start building an LFS system.

Part II – Installation of a basic system on Intel systems

Part Two guides you through the installation of a basic system on Intel systems which will be the foundation for the rest of the system. Whatever you choose to do with your brand new LFS system, it will be built on the foundation that's installed in this part.

Part III – Installation of a basic system on Apple PowerPC systems

Part Three is the Apple PowerPC version of part two.

Part IV – Appendixes

Part Four contains various Appendixes.

I. Part I – Introduction

Table of Contents

Chapter 1. Introduction

Introduction

Having used a number of different Linux distributions, I was never fully satisfied with any of those. I didn't like the way the bootscripts were arranged, or I didn't like the way certain programs were configured by default and more of those things. I came to realize that when I want to be totally satisfied with a Linux system, I have to build my own Linux system from scratch, ideally only using the source code. Not using pre−compiled packages of any kind. No help from some sort of cdrom or bootdisk that would install some basic utilities. You would use your current Linux system and use that one to build your own.

This, at one time, wild idea seemed very difficult and at times almost impossible. The reason for most problems were due to my lack of knowledge about certain programs and procedures. After sorting out all kinds of dependency problems, compilation problems, etcetera, a custom built Linux system was created and fully operational. I called this system an LFS system, which stands for LinuxFromScratch.

How things are going to be done

We are going to build the LFS system using an already installed Linux distribution such as Debian, SuSe, Slackware, Mandrake, RedHat, etc. You don't need to have any kind of bootdisk. We will use an existing Linux system as the base (since we need a compiler, linker, text editor and other tools).

If you don't have Linux installed yet, you won't be able to put this book to use right away. I suggest you first install a Linux distribution. It really doesn't matter which one you install. It also doesn't need to be the latest version, though it shouldn't be a too old one. If it is about a year old or newer it should do just fine. You will safe yourself a lot of trouble if your normal system uses glibc−2.0 or newer. Libc5 isn't supported by this book, though it isn't impossible to use a libc5 system if you have no choice.

There are a few sub−LFS projects running and one of them handles installing LFS using a bootdisk. Using the bookdisk there will be no need for an already installed Linux system. This project is still under development and therefore it's directions not yet included in this book.

Book versions

This is the 2.3.4 development version dated June 5th, 2000. If this version is older than a month you definitely want to take a look at our website and download a newer version.

The latest versions of this book and related files can be downloaded from one of the following sites. Please avoid the main site at Dallas whenever possible. Thanks.

- Dallas, Texas, United States – http://www.linuxfromscratch.org/index2.html
- Columbus, Ohio, United States – http://lfs.bcpub.com/index2.html
- United States – http://clueserver.org/lfs/index2.html
- United States – http://lfs.sourceforge.net/index2.html
- Braunschweig, Niedersachsen, Germany – http://134.169.139.209/index2.html
- Brisbane, Queensland, Australia – http://lfs.mirror.aarnet.edu.au/index2.html

Acknowledgements

I would like to thank the following people and organizations for their contributions towards the LinuxFromScratch project:

- Paul Jensen for providing http://www.pcrdallas.com as the main linuxfromscratch.org host
- Bryan Dumm for providing http://www.bcpub.com as the lfs.bcpub.com mirror
- Alan Olsen for providing http://clueserver.org as the clueserver.org/lfs mirror
- Jan Niemann for providing http://helga.lk.etc.tu−bs.de as the 134.169.139.209 mirror
- Jason Andrade for providing http://mirror.aarnet.edu.au as the lfs.mirror.aarnet.edu.au mirror
- VA Linux Systems who on behalf of Linux.com donated a VA Linux 420 (formerly StartX SP2) workstation towards this project

Changelog

j.3.4 – June 5th, 2000

- Chapter 5: Fixed the kernel header files configuration

- Chapter 5: Fixed the lilo configuration

j.3.3 – May 15th, 2000

- Changed the default mount point from /mnt/xxx to /mnt/lfs (where xxx used to be the partition's designation like hda5, sda5 and others). The reason for the change is to make cross−platform instructions easier.

- Chapter 4: Changed the default modes for the $LFS/root and $LFS/tmp directory to respectively 0750 and 1777.

- Chapter 5: Removed the encoded password from the passwd file. Instead a file with no set password is created. The root password can be set by the user when the system is rebooted into the LFS system (after chapter 8).

- Chapter 5: Fixed the procps compile command for watch.c. It should compile properly now.

- Chapter 5: Fixed gzip patch installation (used the wrong filename in the patch command

- Chapter 5: Changed 'entering the chroot'ed environment' to make bash a login shell.

- Chapter 5: Configuring the kernel has been moved to this chapter because it needs to be done before programs like e2fsprogs and lilo are compiled.

- Chapter 6: Fixed the rc script. It now checks to see if the previous run level starts a service before attempting to stop it in the new run level. Also, if a service is already started in the previous run level it won't attempt to start the service in the new run level again. Thanks to Jason Pearce for providing this fixed script.

- Chapter 7: Fixed the ethnet script − removed paratheses from the environment variables and removed the command to add a route. The ifconfig command used to bring the eth device up already sets this route.

j.3.2 − April 18th, 2000

- Chapter 4.7: Change only the owner of the $LFS/dev/* files
- Fixed a large amount of typo's that occured during the transistion from the LinuxDoc DTD (2.2 and lower) to the DocBook DTD (2.3.1 and higher).
- Moved chapters around quite a bit and applied a new structure in the book. Installations for Intel, Apple PowerPC and future systems will be put in their own dedicated part of the book.
- After the system is prepared to install the basic system software, we no longer reboot the system but instead we setup a chroot'ed environment. This will have the same effect without having to reboot.
- Apple PowerPC has it's own dedicated chapters now. This should increase readability a lot
- All optional chapters have been removed for now. These chapters are going to be restructured into dedicated parts such as a chapter that deals with setting up LFS as an email server. A chapter that deals with setting up LFS as a http server, and so forth. These reorganizations couldn't make this development version in time. So you'll have to read the current stable 2.2 version of this book for those parts.
- Replaced the fixed packages by patch files. This way you can see what needs to be changed in a package in order to get it to compile properly.

j.3.1 − April 12th, 2000

- Chapter 4.4: Added the $LFS/usr/info symlink which points to $LFS/usr/share/info
- Chapter 7.3.1: Added a second variation to a 'swap−line' in a fstab file.
- Chapter 7.3.2: Removed $LFS from the commands.
- Chapter 7.4.43: Added the vi symlink
- Chapter 9.2.5: Improved ethnet script to include routing information
- Chapter 10.1.2: Fixed missing subdirectory 'mqueue' in mkdir /var/spool −> /mkdir /var/spool/mqueue
-

Chapter 10.1.4: Updated the sendmail configuration file with a few necessary options

- Chapter 10.1.7: Fixed wrong directory path /etc/init.d/rc2.d -> /etc/rc2.d

Mailinglists and archives

The linuxfromscratch.org server is hosting the following three public accessible mailinglists:

- lfs−discuss
- lfs−config
- lfs−apps
- lfs−announce
- linux

---

lfs−discuss

The lfs−discuss list is the list that discusses matters regarding this book. If you have problems, comments, suggestions, etc. join this list and post your message. People on this list can take part in the newest developments regarding this book.

---

lfs−config

A more detailed description of this list will be written in the next version of this book

The lfs−config list discusses the configuration issues related to programs that are used in the book.

---

lfs−apps

A more detailed description of this list will be written in the next version of this book

The lfs−apps list discussed the compilation and configuration of software that's not used in this book. The list is mainly used when you have problems installing software on an LFS system when you don't have problems compiling it on your normal distribution. It's not that LFS is incompatible with "normal" distributions but just the fact that you might be missing support−software that programs need or need to configure a few things on your new LFS system. The people on this list can help you with those issues.

---

lfs−announce

The lfs−announce list is a moderated list. You can subscribe to it, but you can't post any messages to this list. This list is used to announce new stable releases. If you want to be informed about development releases as well then you'll have to join the lfs−discuss list. If you're already on the lfs−discuss list there's little use subscribing to this list as well because everything that is posted to the lfs−announce list will be posted to the lfs−discuss list as well.

---

linux

The linux list is a general Linux discussion list that handles everything that has got anything to do with Linux in any way, shape and form.

How to subscribe?

You can subscribe to any of the above mentioned mailinglists by sending an email to majordomo@linuxfromscratch.org and write *subscribe listname* in the body of the message, where listname is replaced by either lfs−discuss, lfs−config, lfs−apps, lfs−announce or linux. No subject required.

You can, if you want, subscribe to multiple lists at the same time using one email. Just repeat the subscribe command for each of the lists you want to subscribe to.

After you have sent the email, the Majordomo program will send you an email back requesting a confirmation of your subscription request. After you have sent back this confirmation email, Majordomo will send you an email again with the message that you have been subscribed to the list(s) along with an introduction message for that particulair list.

How to unsubscribe?

To unsubscribe from a list, send an email to majordomo@linuxfromscratch.org and write *unsubscribe listname* in the body of the message, where listname is replaced by either lfs−discuss, lfs−config, lfs−apps, lfs−announce or linux.

You can, if you want, unsubscribe from multiple lists at the same time using one email. Just repeat the unsubscribe command for each of the lists you want to unsubscribe from.

Mail archives

The lfs−discuss, lfs−config, lfs−apps and linux mailing lists have an archive you can access to find information on subjects already posted to this list. You can find them at http://www.pcrdallas.com/mail−archives

---

Contact information

Direct all your emails to the lfs−discuss mailinglist preferably.

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

If you need to reach Michael Peters personaly, send an email to mpters@mac.com

Chapter 2. Important information

About $LFS

Please read the following carefully: throughout this document you will frequently see the variable name $LFS. $LFS must at all times be replaced by the directory where the partition that contains the LFS system is mounted. How to create and where to mount the partition will be explained later on in full detail in chapter 4. In my case the LFS partition is mounted on /mnt/lfs. If I read this document myself and I see $LFS somewhere, I will pretend that I read /mnt/lfs. If I read that I have to run this command: cp inittab $LFS/etc I actually will run this: cp inittab /mnt/lfs/etc

It's important that you do this no matter where you read it; be it in commands you enter on the prompt, or in some file you edit or create.

If you want, you can set the environment variable LFS. This way you can literally enter $LFS instead of replacing it by something like /mnt/lfs. This is accomplished by running: export LFS=/mnt/lfs

If I read cp inittab $LFS/etc, I literally can type cp inittab $LFS/etc and the shell will replace this command by cp inittab /mnt/lfs/etc automatically.

Do not forget to set the $LFS variable at all times. If you haven't set the variable and you use it in a command, $LFS will be ignored and whatever i left will be executed. The command cp inittab $LFS/etc without the LFS variable set, will result in copying the inittab file to the /etc directory which will overwrite your system's inittab. A file like inittab isn't that big a problem as it can easily be restored, but if you would make this mistake during the installation of the C Library, you can break your system badly and might have to reinstall it if you don't know how to repair it. So that's why I strongly advise against using the $LFS variable. You better replace $LFS yourself by something like /mnt/lfs. If you make a typo while entering /mnt/lfs, the worst thing that can happen is that you'll get an error saying "no such file or directory" but it won't break your system. Don't say I didn't warn you ;)

How to download the software

Throughout this document I will assume that you have stored all the packages you have downloaded in a subdirectory under $LFS/usr/src.

I use the convention of having a $LFS/usr/src/sources directory. Under sources you'll find the directory 0−9 and the directories a through z. A package as sysvinit−2.78.tar.gz is stored under $LFS/usr/src/sources/s/ A package as bash−3.02.tar.gz is stored under $LFS/usr/src/sources/b/ and so forth. You don't have to follow this convention of course, I was just giving an example. It's better to keep the packages out of $LFS/usr/src and move them to a subdirectory, so we'll have a clean $LFS/usr/src directory in which we will unpack the packages and work with them.

The next chapter contains the list of all the packages you need to download, but the partition that is going to contain our LFS system isn't created yet. Therefore store the files temporarily somewhere where you want and remember to copy them to $LFS/usr/src/<somesubdirectory> when you have finished the chapter in which you prepare a new partition (which chapter exactly depends on your architecture).

How to install the software

Before you can actually start doing something with a package, you need to unpack it first. Often you will find the package files being tar'ed and gzip'ed (you can see this from a .tar.gz or .tgz extension). I'm not going to write down every time how to ungzip and how to untar an archive. I will tell you how to do that once, in this paragraph. There is also the possibility that you have the ability of downloading a .tar.bz2 file. Such a file is tar'ed and compressed with the bzip2 program. Bzip2 achieves a better compression than the commonly used gzip does. In order to use bz2 archives you need to have the bzip2 program installed. Most if not every distribution comes with this program so chances are high it is already installed on your system. If not, install it using your distribution's installation tool.

To start with, change to the $LFS/usr/src directory by running:

```
root:~# cd $LFS/usr/src
```

When you have a file that is tar'ed and gzip'ed, you unpack it by running either one of the following two commands, depending on the filename format:

```
root:/usr/src# tar xvfz filename.tar.gz
root:/usr/src# tar xvfz filename.tgz
```

When you have a file that is tar'ed and bzip'ed, you unpack it by running:

```
root:/usr/src#    tar --use-compress-prog=bzip2 -xvf
filename.tar.bz2
```

When you have a file that is tar'ed, you unpack it by running:

```
root:/usr/src# tar xvf filename.tar
```

When the archive is unpacked a new directory will be created under the current directory (and this document assumes that you unpack the archives under the $LFS/usr/src directory). You have to enter that new directory before you continue with the installation instructions. So everytime the book is going to install a program, it's up to you to unpack the source archive. I'm not going to tell you every time to unpack it.

After you have installed a package you can do two things with it. You can either delete the directory that contains the sources or you can keep it. If you decide to keep it, that's fine by me. But if you need the same package again in a later chapter you need to delete the directory first before using it again. If you don't do this, you might end up in trouble because old settings will be used (settings that apply to your normal Linux system but which don't always apply to your LFS system). Doing a simple make clean does not always guarantee a totally clean source tree. The configure script can also have files lying around in various subdirectories which aren't always removed by a make clean process.

II. Part II – Installation of a basic system on Intel systems

Table of Contents

Chapter 3. Packages you need to download

Below is a list of all the packages you need to download for building the basic system. The version numbers printed correspond to versions of the software that is known to work and which this book is based on. If you experience problems which you can't solve yourself, download the version that is assumed in this book (in case you download a newer version).

Please note that this list used to be ordered on usage, meaning that the first package mentioned in this list was the first package used in this book. That's no longer the case because several chapters have been moved around, so that doens't apply. I didn't have the time to re−order this list in this development release. The next release will have this list ordered again.

- Sysvinit (2.78): ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/
- Bash (2.04): ftp://ftp.gnu.org/gnu/bash
- Linux Kernel (2.2.14): ftp://ftp.kernel.org/pub/linux/kernel/
- Binutils (2.9.5.0.37): ftp://ftp.varesearch.com/pub/support/hjl/binutils/
- Bzip2 (0.9.5d): http://sourceware.cygnus.com/bzip2/
- Diff Utils (2.7): ftp://ftp.gnu.org/gnu/diffutils/
- File Utils (4.0): ftp://ftp.gnu.org/gnu/fileutils/
- GCC (2.95.2): ftp://ftp.gnu.org/gnu/gcc/
- Glibc (2.1.3): ftp://ftp.gnu.org/gnu/glibc/
- Glibc−crypt (2.1.3): ftp://ftp.gwdg.de/pub/linux/glibc/
- Glibc−linuxthreads (2.1.3): ftp://ftp.gnu.org/gnu/glibc/
- Grep (2.4.2): ftp://ftp.gnu.org/gnu/grep/
- Gzip (1.2.4a): ftp://ftp.gnu.org/gnu/gzip/
-

Make (3.78.1): ftp://ftp.gnu.org/gnu/make/

- Ed (0.2): ftp://ftp.gnu.org/gnu/ed/
- Patch (2.5.4): ftp://ftp.gnu.org/gnu/patch/
- Sed (3.02): ftp://ftp.gnu.org/gnu/sed/
- Shell Utils (2.0): ftp://ftp.gnu.org/gnu/sh−utils/
- Tar (1.13): ftp://ftp.gnu.org/gnu/tar/
- Text Utils (2.0): ftp://ftp.gnu.org/gnu/textutils/
- Util Linux (2.10h): ftp://ftp.win.tue.nl/pub/linux/utils/util−linux/
- Bison (1.28): ftp://ftp.gnu.org/gnu/bison/
- Mawk (1.3.3) ftp://ftp.whidbey.net/pub/brennan/
- Find Utils (4.1): ftp://ftp.gnu.org/gnu/findutils/
- Termcap (1.3): ftp://ftp.gnu.org/gnu/termcap/
- Ncurses (5.0): ftp://ftp.gnu.org/gnu/ncurses/
- Less (340): ftp://ftp.gnu.org/gnu/less/
- Perl (5.6.0): http://www.perl.com
- M4 (1.4): ftp://ftp.gnu.org/gnu/m4/
- Texinfo (4.0): ftp://ftp.gnu.org/gnu/texinfo/
-

Autoconf (2.13): ftp://ftp.gnu.org/gnu/autoconf/

- Automake (1.4): ftp://ftp.gnu.org/gnu/automake/

- Flex (2.5.4a): ftp://ftp.gnu.org/gnu/flex/

- E2fsprogs (1.18): ftp://tsx−11.mit.edu/pub/linux/packages/ext2fs/

- File (3.26): http://www.linuxfromscratch.org/download/file−3.26−lfs.tar.gz

- Groff (1.15): ftp://ftp.gnu.org/gnu/groff/

- Ld.so (1.9.9): ftp://tsx−11.mit.edu/pub/linux/packages/GCC/

- Libtool (1.3.4): ftp://ftp.gnu.org/gnu/libtool/

- Linux86 (0.14.3): http://www.linuxfromscratch.org/download/linux86−0.14.3−lfs.tar.gz

- Lilo (21.4.2): ftp://sd.dynhost.com/pub/linux/lilo

- Shadow Password Suite (19990827): ftp://piast.t19.pwr.wroc.pl/pub/linux/shadow/

- Man (1.5h1): ftp://ftp.win.tue.nl/pub/linux−local/utils/man/

- Modutils (2.3.9): ftp://ftp.ocs.com.au/pub/modutils/

- Procinfo (17): ftp://ftp.cistron.nl/pub/people/svm/

- Procps (2.0.6): ftp://people.redhat.com/johnsonm/procps/

- Psmisc (19): ftp://lrcftp.epfl.ch/pub/linux/local/psmisc/

- Start−stop−daemon (0.4.1): http://www.linuxfromscratch.org/download/ssd−0.4.1.tar.gz

-

Sysklogd (1.3.31): ftp://sunsite.unc.edu/pub/Linux/system/daemons/

- Vim−rt + Vim−src (5.6): ftp://ftp.vim.org/pub/editors/vim/unix/

Chapter 4. Preparing a new partition

Introduction

In this chapter the partition that is going to host the LFS system is going to be prepared. A new partition will be created, an ext2 file system will be created on it and the directory structure will be created. When this is done, we can move on to the next chapter and start building a new Linux system from scratch.

Creating a new partition

Before we can build our new Linux system, we need to have an empty Linux partition on which we can build our new system. I recommend a partition size of at least 5 00 MB. You can get away with around 250MB for a bare system with no extra bells and whistles (such as software for emailing, networking, Internet, X Window System and such). If you already have a Linux Native partition available, you can skip this subsection.

Start the fdisk program (or some other fdisk program you prefer) with the appropriate hard disk as the option (like /dev/hda if you want to create a new partition on the primary master IDE disk). Create a Linux Native partition, write the partition table and exit the fdisk program. If you get the message that you need to reboot your system to ensure that that partition table is updated, then please reboot your system now before continuing. Remember what your new partition's designation is. It could be something like hda5 (as it is in my case). This newly created partition will be referred to as the LFS partition in this book.

Creating a ext2 file system on the new partition

Once the partition is created, we have to create a new ext2 file system on that partition. To create a new ext2 file system we use the mke2fs command. Enter the new partition as the only option and the file system will be created. If your partition was hda5, you would run:

```
root:~# mke2fs /dev/hda5
```

Mounting the new partition

Now that we have created the ext2 file system, it is ready for use. All we have to do to be able to access it (as in reading from and writing date to it) is mounting it. If you mount it under /mnt/lfs, you can access this partition by going to the /mnt/lfs directory and then do whatever you need to do. This document will assume that you have mounted the partition on a subdirectory under /mnt. It doesn't matter which directory you choose (or you can use just the /mnt directory as the mount point) but this book will assume /mnt/lfs in the commands it tells you to execute.

Create the /mnt/lfs directory by runnning:

```
root:~#   mkdir -p /mnt/lfs
```

Now mount the LFS partition by running:

```
root:~#   mount /dev/xxx /mnt/lfs
```

Replace "xxx" by your partition's designation.

This directory (/mnt/lfs) is the $LFS variable you have read about earlier. So if you read somewhere to "cp inittab $LFS/etc" you actually will type "cp inittab /mnt/lfs/etc".

Creating directories

Let's create the directory tree on the LFS partition according to the FHS standard which can be found at http://www.pathname.com/fhs/. Issuing the following commands will create the necessary directories:

```
root:~#   cd $LFS
root:lfs#   mkdir bin boot dev etc home lib mnt proc root
sbin tmp usr var
root:lfs#   cd $LFS/usr
root:usr#   mkdir bin include lib local sbin share src
root:usr#   ln -s share/man man
root:usr#   ln -s share/doc doc
root:usr#   ln -s share/info info
root:usr#   ln -s ../etc etc
root:usr#   ln -s ../var var
root:usr#   cd $LFS/usr/share
root:share#   mkdir dict doc info locale man nls misc
terminfo zoneinfo
root:share#   cd $LFS/usr/share/man
root:man#   mkdir man1 man2 man3 man4 man5 man6 man7 man8
root:man#   cd $LFS/var
root:var#   mkdir lock log run spool tmp
```

Normally directories are created with permission mode 755, which isn't desired for all directories. I haven't checked the FHS if they suggest default modes for certain directories, so I'll just change the modes for two directories. The first change is a mode 0750 for the $LFS/root directory. This is to make sure that not just everybody can enter the /root directory (the same you would do with /home/username directories). The second change is a mode 1777 for the $LFS/tmp directory. This way every user can write stuff to the /tmp directory if they need to. The sticky (1) bit makes sure users can't delete other user's file which they normally can do because the directory is set in such a way that every body (owner, group, world) can write to that directory.

```
root:~#   cd $LFS
root:lfs#   chmod 0750 root
root:lfs#   chmod 1777 tmp
```

Now that the directories are created, copy the source files you have downloaded in chapter 3 to some subdirectory under $LFS/usr/src (you will need to create this subdirectory yourself).

Copying the /dev directory

We can create every single file that we need to be in the $LFS/dev directory using the mknod command, but that just takes up a lot of time. I choose to just simply copy the current /dev directory to the $LFS partition. Use this command to copy the entire directory while preserving original rights, symlinks and ownerships:

```
root:~# cp −av /dev $LFS
root:~#    chown root $LFS/dev/*
```

I'm aware that this isn't the best way to create the files. I know of a MAKEDEV script but I choose not to use it. I'm actually waiting for the 2.4 Linux kernel to be released. The kernel has a stable version of the devfs which this book will use in the future. Devfs is a dynamic file system which makes the static files in /dev obsolete. You mount the dev file system to a mount point (kind of like the way the proc file system works) and the kernel will create the files in /dev you need on−the−fly. So the waiting is for the next stable kernel to be released.

Chapter 5. Installing basic system software

How and why things are done

In this chapter we will install all the software that belongs to a basic Linux system. After you're done with this chapter you have a fully working Linux system. The remaining chapters deal with optional issues such as setting up networking, Internet servers + clients (telnet, ftp, http, email), setting up Internet itself and the X Window System. You can skip chapters at your own discretion. If you don't plan on going online with the LFS system there's little use to setup Internet for example.

This chapter is devided in two chunks. The first part installs a few necessary programs on the LFS system. These programs are needed to install the rest of the programs that belong to a basic system. When the first part is done, we will enter a chroot'ed environment. This means that we start a shell with $LFS as the root directory (instead of the usual / directory as the root directory). This has the same effect as rebooting the computer into the LFS system, but this way we don't have to reboot. If something goes wrong, you don't need to reboot back in the normal Linux system to fix whatever you need to fix. You just open a new shell on a virtual console, or start a new xterm and you can do what you need to do.

The software in the first part will be linked statically. These programs will be re−installed in the second part and linked dynamically. The reason for the static version first is that there is a chance that our normal Linux system and our LFS system−to−be don't use the same C Library versions. If the programs in the first part are linked against an older C library version, those program might not work too well on the LFS system.

The key to learn what makes Linux tick is to know exactly what packages are used for and why you or the system needs them. In depth descriptions of every package is provided in Appendix A.

About debugging symbols

Every program and library is by default compiled with debugging symbols. This means you can run a program or library through a debugger and the debugger's output will be more user friendly. These debugging symbols also enlarge the program or library significantly. This document will not install software without debugging symbols (as I don't know if the majority of readers do or do not debug software). In stead, you can remove those symbols manually if you want with the strip program.

To remove debugging symbols from a binary (must be an a.out or ELF binary) run **`strip --strip-debug filename`** You can use wild cards if you need to strip debugging symbols from multiple files (use something like strip ––strip–debug $LFS/usr/bin/*).

Before you wonder if these debugging symbols would make a big difference, here are some statistics:

- A static Bash binary with debugging symbols: 2.3MB
- A static Bash binary without debugging symbols: 645KB
- A dynamic Bash binary with debugging symbols: 1.2MB
- A dynamic Bash binary without debugging symbols: 478KB
- $LFS/lib and $LFS/usr/lib (glibc and gcc files) with debugging symbols: 87MB
- $LFS/lib and $LFS/usr/lib (glibc and gcc files) without debugging symbols: 16MB

Sizes may vary depending on which compiler was used and which C library version was used to link dynamic programs against, but your results will be similar if you compare programs with and without debugging symbols. After I was done with this chapter and stripped all debugging symbols from all LFS binaries and libraries I regained a little over 102 MB of disk space. Quite the difference. The difference would be even greater when I would do this at the end of this book when everything is installed.

Preparing the LFS system for installing basic system software

Installing Bash

Install Bash by running the following commands:

```
root:bash-2.04#   ./configure --enable-static-link
root:bash-2.04#   make
root:bash-2.04#   make -e prefix=$LFS/usr install
root:bash-2.04#   mv $LFS/usr/bin/bash $LFS/bin
root:bash-2.04#   cd $LFS/bin
root:bin#   ln -s bash sh
```

Installing Binutils

Install Binutils by running the following commands:

```
root:binutils-2.9.5.0.37#   ./configure --prefix=/usr
root:binutils-2.9.5.0.37#   make -e LDFLAGS=-all-static
root:binutils-2.9.5.0.37a  make -e prefix=$LFS/usr install
```

Installing Bzip2

Before we can install Bzip2 we need to modify the `Makefile` file. Open the `Makefile` file in a text editor and find the lines that start with *$(CC) $(CFLAGS) -o*

Replace those parts with: *$(CC) $(CFLAGS) $(LDFLAGS) -o*

Now install Bzip2 by running the following commands:

```
root:bzip2-0.9.5d#   make -e LDFLAGS=-static
root:bzip2-0.9.5d#   make -e PREFIX=$LFS/usr install
root:bzip2-0.9.5d#   cd $LFS/usr/bin
root:bin#   mv bunzip2 bzip2 $LFS/bin
```

Installing Diffutils

Install Diffutils by running the following commands:

```
root:diffutils-2.7#   ./configure --prefix=/usr
root:diffutils-2.7#   make -e LDFLAGS=-static
root:diffutils-2.7#   make -e prefix=$LFS/usr install
```

This package is known to cause static link problems on certain platforms. If you're having trouble compiling this package as well, you can download a patch from http://www.linuxfromscratch.org/download/diffutils–2.7.patch.gz

Install this patch by running the following command:

```
root:diffutils-2.7#   patch -Np1 -i ../diffutils-2.7.patch
```

Now recompile the package using the same commands as above.

Installing Fileutils

Install Fileutils by running the following commands:

```
root:fileutils-4.0#   ./configure --disable-nls --prefix=/usr
root:fileutils-4.0#   make -e LDFLAGS=-static
root:fileutils-4.0#   make -e prefix=$LFS/usr install
root:fileutils-4.0#   cd $LFS/usr/bin
root:bin#   mv chgrp chmod chown cp dd df ln $LFS/bin
root:bin#   mv ls mkdir mknod mv rm rmdir sync $LFS/bin
```

Installing GCC on the normal system if necessary

In order to compile Glibc−2.1.3 later on you need to have gcc−2.95.2 installed. Although any GCC version above 2.8 would do, 2.95.2 is the highly recommended version to use. Many glibc−2.0 based systems have gcc−2.7.2.3 installed and you can't compile glibc−2.1.3 with that compiler. Many glibc−2.1 based systems have egcs−2.95.x installed and that version doesn't work too well either (sometimes it works fine, sometimes it doesn't depending on various circumstances).

To find out whether your system uses gcc−2.95.2 or not, run the following command:

```
root:~#   gcc --version
```

If you normal Linux system does not have gcc−2.95.2 installed you need to install it now. We won't replace the current compiler on your system, but instead we will install gcc in a separate directory (/usr/local/gcc2952). This way no binaries or header files will be replaced.

After you unpacked the gcc−2.95.2 archive don't enter the newly created gcc−2.95.2 directory but stay in the $LFS/usr/src directory. Install GCC by running the following commands:

```
root:src#   mkdir $LFS/usr/src/gcc-build
root:src#   cd $LFS/usr/src/gcc-build
root:gcc-build#   ../gcc-2.95.2/configure
--prefix=/usr/local/gcc2952 \
> --with-local-prefix=/usr/local/gcc2952  \
> --with-gxx-include-dir=/usr/local/gcc2952/include/g++  \
> --enable-shared --enable-languages=c,c++
root:gcc-build#   make bootstrap
root:gcc-build#   make install
```

Installing GCC on the LFS system

After you unpacked the gcc−2.95.2 archive don't enter the newly created gcc−2.95.2 directory but stay in the $LFS/usr/src directory. Install GCC by running the following commands:

```
root:src#   mkdir $LFS/usr/src/gcc-build
root:src#   cd $LFS/usr/src/gcc-build
root:gcc-build#   ../gcc-2.95.2/configure \
```

```
> --prefix=/usr --with-local-prefix=/usr \
> --with-gxx-include-dir=/usr/include/g++ \
> --enable-languages=c,c++ --disable-nls
root:gcc-build#   make -e LDFLAGS=-static bootstrap
root:gcc-build#   make -e prefix=$LFS/usr
local_prefix=$LFS/usr \
gxx_include_dir=$LFS/usr/include/g++ \
> install
```

Creating necessary symlinks

The system needs a few symlinks to ensure every program is able to find the compiler and the pre−processor. Some programs run the cc program, others run the gcc program. Some programs expect the cpp program in /lib and others expect to find it in /usr/bin. Create those symlinks by running:

```
root:~#   cd $LFS/lib
root:lib#   ln -s ../usr/lib/gcc-lib/<host>/2.95.2/cpp cpp
root:lib#   cd $LFS/usr/lib
root:lib#   ln -s gcc-lib/<host>/2.95.2/cpp cpp
root:lib#   cd $LFS/usr/bin
root:bin#   ln -s gcc cc
```

Replace <host> with the directory where the gcc−2.95.2 files are installed (which is i686−unknown−linux in my case).

Installing Glibc

A note on the glibc−crypt package

An excerpt from the README file that is distributed with the glibc−crypt package:

The add−on is not included in the main distribution of the GNU C library because some governments, most notably those of France, Russia, and the US, have very restrictive rules governing the distribution and use of encryption software. Please read the node "Legal Problems" in the manual for more details.

In particular, the US does not allow export of this software without a licence, including via the Internet. So please do not download it from the main FSF FTP site at ftp.gnu.org if you are outside the US. This software was completely developed outside the US.

"This software" refers to the glibc−crypt package at ftp://ftp.gwdg.de/pub/linux/glibc/. This law only affects people who don't live in the US. It's not prohibited to import DES software, so if you live in the US you can import the file safely from Germany without breaking cryptographic laws. This law is changing lately and I don't know what the status of it is at the moment. Better be safe than sory.

Installing Glibc

Copy the Glibc−crypt and Glibc−linuxthreads archives into the unpacked glibc directory

Unpack the glibc−crypt and glibc−linuxthreads archives there, but don't enter the created directories. Just unpack and leave it with that.

A few default parameters of Glibc need to be changed, such as the directory where the shared libraries are supposed to be installed in and the directory that contains the system configuration files. For this purpose you need to create the `$LFS/usr/src/glibc-build` directory and in that directory you create a new file `configparms` containing:

```
# Begin configparms

slibdir=/lib
sysconfdir=/etc

# End configparms
```

Change to the `$LFS/usr/src/glibc-build` directory and install Glibc by running the following commands if your system already had a suitable GCC version installed:

```
root:glibc-build# ../glibc-2.1.3/configure --prefix=/usr
--enable-add-ons
root:libc-build# make
root:glibc-build# make install_root=$LFS install
```

Change to the `$LFS/usr/src/glibc-build` directory and install Glibc by running the following command if your system did not already have a suitable GCC version installed and you just installed GCC−2.95.2 on your normal Linux system a little while ago:

```
root:glibc-build# CC=/usr/local/gcc2952/bin/gcc \
> ../glibc-2.1.3/configure --prefix=/usr --enable-add-ons
root:glibc-build# make
root:glibc-build# make install_root=$LFS install
```

Copying old NSS library files

If your normal Linux system runs glibc−2.0, you need to copy the NSS library files to the LFS partition. Certain statically linked programs still depend on the NSS library, especially programs that need to lookup usernames,userid's and groupid's. You can check which C library version your normal Linux system uses by running:

```
root:~# ls /lib/libc*
```

Your system uses glib−2.0 if there is a file that looks like *libc−2.0.7.so*

Your system uses glibc−2.1 if there is a file that looks like *libc−2.1.3.so*

Of course, the micro version number can be different (you could have libc−2.1.2 or libc−2.1.1 for example).

If you have a libc−2.0.x file copy the NSS library files by running:

```
root:~# cp −av /lib/libnss*  $LFS/lib
```

There are a few distributions that don't have files from which you can see which version of the C Library it is. If that's the case, it will be hard to determine which C library version you exactly have. Try to obtain this information using your distribution's installation tool. It often says which version it has available. If you can't figure out at all which C Library version is used, then copy the NSS files anyway and hope for the best. That's the best advise I can give I'm afraid.

Installing Grep

Install Grep by running the following commands:

```
root:grep−2.4.2#   ./configure −−prefix=/usr −−disable−nls
root:grep−2.4.2#   make −e LDFLAGS=−static
root:grep−2.4.2#   make −e prefix=$LFS/usr install
```

This package is known to cause static linking problems on certain platforms. If you're having trouble compiling this package as well, you can download a patch from http://www.linuxfromscratch.org/download/grep–2.4.2.patch.gz

Install this patch by running the following command:

```
root:grep-2.4.2#   patch -Np1 -i ../grep-2.4.2.patch
```

Now recompile the package using the same commands as above.

Installing Gzip

Install Gzip by running the following commands:

```
root:gzip-1.2.4a#   ./configure --prefix=/usr
root:gzip-1.2.4a#   make -e LDFLAGS=-static
root:gzip-1.2.4a#   make -e prefix=$LFS/usr install
root:gzip-1.2.4a#   cd $LFS/usr/bin
root:bin#   mv gunzip gzip $LFS/bin
```

This package is known to cause compilation problems on certain platforms. If you're having trouble compiling this package as well, you can download a fixed package from http://www.linuxfromscratch.org/download/gzip–1.2.4a.patch.gz

Install this patch by running the following command:

```
root:gzip-1.2.4a#   patch -Np1 -i ../gzip-1.2.4a.patch.gz
```

Now recompile the package using the same commands as above.

Installing Make

Install Make by running the following commands:

```
root:make-3.78.1#   ./configure --prefix=/usr --disable-nls
root:make-3.78.1#   make -e LDFLAGS=-static
root:make-3.78.1#   make -e prefix=$LFS/usr install
```

Installing Sed

Install Sed by running the following commands:

```
root:sed-3.02#   ./configure --prefix=/usr
root:sed-3.02#   make -e LDFLAGS=-static
root:sed-3.02#   make -e prefix=$LFS/usr install
root:sed-3.02#   mv $LFS/usr/bin/sed $LFS/bin
```

This package is known to cause static linking problems on certain platforms. If you're having trouble compiling this package as well, you can download a patch from http://www.linuxfromscratch.org/download/sed–3.02.patch.gz

Install this patch by running the following command:

```
root:sed-3.02#   patch -Np1 -i ../sed-3.02.patch.gz
```

Now recompile the package using the same commands as above.

Installing Shellutils

Install Shellutils by running the following commands:

```
root:sh-utils-2.0#   ./configure --prefix=/usr --disable-nls
root:sh-utils-2.0#   make -e LDFLAGS=-static
root:sh-utils-2.0#   make -e prefix=$LFS/usr install
root:sh-utils-2.0#   cd $LFS/usr/bin
root:/bin#   mv date echo false pwd stty $LFS/bin
root:bin#   mv su true uname hostname $LFS/bin
```

Installing Tar

Install Tar by running the following commands:

```
root:tar-1.13#   ./configure --prefix=/usr --disable-nls
root:tar-1.13#   make -e LDFLAGS=-static
root:tar-1.13#   make -e prefix=$LFS/usr install
root:tar-1.13#   mv $LFS/usr/bin/tar $LFS/bin
```

Installing Textutils

Install Textutils by running the following commands:

```
root:textutils-2.0#   ./configure --prefix=/usr --disable-nls
root:textutils-2.0#   make -e LDFLAGS=-static
root:textutils-2.0#   make -e prefix=$LFS/usr install
root:textutils-2.0#   mv $LFS/usr/bin/cat $LFS/bin
```

Creating passwd and group files

Create a new file `$LFS/etc/passwd` containing the following:

```
root::0:0:root:/root:/bin/bash
```

Create a new file `$LFS/etc/group` containing the following:

```
root::0:
```

Installing basic system software

The installation of all the software is pretty straightforward and you'll think it's so much easier and shorter to give the generic installation instructions for each package and only explain how to install something if a certain package requires an alternate installation method. Although I agree with you on that, I, however, choose to give the full instructions for each and every package. This is simply to avoid any possible confusion and errors.

Entering the chroot'ed environment

It's time to enter our chroot'ed environment now in order to install the rest of the software we need.

Enter the following commands to setup the chroot'ed environment. From this point on there's no need to use the $LFS variable anymore, because everything you do will be restricted to the LFS partition (since / is actually /mnt/xxx but the shell doesn't know that).

```
root:~#   cd $LFS/root
root:root#   chroot $LFS bash --login
```

Now that we are inside a chroot'ed environment, we can continue to install all the basic system software. Make sure you execute all the following commands in this chapter from within the chroot'ed environment.

Installing Linux Kernel

We won't be compiling a new kernel image yet. We'll do that after we have finished the installation of the basic system software in this chapter. But because certain software need the kernel header files, we're going to unpack the kernel archive now and set it up so that we can compile package that need the kernel.

Create the kernel configuration file by running the following command:

```
root:linux#   yes "" | make config
```

Ignore the warning *Broken pipe* you might see at the end. Now run the following commands to set up all the dependencies correctly:

```
root:linux#   make dep
```

Now that that's done, we need to create the $LFS/usr/include/linux and the $LFS/usr/include/asm symlinks. Create them by running the following commands:

```
root:~#   cd /usr/include
root:include#   ln -s ../src/linux/include/linux linux
root:include#   ln -s ../src/linux/include/asm asm
```

Installing Ed

Install Ed by running the following commands:

```
root:/usr/src/ed-0.2#   ./configure --prefix=/usr
root:/usr/src/ed-0.2#   make
root:/usr/src/ed-0.2#   make install
```

Installing Patch

Install Patch by running the following commands:

```
root:patch-2.5.4#   ./configure --prefix=/usr
root:patch-2.5.4#   make
root:patch-2.5.4#   make install
```

Installing GCC

After you unpacked the gcc−2.95.2 archive don't enter the newly created gcc−2.95.2 directory but stay in the /usr/src directory. Install GCC by running the following commands:

```
root:src#   mkdir /usr/src/gcc-build
root:src#   cd /usr/src/gcc-build
root:gcc-build#   ../gcc-2.95.2/configure \
> --prefix=/usr --with-local-prefix=/usr \
> --with-gxx-include-dir=/usr/include/g++ \
> --enable-shared --enable-languages=c,c++
root:gcc-build#   make bootstrap
root:gcc-build#   make install
```

Installing Bison

Install Bison by running the following commands:

```
root:bison-1.28#   ./configure --prefix=/usr
--datadir=/usr/share/bison
root:bison-1.28#   make
root:bison-1.28#   make install
```

Installing Mawk

Install Mawk by running the following commands:

```
root:mawk-1.3.3#   ./configure
root:mawk-1.3.3#   make
root:mawk-1.3.3#   make -e BINDIR=/usr/bin
MANDIR=/usr/share/man/man1 install
root:mawk-1.3.3#   cd /usr/bin
root:in#   ln -s mawk awk
```

Installing Findutils

Install Findutils by running the following commands:

```
root:findutils-4.1#   ./configure --prefix=/usr
```

```
root:findutils-4.1#   make
root:findutils-4.1#   make install
```

This package is known to cause compilation problem. If you're having trouble compiling this package as well, you can download a patch from http://www.linuxfromscratch.org/download/findutils–4.1.patch.gz

Install this patch by running the following command:

```
root:findutils-4.1#   patch -Np1 -i ../findutils-4.1.patch.gz
```

Now recompile the package using the same commands as above.

Installing Termcap

Install Termcap by running the following commands:

```
root:termcap-1.3#   ./configure --prefix=/usr
root:termcap-1.3#   make
root:termcap-1.3#   make install
```

Installing Ncurses

Install Ncurses by running the following commands:

```
root:ncurses-5.0#   ./configure --prefix=/usr --with-shared
root:ncurses-5.0#   make
root:ncurses-5.0#   make install
```

Installing Less

Install Less by running the following commands:

```
root:less-340#   ./configure --prefix=/usr
root:less-340#   make
root:less-340#   make install
root:less-340#   mv /usr/bin/less /bin
```

Installing Perl

Install Perl by running the following commands:

```
root:perl-5.6.0#   ./Configure
root:perl-5.6.0#   make
root:perl-5.6.0#   make test
root:perl-5.6.0#   make install
```

Note that you have to change the installation path to `/usr` yourself. The Perl installation defaults to the /usr/local/subdir

Also note that a few tests during the make test phase will fail because we don't have network support installed yet.

Installing M4

Install M4 by running the following commands:

```
root:m4-1.4#   ./configure --prefix=/usr
root:m4-1.4#   make
root:m4-1.4#   make install
```

Installing Texinfo

Install Texinfo by running the following commands:

```
root:texinfo-4.0#   ./configure --prefix=/usr
root:texinfo-4.0#   make
root:texinfo-4.0#   make install
```

Installing Autoconf

Install Autoconf by running the following commands:

```
root:autoconf-2.13#   ./configure --prefix=/usr
root:autoconf-2.13#   make
root:autoconf-2.13#   make install
```

Installing Automake

Install Automake by running the following commands:

```
root:automake-1.4#   ./configure --prefix=/usr
root:automake-1.4#   make install
```

Installing Bash

Install Bash by running the following commands:

```
root:bash-2.04#   ./configure --prefix=/usr
root:bash-2.04#   make
root:bash-2.04#   make install
root:bash-2.04#   logout
root:root#   mv $LFS/usr/bin/bash $LFS/bin
root:root#   chroot $LFS bash --login
```

Installing Flex

Install Flex by running the following commands:

```
root:flex-2.5.4a#   ./configure --prefix=/usr
root:flex-2.5.4a#   make
root:flex-2.5.4a#   make install
```

Installing Binutils

Install Binutils by running the following commands:

```
root:binutils-2.9.5.0.37#   ./configure --prefix=/usr
root:binutils-2.9.5.0.37#   make
root:binutils-2.9.5.0.37#   make install
```

Installing Bzip2

Install Bzip2 by running the following commands:

```
root:bzip2-0.9.5d#   make
root:bzip2-0.9.5d#   make PREFIX=/usr install
root:bzip2-0.9.5d#   cd /usr/bin
root:bin#   mv bunzip2 bzip2 /bin
```

Installing Diffutils

Install Diffutils by running the following commands:

```
root:diffutils-2.7#   ./configure --prefix=/usr
```

```
root:diffutils-2.7#   make
root:diffutils-2.7#   make install
```

Installing E2fsprogs

Install E2fsprogs by running the following commands:

```
root:e2fsprogs-1.18#    ./configure --prefix=/usr
--with-root-prefix=/
root:e2fsprogs-1.18#    make
root:e2fsprogs-1.18#    make install
root:e2fsprogs-1.18#    cd /usr/sbin
root:sbin#    mv *e2* *fs* mklost+found /sbin
```

Installing File

Install File by running the following commands:

```
root:file-3.26#   ./configure --prefix=/usr
root:file-3.26#   make
root:file-3.26#   make install
```

Installing Fileutils

Install Fileutils by running the following commands:

```
root:fileutils-4.0#   ./configure --prefix=/usr
root:fileutils-4.0#   make
root:fileutils-4.0#   make install
root:fileutils-4.0#   cd /usr/bin
root:bin#   mv chgrp chmod chown cp dd df ln /bin
root:bin#   mv ls mkdir mknod mv rm rmdir sync /bin
```

Installing Grep

Install Grep by running the following commands:

```
root::grep-2.4.2#   ./configure --prefix=/usr
root:grep-2.4.2#   make
root:grep-2.4.2#   make install
```

Installing Groff

Install Groff by running the following commands:

```
root:groff-1.15#   ./configure --prefix=/usr
root:groff-1.15#   make
root:groff-1.15#   make install
```

Installing Gzip

Install Gzip by running the following commands:

```
root:gzip-1.2.4a#   ./configure --prefix=/usr
root:gzip-1.2.4a#   make
root:gzip-1.2.4a#   make install
root:gzip-1.2.4a#   cd /usr/bin
root:bin#   mv gunzip gzip /bin
```

Installing Ld.so

Install Ld.so by running the following commands:

```
root:ld.so-1.9.10#   cd util
root:util#   make ldd ldconfig
root:util#   cp ldd /bin
root:util#   cp ldconfig /sbin
root:util#   cd ../man
root:man#   cp ldd.1 /usr/share/man/man1
root:man#   cp *.8 /usr/share/man/man8
root:man#   rm /usr/bin/ldd
```

Installing Libtool

Install Libtool by running the following commands:

```
root:libtool-1.3.4#   ./configure --prefix=/usr
root:libtool-1.3.4#   make
root:libtool-1.3.4#   make install
```

Installing Linux86

Install Linux86 by running the following commands:

```
root:linux-86#   cd as
root:as#   make
root:as#   make install
root:as#   cd ../ld
root:ld#   make ld86
root:ld#   make install
root:ld#   cd ../man
root:man#   cp as86.1 ld86.1 /usr/share/man/man1
```

Installing Lilo

Install Lilo by running the following commands:

```
root:lilo-21.4.1#   make
root:lilo-21.4.1#   make install
```

Installing Make

Install Make by running the following commands:

```
root:make-3.78.1#   ./configure --prefix=/usr
root:make-3.78.1#   make
root:make-3.78.1#   make install
```

Installing Shell Utils

Install Shellutils by running the following commands:

```
root:sh-utils-2.0#   ./configure --prefix=/usr
root:sh-utils-2.0#   make
root:sh-utils-2.0#   make install
root:sh-utils-2.0#   cd /usr/bin
root:bin#   mv date echo false pwd stty /bin
root:bin#   mv su true uname hostname /bin
```

Installing Shadow Password Suite

Install the Shadow Password Suite by running the following commands:

```
root:shadow-19990827#   ./configure --prefix=/usr
root:shadow-19990827#   make
root:shadow-19990827#   make install
root:shadow-19990827#   cd etc
root:etc#   cp limits login.access login.defs.linux shells
suauth /etc
```

```
root:etc#   mv /etc/login.defs.linux /etc/login.defs
```

Installing Man

Install Man by running the following commands:

```
root:man-1.5h1#   ./configure -default
root:man-1.5h1#   make
root:man-1.5h1#   make install
```

Installing Modutils

Install Modutils by running the following commands:

```
root:modutils-2.3.9#   ./configure
root:modutils-2.3.9#   make
root:modutils-2.3.9#   make install
```

Installing Procinfo

Install Procinfo by running the following commands:

```
root:procinfo-17#   make
root:procinfo-17#   make install
```

Installing Procps

Install Procps by running the following commands:

```
root:procps-2.0.6#   gcc -c watch.c
root:procps-2.0.6#   make
root:procps-2.0.6#   make -e XSCPT="" install
root:procinfo-17#   mv /usr/bin/kill /bin
```

Installing Psmisc

Install Psmisc by running the following commands:

```
root:psmisc-19#   make
root:psmisc-19#   make install
```

Installing Sed

Install Sed by running the following commands:

```
root:sed-3.02#   ./configure --prefix=/usr
root:sed-3.02#   make
root:sed-3.02#   make install
root:sed-3.02#   mv /usr/bin/sed /bin
```

Installing Start–stop–daemon

Install Start–stop–daemon by running the following commands:

```
root:ssd-0.4.1#   make
root:ssd-0.4.1#   make install
```

Installing Sysklogd

Install Sysklogd by running the following commands:

```
root:sysklogd-1.3-31#   make
root:sysklogd-1.3-31#   make install
```

Installing Sysvinit

Install Sysvinit by running the following commands:

```
root:sysvinit-2.78#   cd src
root:sysvinit-2.78#   make
root:sysvinit-2.78#   make install
```

Installing Tar

Install Tar by running the following commands:

```
root:tar-1.13#   ./configure --prefix=/usr
root:tar-1.13#   make
root:tar-1.13#   make install
root:tar-1.13#   mv /usr/bin/tar /bin
```

Installing Textutils

Install Textutuils by running the following commands:

```
root:textutils-2.0#   ./configure --prefix=/usr
root:textutils-2.0#   make
```

```
root:textutils-2.0#   make install
root:textutils-2.0#   mv /usr/bin/cat /bin
```

Installing Vim

You need to unpack both the vim−rt and vim−src packages to install Vim. Install Vim by running the following commands:

```
root:vim-5.6#   ./configure --prefix=/usr
root:vim-5.6#   make
root:vim-5.6#   make install
root:vim-5.6#   cd /usr/bin
root:bin#   ln -s vim vi
```

Installing Util−Linux

Before we can install the package we have to edit the MCONFIG file, find and modify the following variables as follows:

HAVE_PASSWD=yes
HAVE_SLN=yes
HAVE_TSORT=yes

Install Util−Linux by running the following commands:

```
root:util-linux-2.10h#   groupadd -g 5 tty
root:util-linux-2.10h#   ./configure
root:util-linux-2.10h#   make
root:util-linux-2.10h#   make install
```

Removing old NSS library files

If you have copied the NSS Library files from your normal Linux system to the LFS system (because your normal system runs glibc–2.0) it's time to remove them now by running:

```
root:~#     rm /lib/libnss*.so.1 /lib/libnss*2.0*
```

Configuring essential software

Now that all software is installed, all that we need to do to get a few programs running properly is to create their configuration files.

Configuring Glibc

We need to create the /etc/nsswitch.conf file. Although glibc should provide defaults when this file is missing or corrupt, it's defaults don't work work well with networking which will be dealt with in a later chapter. Also, our timezone needs to be setup.

Create a new file `/etc/nsswitch.conf` containing:

```
# Begin /etc/nsswitch.conf

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

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

netgroup: db files

# End /etc/nsswitch.conf
```

Run the **`tzselect`** script and answer the questions regarding your timezone. When you're done, the script will give you the location of the timezone file you need.

Create the `/etc/localtime` symlink by running:

```
root:~#   cd /etc
root:etc#   rm localtime
root:etc#   ln -s ../usr/share/zoneinfo/<tzselect's output> \
> localtime
```

tzselect's output can be something like *EST5EDT* or *Canada/Eastern*. The symlink you would create with that information would be *ln −s ../usr/share/zoneinfo/EST5EDT localtime* or *ln −s ../usr/share/zoneinfo/Canada/Eastern localtime*

Configuring Dynamic Loader

By default the dynamic loader searches a few default paths for dynamic libraries, so there normally isn't a need for the `/etc/ld.so.conf` file unless you have extra directories in which you want the system to search for paths. The `/usr/local/lib` directory isn't searched through for dynamic libraries by default, so we want to add this path so when you install software you won't be suprised by them not running for some reason.

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

```
# Begin /etc/ld.so.conf

/lib
/usr/lib
/usr/local/lib

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

Although it's not necessary to add the `/lib` and `/usr/lib` directories it doesn't hurt. This way you see right away what's being searched and don't have to remeber the default search paths if you don't want to.

Configuring Lilo

We're not going to create lilo's configuration file from scratch, but we'll use the file from your normal Linux system. This file is different on every machine and thus I can't create it here. Since you would want to have the same options regarding lilo as you have when you're using your normal Linux system you would create the file exactly as it is on the normal system.

Copy the Lilo configuration file and kernel images that Lilo uses by running the following commands from a shell on your normal Linux system. Don't execute these commands from your chroot'ed shell.

```
root:~#   cp /etc/lilo.conf $LFS/etc
root:~#   cp /boot/<kernel images> $LFS/boot
```

Before you can execute the second command you need to know the names of the kernel images. You can't just copy all files from the /boot directory. The /etc/lilo.conf file contains the names of the kernel images you're using. Open the file and look for lines like this:

```
image=/boot/vmlinuz
```

Look for all *image* variables and their values represent the name and location of the image files. These files will usually be in /boot but they might be in other directories as well, depending on your distribution's conventions.

Configuring Sysklogd

Create the `/etc/syslog.conf` file containing the following:

```
# Begin /etc/syslog.conf

auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *

# End /etc/syslog.conf
```

Configuring Shadow Password Suite

This package contains the utilities to modify user's passwords, add new users/groups, delete users/groups and more. I'm not going to explain to you what 'password shadowing' means. You can read all about that in the doc/HOWTO file. There's one thing you should keep in mind, if you decide to use shadow support, that programs that need to verify passwords (examples are xdm, ftp daemons, pop3 daemons, etc) need to be 'shadow−compliant', eg. they need to be able to work with shadowed passwords.

If you decide you don't want to use shadowed passwords (after you're read the doc/HOWTO document), you still use this archive since the utilities in this archive are also used on system which have shadowed passwords disabled. You can read all about this in the HOWTO. Also note that you can switch between shadow and non−shadow at any point you want.

Now is a very good moment to read chapter 5 of the doc/HOWTO file. You can read how you can test if shadowing works and if not, how to disable it. If it doesn't work and you haven't tested it, you'll end up with an unusable system after you logout of all your consoles, since you won't be able to login anymore. You can easily fix this by passing the init=/sbin/sulogin parameter to the kernel, unpack the util−linux archive, go to the login−utils directory, build the login program and replace the /bin/login by the one in the util−linux package. Things are never hopelessly messed up (at least not under Linux), but you can avoid a hassle by testing properly and reading manuals ;)

Configuring Sysvinit

Create a new file `/etc/inittab` containing the following:

```
# Begin /etc/inittab

id:2:initdefault:

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

su:S:wait:/sbin/sulogin

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

ft:6:respawn:/sbin/sulogin

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

1:2345:respawn:/sbin/agetty /dev/tty1 9600
2:2345:respawn:/sbin/agetty /dev/tty2 9600
3:2345:respawn:/sbin/agetty /dev/tty3 9600
4:2345:respawn:/sbin/agetty /dev/tty4 9600
5:2345:respawn:/sbin/agetty /dev/tty5 9600
6:2345:respawn:/sbin/agetty /dev/tty6 9600

# End /etc/inittab
```

Creating the /var/run/utmp file

Programs like login, shutdown, uptime and others want to read from and write to the /var/run/utmp file. This file contains information about who is currently logged in. It also contains information on when the computer was last booted and shutdown.

Create the `/var/run/utmp` and give it the proper permissions by running the following commands:

```
root:~# touch /var/run/utmp
root:~# chmod 644 /var/run/utmp
```

Configuring Vim

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

```
set nocompatible
set bs=2
```

Chapter 6. Creating system boot scripts

What is being done here

This chapter will create the necessary scripts that are run at boottime. These scripts perform tasks such as remounting the root file system mounted read−only by the kernel into read−write mode, activiating the swap partition(s), running a check on the root file system to make sure it's intact and starting the daemons that the system uses.

Create the directories and master files

We need to start by creating a few extra directories that are used by the boot scripts. Create these directories by running:

```
root:~#   cd /etc
root:etc#   mkdir rc0.d rc1.d rc2.d rc3.d
root:etc#   mkdir rc4.d rc5.d rc6.d init.d rcS.d
```

The first main bootscript is the `/etc/init.d/rc` script. Create a new file `/etc/init.d/rc` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/rc
#
# By Jason Pearce  − jason.pearce@linux.org
#

# Un−comment the following for debugging.
# debug=echo

#
# Start script or program.
#
startup() {
case "$1" in
*.sh)
$debug sh "$@"
;;
*)
$debug "$@"
;;
esac
}

# Ignore CTRL−C only in this shell, so we can interrupt subprocesses.
trap ":" INT QUIT TSTP

# Set onlcr to avoid staircase effect.
stty onlcr 0>&1

# Now find out what the current and what the previous runlevel are.
runlevel=$RUNLEVEL
```

```
# Get first argument. Set new runlevel to this argument.

[ "$1" != "" ] && runlevel=$1
if [ "$runlevel" = "" ]
then
echo "Usage: $0 <runlevel>" >&2
exit 1
fi

previous=$PREVLEVEL
[ "$previous" = "" ] && previous=N

export runlevel previous

# Is there an rc directory for this new runlevel?

if [ −d /etc/rc$runlevel.d ]
then
# First, run the KILL scripts for this runlevel.
if [ $previous != N ]
then
for i in /etc/rc$runlevel.d/K*
do
[ ! −f $i ] && continue

suffix=${i#/etc/rc$runlevel.d/K[0−9][0−9]}
previous_start=/etc/rc$previous.d/S[0−9][0−9]$suffix

# Stop the service if there is a start script
# in the previous run level.
[ ! −f $previous_start ] && continue

startup $i stop
done
fi

# Now run the START scripts for this runlevel.
for i in /etc/rc$runlevel.d/S*
do
[ ! −f $i ] && continue

if [ $previous != N ]
then
# Find start script in previous runlevel and
# stop script in this runlevel.
suffix=${i#/etc/rc$runlevel.d/S[0−9][0−9]}
stop=/etc/rc$runlevel.d/K[0−9][0−9]$suffix
previous_start=/etc/rc$previous.d/S[0−9][0−9]$suffix

# If there is a start script in the previous
# level
```

```
# and _no_ stop script in this level, we don't
# have to re−start the service.
[ −f $previous_start ] && [ ! −f $stop ] && continue
fi

case "$runlevel" in
0|6)
startup $i stop
;;
*)
startup $i start
;;
esac
done
fi

# End /etc/init.d/rc
```

The second main bootscript is the `rcS` script. Create a new file `/etc/init.d/rcS` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/rcS

runlevel=S
prevlevel=N
umask 022
export runlevel prevlevel

trap ":" INT QUIT TSTP

for i in /etc/rcS.d/S??*
do
[ ! −f  "$i" ] && continue;
$i start
done

# End /etc/init.d/rcS
```

Creating the reboot script

Create a new file `/etc/init.d/reboot` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/reboot

echo "System reboot in progress..."

/sbin/reboot -d -f -i

# End /etc/init.d/reboot
```

Creating the halt script

Create a new file `/etc/init.d/halt` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/halt

/sbin/halt –d –f –i –p

# End /etc/init.d/halt
```

Creating the mountfs script

Create a new file `/etc/init.d/mountfs` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/mountfs

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

echo -n "Remounting root file system in read-write mode..."
/bin/mount -n -o remount,rw /
check_status

echo > /etc/mtab
/bin/mount -f -o remount,rw /

echo -n "Mounting proc file system..."
/bin/mount proc
check_status

# End /etc/init.d/mountfs
```

Creating the umountfs script

Create a new file `/etc/init.d/umountfs` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/umountfs

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

echo –n "Deactivating swap..."
/sbin/swapoff –a
check_status

echo –n "Unmounting file systems..."
/bin/umount –a –r
check_status

# End /etc/init.d/umountfs
```

Creating the sendsignals script

Create a new file `/etc/init.d/sendsignals` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/sendsignals

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

echo -n "Sending all processes the TERM signal..."
/sbin/killall5 -15
check_status

echo -n "Sending all processes the KILL signal..."
/sbin/killall5 -9
check_status

# End /etc/init.d/sendsignals
```

Creating the checkroot script

Create a new file `/etc/init.d/checkroot` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/checkroot

echo −n "Activating swap..."
/sbin/swapon −a

if [ −f /fastboot ]
then
  echo "Fast boot, no file system check"

  /bin/mount −n −o remount,ro /
  if [ $? = 0 ]
  then

    if [ −f /forcecheck ]
    then
      force="−f"
    else
      force=""
    fi

    echo "Checking root file system..."
    /sbin/fsck $force −a /

    if [ $? −gt 1 ]
    then
      echo
      echo "fsck failed. Please repair your file system manually by"
      echo "running /sbin/fsck without the −a option"
      echo
      echo "Please note that the file system is currently mounted in"
              echo "read−only mode."
      echo
        echo "I will start sulogin now. CTRL+D will reboot your system."
      echo
              /sbin/sulogin
              /sbin/reboot −f
    fi
  else
    echo "Cannot check root file system because it is not mounted in"
    echo "read−only mode."
  fi
fi
```

```
# End /etc/init.d/checkroot
```

Creating the sysklogd script

Create a new file /etc/init.d/sysklogd containing the following:

```
#!/bin/sh
# Begin /etc/init.d/sysklogd

check_status()
{
if [ $? = 0 ]
then
  echo "OK"
else
  echo "FAILED"
fi
}

case "$1" in
  start)
    echo −n "Starting system log daemon..."
    start−stop−daemon −S −q −o −x /usr/sbin/syslogd −− −m 0
    check_status

    echo −n "Starting kernel log daemon..."
    start−stop−daemon −S −q −o −x /usr/sbin/klogd
    check_status
    ;;

  stop)
    echo −n "Stopping kernel log daemon..."
    start−stop−daemon −K −q −o −p  /var/run/klogd.pid
    check_status

    echo −n "Stopping system log daemon..."
    start−stop−daemon −K −q −o −p /var/run/syslogd.pid
    check_status
    ;;

  reload)
    echo −n "Reloading system load daemon configuration file..."
    start−stop−daemon −K −q −o −s 1 −p /var/run/syslogd.pid
    check_status
    ;;

  restart)
    echo −n "Stopping kernel log daemon..."
    start−stop−daemon −K −q −o −p /var/run/klogd.pid
```

```
    check_status

    echo -n "Stopping system log daemon..."
    start-stop-daemon -K -q -o -p /var/run/syslogd.pid
    check_status

    sleep 1

    echo -n "Starting system log daemon..."
    start-stop-daemon -S -q -o -x /usr/sbin/syslogd -- -m 0
    check_status

    echo -n "Starting kernel log daemon..."
    start-stop-daemon -S -q -o -x /usr/sbin/klogd
    check_status
    ;;

  *)
    echo "Usage: $0 {start|stop|reload|restart}"
    exit 1
    ;;
esac

# End /etc/init.d/sysklogd
```

Setting up symlinks and permissions

Give these files the proper permissions and create the necessary symlinks by running the following commands:

```
root:~#   cd /etc/init.d
root:init.d#   chmod 755 rcS reboot halt mountfs umountfs
root:init.d#   chmod 755 sendsignals checkroot sysklogd
root:init.d#   cd ../rc0.d
root:rc0.d#   ln -s ../init.d/sysklogd K90sysklogd
root:rc0.d#   ln -s ../init.d/sendsignals S80sendsignals
root:rc0.d#   ln -s ../init.d/umountfs S90umountfs
root:rc0.d#   ln -s ../init.d/halt S99halt
root:rc0.d#   cd ../rc6.d
root:rc6.d#   ln -s ../init.d/sysklogd K90sysklogd
root:rc6.d#   ln -s ../init.d/sendsignals S80sendsignals
root:rc6.d#   ln -s ../init.d/umountfs S90umountfs
root:rc6.d#   ln -s ../init.d/reboot S99reboot
root:rc6.d#   cd ../rcS.d
root:rcS.d#   ln -s ../init.d/checkroot S05checkroot
root:rcS.d#   ln -s ../init.d/mountfs S10mountfs
root:rcS.d#   cd /etc/rc2.d
root:rc2.d#   ln -s ../init.d/sysklogd S03sysklogd
```

Creating the /etc/fstab file

In order for certain programs to be able to determine where certain partitions are supposed to be mounted by default, the /etc/fstab file is used. Create a new file `/etc/fstab` containing the following:

```
# Begin /etc/fstab

/dev/<LFS−partition designation> / ext2 defaults 0 1
/dev/<swap−partition designation> none swap sw 0 0
proc /proc proc defaults 0 0

# End /etc/fstab
```

Replace <LFS−partition designation> and <swap−partition designation> with the appropriate devices (/dev/hda5 and /dev/hda6 in my case).

Chapter 7. Setting up basic networking

Introduction

This chapter will setup basic networking. Although you might not be connected to a network, Linux software uses network functions anyway. We'll be installing at least the local loopback device and a network card as well if applicable. Also the proper bootscripts will be created so that networking will be enabled during boot time.

Installing network software

Installing Netkit−base

Install Netkit−base by running the following commands:

```
root:netkit-base-0.17...#   ./configure --prefix=/usr
root:netkit-base-0.17...#   make
root:netkit-base-0.17...#   make install
root:netkit-base-0.17...#   cd etc.sample
root:netkit-base-0.17.../etc.sample#   cp services protocols
/etc
```

Installing Net−tools

Install Net−tools by running the following commands:

```
root:net-tools-1.54#   make
root:net-tools-1.54#   make install
```

Creating network boot scripts

Creating the /etc/init.d/localnet bootscript

Create a new file `/etc/init.d/localnet` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/localnet

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}
echo −n "Setting up loopback device..."
/sbin/ifconfig lo 127.0.0.1
check_status

echo −n "Setting up hostname..."
/bin/hostname −−file /etc/hostname
check_status

# End /etc/init.d/localnet
```

Setting up permissions and symlink

Set the proper file permissions and create the necessary symlink by running the following commands:

```
root:~#     cd /etc/init.d
root:init.d#    chmod 755 /etc/init.d/localnet
root:init.d#    cd ../rcS.d
root:rcS.d#    ln −s ../init.d/localnet S03localnet
```

Creating the /etc/hostname file

Create a new file `/etc/hostname` and put the hostname in it. This is not the FQDN (Fully Qualified Domain Name). This is the name you wish to call your computer in a network. An example:

```
lfs
```

The file must not contain empty lines or spaces after the hostname. Don't press enter either when you entered the name.

Creating the /etc/hosts file

If you want to configure a network card, you have to decide on the IP−address, FQDN and possible aliases for use in the /etc/hosts file. An example is:

```
<my−IP> myhost.mydomain.org aliases
```

Make sure the IP−address is in the private network IP−address range. Valid ranges are:

```
Class Networks
A     10.0.0.0
B     172.16.0.0 through 172.31.0.0
C     192.168.0.0 through 192.168.255.0
```

A valid IP address could be 192.168.1.1. A valid FQDN for this IP could be www.linuxfromscratch.org

If you're not going to use a network card, you still need to come up with a FQDN. This is necessary for programs like Sendmail to operate correctly (in fact; Sendmail won't run when it can't determine the FQDN).

If you don't configure a network card, create a new file `/etc/hosts` containing:

```
# Begin /etc/hosts (no network card version)
```

```
127.0.0.1 www.linuxfromscratch.org <contents of /etc/hostname> localhost

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

If you do configure a network card, create a new file `/etc/hosts` containing:

```
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
192.168.1.1 www.linuxfromscratch.org <contents of /etc/hostname>

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

Of course, change the 192.168.1.1 and www.linuxfromscratch.org to your own liking (or requirements if you are assigned an IP−address by a network/system administrator and you plan on connecting this machine to that network).

Creating the /etc/init.d/ethnet file

This section only applies if you are going to configure a network card. If you're not, skip this section.

Create a new file `/etc/init.d/ethnet` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/ethnet

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

IPADDR="209.83.245.12" # Replace with your own IP address
NETMAKSK="255.255.255.0" # Replace with your own Netmask
BROADCAST="209.83.245.255" # Replace with your own Broadcast addr.
GATEWAY="209.83.245.1" # Replace with your own Gateway address
```

```
echo -n "Setting up eth0..."
/sbin/ifconfig eth0 $IPADDR broadcast $BROADCAST netmask $NETMASK
check_status

echo "Adding default gateway..."
/sbin/route add default gw $GATEWAY metric 1
check_status

# End /etc/init.d/ethnet
```

Setting up permissions and symlink

Set the proper file permissions and create the necessary symlink by running the following commands:

```
root:~#    cd /etc/init.d
root:init.d#    chmod 755 /etc/init.d/ethnet
root:init.d#    cd ../rc2.d
root:rc2.d#    ln -s ../init.d/ethnet S10ethnet
```

Chapter 8. Making the LFS system bootable

Introduction

This chapter will make LFS bootable. This chapter deals with building a new kernel for our new LFS system and adding the proper entries to LILO so that you can select to boot the LFS system at the LILO: prompt.

Installing a kernel

A kernel is the heart of a Linux system. We could use the kernel image from our normal system, but we might as well compile a new kernel from the most recent kernel sources available.

Building the kernel involves a few steps: configuring it and compiling it. There are a few ways to configure the kernel. If you don't like the way this book does it, read the `README` file and find out what your other options are. Run the following commands to build the kernel:

```
root:linux#   make mrproper
root:linux#    make menuconfig
root:linux#    make dep
root:linux#    make bzImage
root:linux#    cp arch/i386/boot/bzImage /boot/lfskernel
root:linux#   cp System.map /boot
```

Adding an entry to LILO

In order to being able to boot from this partition, we need to update our /etc/lilo.conf file. Add the following lines to lilo.conf:

```
image=/boot/lfskernel
  label=lfs
  root=<partition>
  read-only
```

<partition> must be replaced by your partition's designation (which would be /dev/hda5 in my case).

Now update the boot loader by running:

```
root:~#    lilo
```

Testing the system

Now that all software has been installed, bootscripts have been written and the local network is setup, it's time for you to reboot your computer and test these new scripts to verify that they actually work. You first want to execute them manually from the /etc/init.d directory so you can fix the most obvious problems (typos, wrong paths and such). When those scripts seem to work just fine manually they should also work during a system start or shutdown. There's only one way to test that. Shutdown your system with shutdown –r now and reboot into LFS. After the reboot you will have a normal login prompt like you have on your normal Linux system (unless you use XDM or some sort of other Display Manger (like KDM – KDE's version of XDM).

When you are at the login prompt, login as user root and when asked for a password just press enter. The first thing you want to do is set a password for user root by running the following command:

```
:root:~#   passwd
```

At this point your basic LFS system is ready for use. Everything else that follows now is optional, so you can skip packages at your own discretion. But do keepein mind that if you skip packages (especially libraries) you can break dependencies of other packages. For example, when the Lynx browser is installed, the zlib library is installed as well. You can decide to skip the zlib library, but this library isn't used by Lynx alone. Other packages require this library too. The same may apply to other libraries and programs.

III. Part III – Installation of a basic system on Apple PowerPC systems

Table of Contents

Chapter 9. Packages you need to download

Below is a list of all the packages you need to download for building the basic system. The version numbers printed correspond to versions of the software that is known to work and which this book is based on. If you experience problems which you can't solve yourself, download the version that is assumed in this book (in case you download a newer version).

Please note that this list used to be ordered on usage, meaning that the first package mentioned in this list was the first package used in this book. That's no longer the case because several chapters have been moved around, so that doens't apply. I didn't have the time to re−order this list in this development release. The next release will have this list ordered again.

- Sysvinit (2.78): ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/
- Bash (2.04): ftp://ftp.gnu.org/gnu/bash
- Linux Kernel (2.2.14): ftp://ftp.kernel.org/pub/linux/kernel/
- Kernel USB patch: 216.22.163.20/usb−2.3.50−1−for−2.2.14.diff.gz
- Binutils (2.9.5.0.37): ftp://ftp.varesearch.com/pub/support/hjl/binutils/
- Bzip2 (0.9.5d): http://sourceware.cygnus.com/bzip2/
- Diff Utils (2.7): ftp://ftp.gnu.org/gnu/diffutils/
- File Utils (4.0): ftp://ftp.gnu.org/gnu/fileutils/
- GCC (2.95.2): ftp://ftp.gnu.org/gnu/gcc/
- Glibc (2.1.3): ftp://ftp.gnu.org/gnu/glibc/
- Glibc−crypt (2.1.3): ftp://ftp.gwdg.de/pub/linux/glibc/
- Glibc−linuxthreads (2.1.3): ftp://ftp.gnu.org/gnu/glibc/
- Glibc−patch: ftp://216.22.163.20/glibc−2.1.3−ctype.patch

Grep (2.4.2): ftp://ftp.gnu.org/gnu/grep/

- Gzip (1.2.4a): ftp://ftp.gnu.org/gnu/gzip/
- Make (3.78.1): ftp://ftp.gnu.org/gnu/make/
- Ed (0.2): ftp://ftp.gnu.org/gnu/ed/
- Patch (2.5.4): ftp://ftp.gnu.org/gnu/patch/
- Sed (3.02): ftp://ftp.gnu.org/gnu/sed/
- Shell Utils (2.0): ftp://ftp.gnu.org/gnu/sh−utils/
- Tar (1.13): ftp://ftp.gnu.org/gnu/tar/
- Text Utils (2.0): ftp://ftp.gnu.org/gnu/textutils/
- Util Linux (2.10h): ftp://ftp.win.tue.nl/pub/linux/utils/util−linux/
- Pmac Utils((1.1.1): ftp://216.22.163.20/pmac−utils−1.1.1−patched.tar.gz
- Bison (1.28): ftp://ftp.gnu.org/gnu/bison/
- Mawk (1.3.3) ftp://ftp.whidbey.net/pub/brennan/
- Find Utils (4.1): ftp://ftp.gnu.org/gnu/findutils/
- Termcap (1.3): ftp://ftp.gnu.org/gnu/termcap/
- Ncurses (4.2): ftp://ftp.gnu.org/gnu/ncurses/
- Less (340): ftp://ftp.gnu.org/gnu/less/
-

Perl (5.6.0): http://www.perl.com

- M4 (1.4): ftp://ftp.gnu.org/gnu/m4/
- Texinfo (4.0): ftp://ftp.gnu.org/gnu/texinfo/
- Autoconf (2.13): ftp://ftp.gnu.org/gnu/autoconf/
- Automake (1.4): ftp://ftp.gnu.org/gnu/automake/
- Flex (2.5.4a): ftp://ftp.gnu.org/gnu/flex/
- E2fsprogs (1.18): ftp://tsx−11.mit.edu/pub/linux/packages/ext2fs/
- File (3.26): http://www.linuxfromscratch.org/download/file−3.26−lfs.tar.gz
- Groff (1.15): ftp://ftp.gnu.org/gnu/groff/
- Ld.so (1.9.9): ftp://tsx−11.mit.edu/pub/linux/packages/GCC/
- Libtool (1.3.4): ftp://ftp.gnu.org/gnu/libtool/
- Linux86 (0.14.3): http://www.linuxfromscratch.org/download/linux86−0.14.3−lfs.tar.gz
- Shadow Password Suite (19990827): ftp://piast.t19.pwr.wroc.pl/pub/linux/shadow/
- Man (1.5h1): ftp://ftp.win.tue.nl/pub/linux−local/utils/man/
- Modutils (2.3.9): ftp://ftp.ocs.com.au/pub/modutils/
- Procinfo (17): ftp://ftp.cistron.nl/pub/people/svm/
- Procps (2.0.6): ftp://people.redhat.com/johnsonm/procps/
-

Psmisc (19): ftp://lrcftp.epfl.ch/pub/linux/local/psmisc/

- Start−stop−daemon (0.4.1): http://www.linuxfromscratch.org/download/ssd−0.4.1.tar.gz
- Sysklogd (1.3.31): ftp://sunsite.unc.edu/pub/Linux/system/daemons/
- Vim−rt + Vim−src (5.6): ftp://ftp.vim.org/pub/editors/vim/unix/

Chapter 10. Preparing a new partition

Introduction

In this chapter the partition that is going to host the LFS system is going to be prepared. A new partition will be created, an ext2 file system will be created on it and the directory structure will be created. When this is done, we can move on to the next chapter and start building a new Linux system from scratch.

Creating a new partition

Before we can build our new Linux system, we need to have an empty Linux partition on which we can build our new system. I recommend a partition size of at least 5 00 MB. You can get away with around 250MB for a bare system with no extra bells and whistles (such as software for emailing, networking, Internet, X Window System and such). If you already have a Linux Native partition available, you can skip this subsection.

Start the pdisk program (or some other fdisk program you prefer) with the appropriate hard disk as the option (like /dev/shda if you want to create a new partition on the first SCSI disk). The partition that is available for partitioning is called *Apple_Free_Space*. To create a linux capable partition in that free space, type c followed by the partition designation of the free space p<n>, the size in MB of the desired partition <size>M and the name of the partition <name>. The example below creates a 1.8 GB partition name root starting at the beginning of the free space designated as partition 6: *c p6 1800M root*

Mounting the new partition

Now that we have created the ext2 file system, it is ready for use. All we have to do to be able to access it (as in reading from and writing date to it) is mounting it. If you mount it under /mnt/lfs, you can access this partition by going to the /mnt/lfs directory and then do whatever you need to do. This document will assume that you have mounted the partition on a subdirectory under /mnt. It doesn't matter which directory you choose (or you can use just the /mnt directory as the mount point) but this book will assume /mnt/lfs in the commands it tells you to execute.

Create the /mnt/lfs directory by runnning:

```
root:~#   mkdir -p /mnt/lfs
```

Now mount the LFS partition by running:

```
root:~#   mount /dev/xxx /mnt/lfs
```

Replace "xxx" by your partition's designation.

This directory (/mnt/lfs) is the $LFS variable you have read about earlier. So if you read somewhere to "cp inittab $LFS/etc" you actually will type "cp inittab /mnt/lfs/etc".

Creating directories

Let's create the directory tree on the LFS partition according to the FHS standard which can be found at http://www.pathname.com/fhs/. Issuing the following commands will create the necessary directories:

```
root:~#   cd $LFS
root:lfs#   mkdir bin boot dev etc home lib mnt proc root
sbin tmp usr var
root:lfs#   cd $LFS/usr
root:usr#   mkdir bin include lib local sbin share src
root:usr#   ln -s share/man man
root:usr#   ln -s share/doc doc
root:usr#   ln -s share/info info
root:usr#   ln -s ../etc etc
root:usr#   ln -s ../var var
root:usr#   cd $LFS/usr/share
root:share#   mkdir dict doc info locale man nls misc
terminfo zoneinfo
root:share#   cd $LFS/usr/share/man
root:man#   mkdir man1 man2 man3 man4 man5 man6 man7 man8
root:man#   cd $LFS/var
root:var#   mkdir lock log run spool tmp
```

Normally directories are created with permission mode 755, which isn't desired for all directories. I haven't checked the FHS if they suggest default modes for certain directories, so I'll just change the modes for two directories. The first change is a mode 0750 for the $LFS/root directory. This is to make sure that not just everybody can enter the /root directory (the same you would do with /home/username directories). The second change is a mode 1777 for the $LFS/tmp directory. This way every user can write stuff to the /tmp directory if they need to. The sticky (1) bit makes sure users can't delete other user's file which they normally can do because the directory is set in such a way that every body (owner, group, world) can write to that directory.

```
root:~#   cd $LFS
root:lfs#   chmod 0750 root
root:lfs#   chmod 1777 tmp
```

Now that the directories are created, copy the source files you have downloaded in chapter 3 to some subdirectory under $LFS/usr/src (you will need to create this subdirectory yourself).

Copying the /dev directory

We can create every single file that we need to be in the $LFS/dev directory using the mknod command, but that just takes up a lot of time. I choose to just simply copy the current /dev directory to the $LFS partition. Use this command to copy the entire directory while preserving original rights, symlinks and ownerships:

```
root:~# cp -av /dev $LFS
root:~#    chown root $LFS/dev/*
```

I'm aware that this isn't the best way to create the files. I know of a MAKEDEV script but I choose not to use it. I'm actually waiting for the 2.4 Linux kernel to be released. The kernel has a stable version of the devfs which this book will use in the future. Devfs is a dynamic file system which makes the static files in /dev obsolete. You mount the dev file system to a mount point (kind of like the way the proc file system works) and the kernel will create the files in /dev you need on−the−fly. So the waiting is for the next stable kernel to be released.

Chapter 11. Installing basic system software

How and why things are done

In this chapter we will install all the software that belongs to a basic Linux system. After you're done with this chapter you have a fully working Linux system. The remaining chapters deal with optional issues such as setting up networking, Internet servers + clients (telnet, ftp, http, email), setting up Internet itself and the X Window System. You can skip chapters at your own discretion. If you don't plan on going online with the LFS system there's little use to setup Internet for example.

This chapter is devided in two chunks. The first part installs a few necessary programs on the LFS system. These programs are needed to install the rest of the programs that belong to a basic system. When the first part is done, we will enter a chroot'ed environment. This means that we start a shell with $LFS as the root directory (instead of the usual / directory as the root directory). This has the same effect as rebooting the computer into the LFS system, but this way we don't have to reboot. If something goes wrong, you don't need to reboot back in the normal Linux system to fix whatever you need to fix. You just open a new shell on a virtual console, or start a new xterm and you can do what you need to do.

The software in the first part will be linked statically. These programs will be re–installed in the second part and linked dynamically. The reason for the static version first is that there is a chance that our normal Linux system and our LFS system–to–be don't use the same C Library versions. If the programs in the first part are linked against an older C library version, those program might not work too well on the LFS system.

The key to learn what makes Linux tick is to know exactly what packages are used for and why you or the system needs them. In depth descriptions of every package is provided in Appendix A.

About debugging symbols

Every program and library is by default compiled with debugging symbols. This means you can run a program or library through a debugger and the debugger's output will be more user friendly. These debugging symbols also enlarge the program or library significantly. This document will not install software without debugging symbols (as I don't know if the majority of readers do or do not debug software). In stead, you can remove those symbols manually if you want with the strip program.

To remove debugging symbols from a binary (must be an a.out or ELF binary) run **`strip --strip-debug filename`** You can use wild cards if you need to strip debugging symbols from multiple files (use something like strip −−strip−debug $LFS/usr/bin/*).

Before you wonder if these debugging symbols would make a big difference, here are some statistics:

- A static Bash binary with debugging symbols: 2.3MB
- A static Bash binary without debugging symbols: 645KB
- A dynamic Bash binary with debugging symbols: 1.2MB
- A dynamic Bash binary without debugging symbols: 478KB
- $LFS/lib and $LFS/usr/lib (glibc and gcc files) with debugging symbols: 87MB
- $LFS/lib and $LFS/usr/lib (glibc and gcc files) without debugging symbols: 16MB

Sizes may vary depending on which compiler was used and which C library version was used to link dynamic programs against, but your results will be similar if you compare programs with and without debugging symbols. After I was done with this chapter and stripped all debugging symbols from all LFS binaries and libraries I regained a little over 102 MB of disk space. Quite the difference. The difference would be even greater when I would do this at the end of this book when everything is installed.

Preparing the LFS system for installing basic system software

Installing Bash

Install Bash by running the following commands:

```
root:bash-2.04#   ./configure --enable-static-link
root:bash-2.04#   make
root:bash-2.04#   make -e prefix=$LFS/usr install
root:bash-2.04#   mv $LFS/usr/bin/bash $LFS/bin
root:bash-2.04#   cd $LFS/bin
root:bin#   ln -s bash sh
```

Installing Binutils

Install Binutils by running the following commands:

```
root:binutils-2.9.5.0.37#   ./configure --prefix=/usr
root:binutils-2.9.5.0.37#   make -e LDFLAGS=-all-static
root:binutils-2.9.5.0.37a  make -e prefix=$LFS/usr install
```

Installing Bzip2

Before we can install Bzip2 we need to modify the `Makefile` file. Open the `Makefile` file in a text editor and find the lines that start with *$(CC) $(CFLAGS) -o*

Replace those parts with: *$(CC) $(CFLAGS) $(LDFLAGS) -o*

Now install Bzip2 by running the following commands:

```
root:bzip2-0.9.5d#   make -e LDFLAGS=-static
root:bzip2-0.9.5d#   make -e PREFIX=$LFS/usr install
root:bzip2-0.9.5d#   cd $LFS/usr/bin
root:bin#   mv bunzip2 bzip2 $LFS/bin
```

Installing Diffutils

Install Diffutils by running the following commands:

```
root:diffutils-2.7#   ./configure --prefix=/usr
root:diffutils-2.7#   make -e LDFLAGS=-static
root:diffutils-2.7#   make -e prefix=$LFS/usr install
```

This package is known to cause static link problems on certain platforms. If you're having trouble compiling this package as well, you can download a patch from http://www.linuxfromscratch.org/download/diffutils–2.7.patch.gz

Install this patch by running the following command:

```
root:diffutils-2.7#   patch -Np1 -i ../diffutils-2.7.patch
```

Now recompile the package using the same commands as above.

Installing Fileutils

Install Fileutils by running the following commands:

```
root:fileutils-4.0#   ./configure --disable-nls --prefix=/usr
root:fileutils-4.0#   make -e LDFLAGS=-static
root:fileutils-4.0#   make -e prefix=$LFS/usr install
root:fileutils-4.0#   cd $LFS/usr/bin
root:bin#   mv chgrp chmod chown cp dd df ln $LFS/bin
root:bin#   mv ls mkdir mknod mv rm rmdir sync $LFS/bin
```

Installing GCC on the normal system if necessary

In order to compile Glibc−2.1.3 later on you need to have gcc−2.95.2 installed. Although any GCC version above 2.8 would do, 2.95.2 is the highly recommended version to use. Many glibc−2.0 based systems have gcc−2.7.2.3 installed and you can't compile glibc−2.1.3 with that compiler. Many glibc−2.1 based systems have egcs−2.95.x installed and that version doesn't work too well either (sometimes it works fine, sometimes it doesn't depending on various circumstances).

To find out whether your system uses gcc−2.95.2 or not, run the following command:

```
root:~#   gcc --version
```

If you normal Linux system does not have gcc−2.95.2 installed you need to install it now. We won't replace the current compiler on your system, but instead we will install gcc in a separate directory (/usr/local/gcc2952). This way no binaries or header files will be replaced.

After you unpacked the gcc−2.95.2 archive don't enter the newly created gcc−2.95.2 directory but stay in the $LFS/usr/src directory. Install GCC by running the following commands:

```
root:src#   mkdir $LFS/usr/src/gcc-build
root:src#   cd $LFS/usr/src/gcc-build
root:gcc-build#   ../gcc-2.95.2/configure
--prefix=/usr/local/gcc2952 \
> --with-local-prefix=/usr/local/gcc2952  \
> --with-gxx-include-dir=/usr/local/gcc2952/include/g++  \
> --enable-shared --enable-languages=c,c++
root:gcc-build#   make bootstrap
root:gcc-build#   make install
```

Installing GCC on the LFS system

After you unpacked the gcc−2.95.2 archive don't enter the newly created gcc−2.95.2 directory but stay in the $LFS/usr/src directory. Install GCC by running the following commands:

```
root:src#   mkdir $LFS/usr/src/gcc-build
root:src#   cd $LFS/usr/src/gcc-build
root:gcc-build#   ../gcc-2.95.2/configure \
```

```
> --prefix=/usr --with-local-prefix=/usr \
> --with-gxx-include-dir=/usr/include/g++ \
> --enable-languages=c,c++ --disable-nls
root:gcc-build#   make -e LDFLAGS=-static bootstrap
root:gcc-build#   make -e prefix=$LFS/usr
local_prefix=$LFS/usr \
gxx_include_dir=$LFS/usr/include/g++ \
> install
```

Creating necessary symlinks

The system needs a few symlinks to ensure every program is able to find the compiler and the pre−processor. Some programs run the cc program, others run the gcc program. Some programs expect the cpp program in /lib and others expect to find it in /usr/bin. Create those symlinks by running:

```
root:~#   cd $LFS/lib
root:lib#   ln -s ../usr/lib/gcc-lib/<host>/2.95.2/cpp cpp
root:lib#   cd $LFS/usr/lib
root:lib#   ln -s gcc-lib/<host>/2.95.2/cpp cpp
root:lib#   cd $LFS/usr/bin
root:bin#   ln -s gcc cc
```

Replace <host> with the directory where the gcc−2.95.2 files are installed (which is i686−unknown−linux in my case).

Installing Glibc

A note on the glibc−crypt package

An excerpt from the README file that is distributed with the glibc−crypt package:

The add−on is not included in the main distribution of the GNU C library because some governments, most notably those of France, Russia, and the US, have very restrictive rules governing the distribution and use of encryption software. Please read the node "Legal Problems" in the manual for more details.

In particular, the US does not allow export of this software without a licence, including via the Internet. So please do not download it from the main FSF FTP site at ftp.gnu.org if you are outside the US. This software was completely developed outside the US.

"This software" refers to the glibc−crypt package at ftp://ftp.gwdg.de/pub/linux/glibc/. This law only affects people who don't live in the US. It's not prohibited to import DES software, so if you live in the US you can import the file safely from Germany without breaking cryptographic laws. This law is changing lately and I don't know what the status of it is at the moment. Better be safe than sory.

Installing Glibc

Copy the Glibc−crypt and Glibc−linuxthreads archives into the unpacked glibc directory. Copy the glibc−2.1.3−ctype.patch file to $LFS/usr/src

Unpack the glibc−crypt and glibc−linuxthreads archives there, but don't enter the created directories. Just unpack and leave it with that.

A few default parameters of Glibc need to be changed, such as the directory where the shared libraries are supposed to be installed in and the directory that contains the system configuration files. For this purpose you need to create the `$LFS/usr/src/glibc-build` directory and in that directory you create a new file `configparms` containing:

```
# Begin configparms

slibdir=/lib
sysconfdir=/etc

# End configparms
```

Change to the `$LFS/usr/src/glibc-2.1.3` directory and install Glibc by running the following commands if your system already had a suitable GCC version installed:

```
root:glibc-2.1.3#   patch -p1 < ../glibc-2.1.3-ctype.patch
root:glibc-2.1.3#   cd ../glibc-build
root:glibc-build#   ../glibc-2.1.3/configure \
> --prefix=/usr --enable-add-ons
root:glibc-build#   make
root:glibc-build#   make install_root=$LFS install
```

Change to the `$LFS/usr/src/glibc-build` directory and install Glibc by running the following command if your system did not already have a suitable GCC version installed and you just installed GCC−2.95.2 on your normal Linux system a little while ago:

```
root:glibc-2.1.3#   patch -p1 < ../glibc-2.1.3-ctype.patch
root:glibc-2.1.3#   cd ../glibc-build
```

```
root:glibc-build#   CC=/usr/local/gcc2952/bin/gcc \
> ../glibc-2.1.3/configure --prefix=/usr \
> --enable-add-ons
root:glibc-build#   make
root:glibc-build#   make install_root=$LFS install
```

Copying old NSS library files

If your normal Linux system runs glibc−2.0, you need to copy the NSS library files to the LFS partition. Certain statically linked programs still depend on the NSS library, especially programs that need to lookup usernames,userid's and groupid's. You can check which C library version your normal Linux system uses by running:

```
root:~# ls /lib/libc*
```

Your system uses glib−2.0 if there is a file that looks like *libc−2.0.7.so*

Your system uses glibc−2.1 if there is a file that looks like *libc−2.1.3.so*

Of course, the micro version number can be different (you could have libc−2.1.2 or libc−2.1.1 for example).

If you have a libc−2.0.x file copy the NSS library files by running:

```
root:~# cp −av /lib/libnss*  $LFS/lib
```

There are a few distributions that don't have files from which you can see which version of the C Library it is. If that's the case, it will be hard to determine which C library version you exactly have. Try to obtain this information using your distribution's installation tool. It often says which version it has available. If you can't figure out at all which C Library version is used, then copy the NSS files anyway and hope for the best. That's the best advise I can give I'm afraid.

Installing Grep

Install Grep by running the following commands:

```
root:grep-2.4.2#   ./configure --prefix=/usr --disable-nls
```

```
root:grep-2.4.2#   make -e LDFLAGS=-static
root:grep-2.4.2#   make -e prefix=$LFS/usr install
```

This package is known to cause static linking problems on certain platforms. If you're having trouble compiling this package as well, you can download a patch from http://www.linuxfromscratch.org/download/grep–2.4.2.patch.gz

Install this patch by running the following command:

```
root:grep-2.4.2#   patch -Np1 -i ../grep-2.4.2.patch
```

Now recompile the package using the same commands as above.

Installing Gzip

Install Gzip by running the following commands:

```
root:gzip-1.2.4a#   ./configure --prefix=/usr
root:gzip-1.2.4a#   make -e LDFLAGS=-static
root:gzip-1.2.4a#   make -e prefix=$LFS/usr install
root:gzip-1.2.4a#   cd $LFS/usr/bin
root:bin#   mv gunzip gzip $LFS/bin
```

This package is known to cause compilation problems on certain platforms. If you're having trouble compiling this package as well, you can download a fixed package from http://www.linuxfromscratch.org/download/gzip–1.2.4a.patch.gz

Install this patch by running the following command:

```
root:gzip-1.2.4a#   patch -Np1 -i ../gzip-1.2.4a.patch.gz
```

Now recompile the package using the same commands as above.

Installing Make

Install Make by running the following commands:

```
root:make-3.78.1#   ./configure --prefix=/usr --disable-nls
root:make-3.78.1#   make -e LDFLAGS=-static
root:make-3.78.1#   make -e prefix=$LFS/usr install
```

Installing Sed

Install Sed by running the following commands:

```
root:sed-3.02#   ./configure --prefix=/usr
root:sed-3.02#   make -e LDFLAGS=-static
root:sed-3.02#   make -e prefix=$LFS/usr install
root:sed-3.02#   mv $LFS/usr/bin/sed $LFS/bin
```

This package is known to cause static linking problems on certain platforms. If you're having trouble compiling this package as well, you can download a patch from http://www.linuxfromscratch.org/download/sed–3.02.patch.gz

Install this patch by running the following command:

```
root:sed-3.02#   patch -Np1 -i ../sed-3.02.patch.gz
```

Now recompile the package using the same commands as above.

Installing Shellutils

Install Shellutils by running the following commands:

```
root:sh-utils-2.0#   ./configure --prefix=/usr --disable-nls
```

```
root:sh-utils-2.0#   make -e LDFLAGS=-static
root:sh-utils-2.0#   make -e prefix=$LFS/usr install
root:sh-utils-2.0#   cd $LFS/usr/bin
root:/bin#   mv date echo false pwd stty $LFS/bin
root:bin#   mv su true uname hostname $LFS/bin
```

Installing Tar

Install Tar by running the following commands:

```
root:tar-1.13#   ./configure --prefix=/usr --disable-nls
root:tar-1.13#   make -e LDFLAGS=-static
root:tar-1.13#   make -e prefix=$LFS/usr install
root:tar-1.13#   mv $LFS/usr/bin/tar $LFS/bin
```

Installing Textutils

Install Textutils by running the following commands:

```
root:textutils-2.0#   ./configure --prefix=/usr --disable-nls
root:textutils-2.0#   make -e LDFLAGS=-static
root:textutils-2.0#   make -e prefix=$LFS/usr install
root:textutils-2.0#   mv $LFS/usr/bin/cat $LFS/bin
```

Creating passwd and group files

Create a new file `$LFS/etc/passwd` containing the following:

```
root::0:0:root:/root:/bin/bash
```

Create a new file `$LFS/etc/group` containing the following:

```
root::0:
```

Installing basic system software

The installation of all the software is pretty straightforward and you'll think it's so much easier and shorter to give the generic installation instructions for each package and only explain how to install something if a certain package requires an alternate installation method. Although I agree with you on that, I, however, choose to give the full instructions for each and every package. This is simply to avoid any possible confusion and errors.

Entering the chroot'ed environment

It's time to enter our chroot'ed environment now in order to install the rest of the software we need.

Enter the following commands to setup the chroot'ed environment. From this point on there's no need to use the $LFS variable anymore, because everything you do will be restricted to the LFS partition (since / is actually /mnt/xxx but the shell doesn't know that).

```
root:~#   cd $LFS/root
root:root#   chroot $LFS bash --login
```

Now that we are inside a chroot'ed environment, we can continue to install all the basic system software. Make sure you execute all the following commands in this chapter from within the chroot'ed environment.

Installing Ed

Install Ed by running the following commands:

```
root:/usr/src/ed-0.2#   ./configure --prefix=/usr
root:/usr/src/ed-0.2#   make
root:/usr/src/ed-0.2#   make install
```

Installing Patch

Install Patch by running the following commands:

```
root:patch-2.5.4#   ./configure --prefix=/usr
root:patch-2.5.4#   make
root:patch-2.5.4#   make install
```

Installing GCC

After you unpacked the gcc−2.95.2 archive don't enter the newly created gcc−2.95.2 directory but stay in the /usr/src directory. Install GCC by running the following commands:

```
root:src#   mkdir /usr/src/gcc-build
root:src#   cd /usr/src/gcc-build
root:gcc-build#   ../gcc-2.95.2/configure \
> --prefix=/usr --with-local-prefix=/usr \
> --with-gxx-include-dir=/usr/include/g++ \
> --enable-shared --enable-languages=c,c++
root:gcc-build#   make bootstrap
root:gcc-build#   make install
```

Installing Bison

Install Bison by running the following commands:

```
root:bison-1.28#   ./configure --prefix=/usr
--datadir=/usr/share/bison
root:bison-1.28#   make
root:bison-1.28#   make install
```

Installing Mawk

Install Mawk by running the following commands:

```
root:mawk-1.3.3#   ./configure
root:mawk-1.3.3#   make
root:mawk-1.3.3#   make -e BINDIR=/usr/bin
```

```
MANDIR=/usr/share/man/man1 install
root:mawk-1.3.3#   cd /usr/bin
root:in#   ln -s mawk awk
```

Installing Findutils

Install Findutils by running the following commands:

```
root:findutils-4.1#   ./configure --prefix=/usr
root:findutils-4.1#   make
root:findutils-4.1#   make install
```

This package is known to cause compilation problem. If you're having trouble compiling this package as well, you can download a patch from http://www.linuxfromscratch.org/download/findutils–4.1.patch.gz

Install this patch by running the following command:

```
root:findutils-4.1#   patch -Np1 -i ../findutils-4.1.patch.gz
```

Now recompile the package using the same commands as above.

Installing Termcap

Install Termcap by running the following commands:

```
root:termcap-1.3#   ./configure --prefix=/usr
root:termcap-1.3#   make
root:termcap-1.3#   make install
```

Installing Ncurses

Install Ncurses by running the following commands:

```
root:ncurses-4.2#   ./configure --prefix=/usr --with-shared
root:ncurses-4.2#   make
root:ncurses-4.2#   make install
```

Installing Less

Install Less by running the following commands:

```
root:less-340#   ./configure --prefix=/usr
root:less-340#   make
root:less-340#   make install
root:less-340#   mv /usr/bin/less /bin
```

Installing Perl

Install Perl by running the following commands:

```
root:perl-5.6.0#   ./Configure
root:perl-5.6.0#   make
root:perl-5.6.0#   make test
root:perl-5.6.0#   make install
```

Note that you have to change the installation path to `/usr` yourself. The Perl installation defaults to the /usr/local/subdir

Also note that a few tests during the make test phase will fail because we don't have network support installed yet.

Installing M4

Install M4 by running the following commands:

```
root:m4-1.4#   ./configure --prefix=/usr
root:m4-1.4#   make
root:m4-1.4#   make install
```

Installing Texinfo

Install Texinfo by running the following commands:

```
root:texinfo-4.0#   ./configure --prefix=/usr
root:texinfo-4.0#   make
root:texinfo-4.0#   make install
```

Installing Autoconf

Install Autoconf by running the following commands:

```
root:autoconf-2.13#   ./configure --prefix=/usr
root:autoconf-2.13#   make
root:autoconf-2.13#   make install
```

Installing Automake

Install Automake by running the following commands:

```
root:automake-1.4#   ./configure --prefix=/usr
root:automake-1.4#   make install
```

Installing Bash

Install Bash by running the following commands:

```
root:bash-2.04#   ./configure --prefix=/usr
root:bash-2.04#   make
root:bash-2.04#   make install
root:bash-2.04#   logout
root:root#   mv $LFS/usr/bin/bash $LFS/bin
root:root#   chroot $LFS bash --login
```

Installing Flex

Install Flex by running the following commands:

```
root:flex-2.5.4a#   ./configure --prefix=/usr
root:flex-2.5.4a#   make
root:flex-2.5.4a#   make install
```

Installing Binutils

Install Binutils by running the following commands:

```
root:binutils-2.9.5.0.37#   ./configure --prefix=/usr
root:binutils-2.9.5.0.37#   make
root:binutils-2.9.5.0.37#   make install
```

Installing Bzip2

Install Bzip2 by running the following commands:

```
root:bzip2-0.9.5d#   make
root:bzip2-0.9.5d#   make PREFIX=/usr install
root:bzip2-0.9.5d#   cd /usr/bin
root:bin#   mv bunzip2 bzip2 /bin
```

Installing Diffutils

Install Diffutils by running the following commands:

```
root:diffutils-2.7#   ./configure --prefix=/usr
root:diffutils-2.7#   make
root:diffutils-2.7#   make install
```

Installing Linux Kernel

We won't be compiling a new kernel image yet. We'll do that after we have finished the installation of the basic system software in this chapter. But because certain software need the kernel header files, we're going to unpack the kernel archive now and set it up so that we can compile package that need the kernel.

Create the kernel configuration file by running the following command:

```
root:linux#   yes "" | make config
```

Ignore the warning *Broken pipe* you might see at the end. Now run the following commands to set up all the dependencies correctly:

```
root:linux#   make dep
```

Now that that's done, we need to create the `$LFS/usr/include/linux` and the `$LFS/usr/include/asm` symlinks. Create them by running the following commands:

```
root:~#   cd /usr/include
root:include#   ln -s ../src/linux/include/linux linux
root:include#   ln -s ../src/linux/include/asm asm
```

Installing E2fsprogs

Install E2fsprogs by running the following commands:

```
root:e2fsprogs-1.18#    ./configure --prefix=/usr
--with-root-prefix=/
root:e2fsprogs-1.18#    make
root:e2fsprogs-1.18#    make install
root:e2fsprogs-1.18#    cd /usr/sbin
root:sbin#    mv *e2* *fs* mklost+found /sbin
```

Installing File

Install File by running the following commands:

```
root:file-3.26#   ./configure --prefix=/usr
root:file-3.26#   make
root:file-3.26#   make install
```

Installing Fileutils

Install Fileutils by running the following commands:

```
root:fileutils-4.0#   ./configure --prefix=/usr
root:fileutils-4.0#   make
root:fileutils-4.0#   make install
root:fileutils-4.0#   cd /usr/bin
root:bin#   mv chgrp chmod chown cp dd df ln /bin
root:bin#   mv ls mkdir mknod mv rm rmdir sync /bin
```

Installing Grep

Install Grep by running the following commands:

```
root::grep-2.4.2#   ./configure --prefix=/usr
root:grep-2.4.2#   make
root:grep-2.4.2#   make install
```

Installing Groff

Install Groff by running the following commands:

```
root:groff-1.15#   ./configure --prefix=/usr
root:groff-1.15#   make
root:groff-1.15#   make install
```

Installing Gzip

Install Gzip by running the following commands:

```
root:gzip-1.2.4a#   ./configure --prefix=/usr
root:gzip-1.2.4a#   make
root:gzip-1.2.4a#   make install
root:gzip-1.2.4a#   cd /usr/bin
root:bin#   mv gunzip gzip /bin
```

Installing Ld.so

Install Ld.so by running the following commands:

```
root:ld.so-1.9.10#   cd util
root:util#   make ldd ldconfig
root:util#   cp ldd /bin
root:util#   cp ldconfig /sbin
root:util#   cd ../man
root:man#   cp ldd.1 /usr/share/man/man1
```

```
root:man#   cp *.8 /usr/share/man/man8
root:man#   rm /usr/bin/ldd
```

Installing Libtool

Install Libtool by running the following commands:

```
root:libtool-1.3.4#   ./configure --prefix=/usr
root:libtool-1.3.4#   make
root:libtool-1.3.4#   make install
```

Installing Linux86

Install Linux86 by running the following commands:

```
root:linux-86#   cd as
root:as#   make
root:as#   make install
root:as#   cd ../ld
root:ld#   make ld86
root:ld#   make install
root:ld#   cd ../man
root:man#   cp as86.1 ld86.1 /usr/share/man/man1
```

Installing Make

Install Make by running the following commands:

```
root:make-3.78.1#   ./configure --prefix=/usr
root:make-3.78.1#   make
root:make-3.78.1#   make install
```

Installing Shell Utils

Install Shellutils by running the following commands:

```
root:sh-utils-2.0#   ./configure --prefix=/usr
root:sh-utils-2.0#   make
root:sh-utils-2.0#   make install
root:sh-utils-2.0#   cd /usr/bin
root:bin#   mv date echo false pwd stty /bin
root:bin#   mv su true uname hostname /bin
```

Installing Shadow Password Suite

Install the Shadow Password Suite by running the following commands:

```
root:shadow-19990827#   ./configure --prefix=/usr
root:shadow-19990827#   make
root:shadow-19990827#   make install
root:shadow-19990827#   cd etc
root:etc#   cp limits login.access login.defs.linux shells
suauth /etc
root:etc#   mv /etc/login.defs.linux /etc/login.defs
```

Installing Man

Install Man by running the following commands:

```
root:man-1.5h1#   ./configure -default
root:man-1.5h1#   make
root:man-1.5h1#   make install
```

Installing Modutils

Install Modutils by running the following commands:

```
root:modutils-2.3.9#  ./configure
root:modutils-2.3.9#  make
root:modutils-2.3.9#  make install
```

Installing Procinfo

Install Procinfo by running the following commands:

```
root:procinfo-17#  make
root:procinfo-17#  make install
```

Installing Procps

Install Procps by running the following commands:

```
root:procps-2.0.6#  gcc -c watch.c
root:procps-2.0.6#  make
root:procps-2.0.6#  make -e XSCPT="" install
root:procinfo-17#  mv /usr/bin/kill /bin
```

Installing Psmisc

Install Psmisc by running the following commands:

```
root:psmisc-19#  make
root:psmisc-19#  make install
```

Installing Sed

Install Sed by running the following commands:

```
root:sed-3.02#   ./configure --prefix=/usr
root:sed-3.02#   make
root:sed-3.02#   make install
root:sed-3.02#   mv /usr/bin/sed /bin
```

Installing Start–stop–daemon

Install Start–stop–daemon by running the following commands:

```
root:ssd-0.4.1#   make
root:ssd-0.4.1#   make install
```

Installing Sysklogd

Install Sysklogd by running the following commands:

```
root:sysklogd-1.3-31#   make
root:sysklogd-1.3-31#   make install
```

Installing Sysvinit

Install Sysvinit by running the following commands:

```
root:sysvinit-2.78#   cd src
root:sysvinit-2.78#   make
root:sysvinit-2.78#   make install
```

Installing Tar

Install Tar by running the following commands:

```
root:tar-1.13#   ./configure --prefix=/usr
root:tar-1.13#   make
root:tar-1.13#   make install
root:tar-1.13#   mv /usr/bin/tar /bin
```

Installing Textutils

Install Textutuils by running the following commands:

```
root:textutils-2.0#   ./configure --prefix=/usr
root:textutils-2.0#   make
root:textutils-2.0#   make install
root:textutils-2.0#   mv /usr/bin/cat /bin
```

Installing Vim

You need to unpack both the vim−rt and vim−src packages to install Vim. Install Vim by running the following commands:

```
root:vim-5.6#   ./configure --prefix=/usr
root:vim-5.6#   make
root:vim-5.6#   make install
root:vim-5.6#   cd /usr/bin
root:bin#   ln -s vim vi
```

Installing Util−Linux

Before we can install the package we have to edit the MCONFIG file, find and modify the following variables as follows:

HAVE_PASSWD=yes
HAVE_SLN=yes
HAVE_TSORT=yes

Install Util−Linux by running the following commands:

```
root:util-linux-2.10h#   groupadd -g 5 tty
root:util-linux-2.10h#   ./configure
root:util-linux-2.10h#   make
root:util-linux-2.10h#   make install
```

Installing Pmac−utils

Install Pmac−utils by running the following commands:

```
root:pmac-utils-1.1.1#    make clock
root:pmac-utils-1.1.1#    cp clock /sbin
root:pmac-utils-1.1.1#    rm /sbin/hwclock
```

Create a new file `/sbin/hwclock` containing the following:

```
#!/bin/sh
# Begin /sbin/hwclock

/sbin/clock -s
```

End /sbin/hwclock

Set the right permissions by running the following command:

```
root:~# chmod 755 /sbin/hwclock
```

Removing old NSS library files

If you have copied the NSS Library files from your normal Linux system to the LFS system (because your normal system runs glibc–2.0) it's time to remove them now by running:

```
root:~#    rm /lib/libnss*.so.1 /lib/libnss*2.0*
```

Configuring essential software

Now that all software is installed, all that we need to do to get a few programs running properly is to create their configuration files.

Configuring Glibc

We need to create the /etc/nsswitch.conf file. Although glibc should provide defaults when this file is missing or corrupt, it's defaults don't work work well with networking which will be dealt with in a later chapter. Also, our timezone needs to be setup.

Create a new file `/etc/nsswitch.conf` containing:

```
# Begin /etc/nsswitch.conf

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

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

netgroup: db files

# End /etc/nsswitch.conf
```

Run the **`tzselect`** script and answer the questions regarding your timezone. When you're done, the script will give you the location of the timezone file you need.

Create the `/etc/localtime` symlink by running:

```
root:~#   cd /etc
root:etc#   rm localtime
root:etc#   ln -s ../usr/share/zoneinfo/<tzselect's output> \
> localtime
```

tzselect's output can be something like *EST5EDT* or *Canada/Eastern*. The symlink you would create with that information would be *ln −s ../usr/share/zoneinfo/EST5EDT localtime* or *ln −s ../usr/share/zoneinfo/Canada/Eastern localtime*

Configuring Dynamic Loader

By default the dynamic loader searches a few default paths for dynamic libraries, so there normally isn't a need for the `/etc/ld.so.conf` file unless you have extra directories in which you want the system to search for paths. The `/usr/local/lib` directory isn't searched through for dynamic libraries by default, so we want to add this path so when you install software you won't be suprised by them not running for some reason.

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

```
# Begin /etc/ld.so.conf

/lib
/usr/lib
/usr/local/lib

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

Although it's not necessary to add the `/lib` and `/usr/lib` directories it doesn't hurt. This way you see right away what's being searched and don't have to remeber the default search paths if you don't want to.

Configuring Sysklogd

Create the `/etc/syslog.conf` file containing the following:

```
# Begin /etc/syslog.conf

auth,authpriv.* −/var/log/auth.log
*.*;auth,authpriv.none −/var/log/sys.log
daemon.* −/var/log/daemon.log
kern.* −/var/log/kern.log
mail.* −/var/log/mail.log
user.* −/var/log/user.log
*.emerg *

# End /etc/syslog.conf
```

Configuring Shadow Password Suite

This package contains the utilities to modify user's passwords, add new users/groups, delete users/groups and more. I'm not going to explain to you what 'password shadowing' means. You can read all about that in the doc/HOWTO file. There's one thing you should keep in mind, if you decide to use shadow support, that programs that need to verify passwords (examples are xdm, ftp daemons, pop3 daemons, etc) need to be 'shadow−compliant', eg. they need to be able to work with shadowed passwords.

If you decide you don't want to use shadowed passwords (after you're read the doc/HOWTO document), you still use this archive since the utilities in this archive are also used on system which have shadowed passwords disabled. You can read all about this in the HOWTO. Also note that you can switch between shadow and non−shadow at any point you want.

Now is a very good moment to read chapter 5 of the doc/HOWTO file. You can read how you can test if shadowing works and if not, how to disable it. If it doesn't work and you haven't tested it, you'll end up with an unusable system after you logout of all your consoles, since you won't be able to login anymore. You can easily fix this by passing the init=/sbin/sulogin parameter to the kernel, unpack the util−linux archive, go to the login−utils directory, build the login program and replace the /bin/login by the one in the util−linux package. Things are never hopelessly messed up (at least not under Linux), but you can avoid a hassle by testing properly and reading manuals ;)

Configuring Sysvinit

Create a new file `/etc/inittab` containing the following:

```
# Begin /etc/inittab

id:2:initdefault:

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

su:S:wait:/sbin/sulogin

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

ft:6:respawn:/sbin/sulogin
```

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

1:2345:respawn:/sbin/agetty /dev/tty1 9600
2:2345:respawn:/sbin/agetty /dev/tty2 9600
3:2345:respawn:/sbin/agetty /dev/tty3 9600
4:2345:respawn:/sbin/agetty /dev/tty4 9600
5:2345:respawn:/sbin/agetty /dev/tty5 9600
6:2345:respawn:/sbin/agetty /dev/tty6 9600

# End /etc/inittab
```

Creating the /var/run/utmp file

Programs like login, shutdown, uptime and others want to read from and write to the /var/run/utmp file. This file contains information about who is currently logged in. It also contains information on when the computer was last booted and shutdown.

Create the `/var/run/utmp` and give it the proper permissions by running the following commands:

```
root:~# touch /var/run/utmp
root:~# chmod 644 /var/run/utmp
```

Configuring Vim

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

```
set nocompatible
set bs=2
```

Chapter 12. Creating system boot scripts

What is being done here

This chapter will create the necessary scripts that are run at boottime. These scripts perform tasks such as remounting the root file system mounted read−only by the kernel into read−write mode, activiating the swap partition(s), running a check on the root file system to make sure it's intact and starting the daemons that the system uses.

Create the directories and master files

We need to start by creating a few extra directories that are used by the boot scripts. Create these directories by running:

```
root:~#   cd /etc
root:etc#   mkdir rc0.d rc1.d rc2.d rc3.d
root:etc#   mkdir rc4.d rc5.d rc6.d init.d rcS.d
```

The first main bootscript is the `/etc/init.d/rc` script. Create a new file `/etc/init.d/rc` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/rc
#
# By Jason Pearce  − jason.pearce@linux.org
#

# Un−comment the following for debugging.
# debug=echo

#
# Start script or program.
#
startup() {
case "$1" in
*.sh)
$debug sh "$@"
;;
*)
$debug "$@"
;;
esac
}

# Ignore CTRL−C only in this shell, so we can interrupt subprocesses.
trap ":" INT QUIT TSTP

# Set onlcr to avoid staircase effect.
stty onlcr 0>&1

# Now find out what the current and what the previous runlevel are.
runlevel=$RUNLEVEL
```

```
# Get first argument. Set new runlevel to this argument.

[ "$1" != "" ] && runlevel=$1
if [ "$runlevel" = "" ]
then
echo "Usage: $0 <runlevel>" >&2
exit 1
fi

previous=$PREVLEVEL
[ "$previous" = "" ] && previous=N

export runlevel previous

# Is there an rc directory for this new runlevel?

if [ −d /etc/rc$runlevel.d ]
then
# First, run the KILL scripts for this runlevel.
if [ $previous != N ]
then
for i in /etc/rc$runlevel.d/K*
do
[ ! −f $i ] && continue

suffix=${i#/etc/rc$runlevel.d/K[0−9][0−9]}
previous_start=/etc/rc$previous.d/S[0−9][0−9]$suffix

# Stop the service if there is a start script
# in the previous run level.
[ ! −f $previous_start ] && continue

startup $i stop
done
fi

# Now run the START scripts for this runlevel.
for i in /etc/rc$runlevel.d/S*
do
[ ! −f $i ] && continue

if [ $previous != N ]
then
# Find start script in previous runlevel and
# stop script in this runlevel.
suffix=${i#/etc/rc$runlevel.d/S[0−9][0−9]}
stop=/etc/rc$runlevel.d/K[0−9][0−9]$suffix
previous_start=/etc/rc$previous.d/S[0−9][0−9]$suffix

# If there is a start script in the previous
# level
```

```
# and _no_ stop script in this level, we don't
# have to re−start the service.
[ −f $previous_start ] && [ ! −f $stop ] && continue
fi

case "$runlevel" in
0|6)
startup $i stop
;;
*)
startup $i start
;;
esac
done
fi

# End /etc/init.d/rc
```

The second main bootscript is the `rcS` script. Create a new file `/etc/init.d/rcS` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/rcS

runlevel=S
prevlevel=N
umask 022
export runlevel prevlevel

trap ":" INT QUIT TSTP

for i in /etc/rcS.d/S??*
do
[ ! −f "$i" ] && continue;
$i start
done

# End /etc/init.d/rcS
```

Creating the reboot script

Create a new file /etc/init.d/reboot containing the following:

```
#!/bin/sh
# Begin /etc/init.d/reboot

echo "System reboot in progress..."

/sbin/reboot –d –f –i

# End /etc/init.d/reboot
```

Creating the halt script

Create a new file `/etc/init.d/halt` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/halt

/sbin/halt -d -f -i -p

# End /etc/init.d/halt
```

Creating the mountfs script

Create a new file `/etc/init.d/mountfs` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/mountfs

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

echo -n "Remounting root file system in read-write mode..."
/bin/mount -n -o remount,rw /
check_status

echo > /etc/mtab
/bin/mount -f -o remount,rw /

echo -n "Mounting proc file system..."
/bin/mount proc
check_status

# End /etc/init.d/mountfs
```

Creating the umountfs script

Create a new file `/etc/init.d/umountfs` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/umountfs

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

echo -n "Deactivating swap..."
/sbin/swapoff -a
check_status

echo -n "Unmounting file systems..."
/bin/umount -a -r
check_status

# End /etc/init.d/umountfs
```

Creating the sendsignals script

Create a new file `/etc/init.d/sendsignals` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/sendsignals

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

echo −n "Sending all processes the TERM signal..."
/sbin/killall5 −15
check_status

echo −n "Sending all processes the KILL signal..."
/sbin/killall5 −9
check_status

# End /etc/init.d/sendsignals
```

Creating the checkroot script

Create a new file `/etc/init.d/checkroot` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/checkroot

echo −n "Activating swap..."
/sbin/swapon −a

if [ −f /fastboot ]
then
  echo "Fast boot, no file system check"

  /bin/mount −n −o remount,ro /
  if [ $? = 0 ]
  then

    if [ −f /forcecheck ]
    then
      force="−f"
    else
      force=""
    fi

    echo "Checking root file system..."
    /sbin/fsck $force −a /

    if [ $? −gt 1 ]
    then
      echo
      echo "fsck failed. Please repair your file system manually by"
      echo "running /sbin/fsck without the −a option"
      echo
      echo "Please note that the file system is currently mounted in"
              echo "read−only mode."
      echo
        echo "I will start sulogin now. CTRL+D will reboot your system."
      echo
              /sbin/sulogin
              /sbin/reboot −f
    fi
  else
    echo "Cannot check root file system because it is not mounted in"
    echo "read−only mode."
  fi
fi
```

```
# End /etc/init.d/checkroot
```

Creating the setclock script

Create a new file `/etc/init.d/setclock` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/setclock

check_status()
{
  if [ $? = 0 ]
  then echo ""
  else
    echo "FAILED"
  fi
}

echo -n "Setting clock..."
/sbin/hwclock
check_status

# End /etc/init.d/setclock
```

Creating the sysklogd script

Create a new file /etc/init.d/sysklogd containing the following:

```
#!/bin/sh
# Begin /etc/init.d/sysklogd

check_status()
{
if [ $? = 0 ]
then
  echo "OK"
else
  echo "FAILED"
fi
}

case "$1" in
  start)
    echo −n "Starting system log daemon..."
    start−stop−daemon −S −q −o −x /usr/sbin/syslogd −− −m 0
    check_status

    echo −n "Starting kernel log daemon..."
    start−stop−daemon −S −q −o −x /usr/sbin/klogd
    check_status
    ;;

  stop)
    echo −n "Stopping kernel log daemon..."
    start−stop−daemon −K −q −o −p  /var/run/klogd.pid
    check_status

    echo −n "Stopping system log daemon..."
    start−stop−daemon −K −q −o −p /var/run/syslogd.pid
    check_status
    ;;

  reload)
    echo −n "Reloading system load daemon configuration file..."
    start−stop−daemon −K −q −o −s 1 −p /var/run/syslogd.pid
    check_status
    ;;

  restart)
    echo −n "Stopping kernel log daemon..."
    start−stop−daemon −K −q −o −p /var/run/klogd.pid
```

```
    check_status

    echo -n "Stopping system log daemon..."
    start-stop-daemon -K -q -o -p /var/run/syslogd.pid
    check_status

    sleep 1

    echo -n "Starting system log daemon..."
    start-stop-daemon -S -q -o -x /usr/sbin/syslogd -- -m 0
    check_status

    echo -n "Starting kernel log daemon..."
    start-stop-daemon -S -q -o -x /usr/sbin/klogd
    check_status
    ;;

  *)
    echo "Usage: $0 {start|stop|reload|restart}"
    exit 1
    ;;
esac

# End /etc/init.d/sysklogd
```

Setting up symlinks and permissions

Give these files the proper permissions and create the necessary symlinks by running the following commands:

```
root:~#   cd /etc/init.d
root:init.d#   chmod 755 rcS reboot halt mountfs umountfs
root:init.d#   chmod 755 sendsignals checkroot sysklogd
root:init.d#   cd ../rc0.d
root:rc0.d#   ln -s ../init.d/sysklogd K90sysklogd
root:rc0.d#   ln -s ../init.d/sendsignals S80sendsignals
root:rc0.d#   ln -s ../init.d/umountfs S90umountfs
root:rc0.d#   ln -s ../init.d/halt S99halt
root:rc0.d#   cd ../rc6.d
root:rc6.d#   ln -s ../init.d/sysklogd K90sysklogd
root:rc6.d#   ln -s ../init.d/sendsignals S80sendsignals
root:rc6.d#   ln -s ../init.d/umountfs S90umountfs
root:rc6.d#   ln -s ../init.d/reboot S99reboot
root:rc6.d#   cd ../rcS.d
root:rcS.d#   ln -s ../init.d/setclock S01setclock
root:rcS.d#   ln -s ../init.d/checkroot S05checkroot
root:rcS.d#   ln -s ../init.d/mountfs S10mountfs
root:rcS.d#   cd /etc/rc2.d
root:rc2.d#   ln -s ../init.d/sysklogd S03sysklogd
```

Creating the /etc/fstab file

In order for certain programs to be able to determine where certain partitions are supposed to be mounted by default, the /etc/fstab file is used. Create a new file `/etc/fstab` containing the following:

```
# Begin /etc/fstab

/dev/<LFS−partition designation> / ext2 defaults 0 1
/dev/<swap−partition designation> none swap sw 0 0
proc /proc proc defaults 0 0

# End /etc/fstab
```

Replace <LFS−partition designation> and <swap−partition designation> with the appropriate devices (/dev/hda5 and /dev/hda6 in my case).

Chapter 13. Setting up basic networking

Introduction

This chapter will setup basic networking. Although you might not be connected to a network, Linux software uses network functions anyway. We'll be installing at least the local loopback device and a network card as well if applicable. Also the proper bootscripts will be created so that networking will be enabled during boot time.

Installing network software

Installing Netkit−base

Install Netkit−base by running the following commands:

```
root:netkit-base-0.17...#   ./configure --prefix=/usr
root:netkit-base-0.17...#   make
root:netkit-base-0.17...#   make install
root:netkit-base-0.17...#   cd etc.sample
root:netkit-base-0.17.../etc.sample#   cp services protocols
/etc
```

Installing Net−tools

Install Net−tools by running the following commands:

```
root:net-tools-1.54#   make
root:net-tools-1.54#   make install
```

Creating network boot scripts

Creating the /etc/init.d/localnet bootscript

Create a new file `/etc/init.d/localnet` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/localnet

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}
echo −n "Setting up loopback device..."
/sbin/ifconfig lo 127.0.0.1
check_status

echo −n "Setting up hostname..."
/bin/hostname −−file /etc/hostname
check_status

# End /etc/init.d/localnet
```

Setting up permissions and symlink

Set the proper file permissions and create the necessary symlink by running the following commands:

```
root:~#    cd /etc/init.d
root:init.d#    chmod 755 /etc/init.d/localnet
root:init.d#    cd ../rcS.d
root:rcS.d#    ln −s ../init.d/localnet S03localnet
```

Creating the /etc/hostname file

Create a new file `/etc/hostname` and put the hostname in it. This is not the FQDN (Fully Qualified Domain Name). This is the name you wish to call your computer in a network. An example:

```
lfs
```

The file must not contain empty lines or spaces after the hostname. Don't press enter either when you entered the name.

Creating the /etc/hosts file

If you want to configure a network card, you have to decide on the IP−address, FQDN and possible aliases for use in the /etc/hosts file. An example is:

```
<my−IP> myhost.mydomain.org aliases
```

Make sure the IP−address is in the private network IP−address range. Valid ranges are:

```
Class Networks
A     10.0.0.0
B     172.16.0.0 through 172.31.0.0
C     192.168.0.0 through 192.168.255.0
```

A valid IP address could be 192.168.1.1. A valid FQDN for this IP could be www.linuxfromscratch.org

If you're not going to use a network card, you still need to come up with a FQDN. This is necessary for programs like Sendmail to operate correctly (in fact; Sendmail won't run when it can't determine the FQDN).

If you don't configure a network card, create a new file `/etc/hosts` containing:

```
# Begin /etc/hosts (no network card version)
```

```
127.0.0.1 www.linuxfromscratch.org <contents of /etc/hostname> localhost

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

If you do configure a network card, create a new file `/etc/hosts` containing:

```
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
192.168.1.1 www.linuxfromscratch.org <contents of /etc/hostname>

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

Of course, change the 192.168.1.1 and www.linuxfromscratch.org to your own liking (or requirements if you are assigned an IP−address by a network/system administrator and you plan on connecting this machine to that network).

Creating the /etc/init.d/ethnet file

This section only applies if you are going to configure a network card. If you're not, skip this section.

Create a new file `/etc/init.d/ethnet` containing the following:

```
#!/bin/sh
# Begin /etc/init.d/ethnet

check_status()
{
  if [ $? = 0 ]
  then
    echo "OK"
  else
    echo "FAILED"
  fi
}

IPADDR="209.83.245.12" # Replace with your own IP address
NETMAKSK="255.255.255.0" # Replace with your own Netmask
BROADCAST="209.83.245.255" # Replace with your own Broadcast addr.
GATEWAY="209.83.245.1" # Replace with your own Gateway address
```

```
echo -n "Setting up eth0..."
/sbin/ifconfig eth0 $IPADDR broadcast $BROADCAST netmask $NETMASK
check_status

echo "Adding default gateway..."
/sbin/route add default gw $GATEWAY metric 1
check_status

# End /etc/init.d/ethnet
```

Setting up permissions and symlink

Set the proper file permissions and create the necessary symlink by running the following commands:

```
root:~#    cd /etc/init.d
root:init.d#    chmod 755 /etc/init.d/ethnet
root:init.d#    cd ../rc2.d
root:rc2.d#    ln -s ../init.d/ethnet S10ethnet
```

Chapter 14. Making the LFS system bootable

Introduction

This chapter will make LFS bootable. This chapter deals with building a new kernel for our new LFS system and moving the kernel to the MacOS side so we can boot the LFS system.

Installing a kernel

A kernel is the heart of a Linux system. We could use the kernel image from our normal system, but we might as well compile a new kernel from the most recent kernel sources available.

Building the kernel involves a few steps: configuring it and compiling it. There are a few ways to configure the kernel. If you don't like the way this book does it, read the `README` file and find out what your other options are. Run the following commands to build the kernel:

If you need to apply the Kernel USB patch, do that by running the following commands:

```
root:~#   cd /usr/src/linux
root:linux#   patch -p1 -i ../usb-2.3.50-1-for-2.2.14.diff.gz
```

The build the actual kernel, run the following commands:

```
root:linux#   make mrproper
root:linux#   make pmac_config
root:linux#   make menuconfig
root:linux#   make dep
root:linux#   make vmlinux
root:linux#   cp System.map /boot
root:linux#   cp vmlinux /boot/lfskernel
```

Updating BootX

Now we have to get /boot/lfskernel to the Mac OS side so we can boot our LFS system. There are a few ways to copy the /boot/lfskernel file to the Linux kernel folder on the Mac OS side.

The easiest way is be to mount a Mac HFS partition under Linux and copy the kernel to that partition in the right folder. The Linux kernel currently does not support the HFS+ partition, do do not attempt to mount a Mac HFS+ (also known as HFS Extended) partition under Linux.

Copy the kernel to your Mac HFS partition by running the following commands:

```
root:~#   mkdir /mnt/exchange
root:~#   mount -t hfs /dev/sda1 /mnt/exchange
root:~#   cp /boot/lfskernel /mnt/exchange
root:~#   umount /dev/sda1
```

Of course, replace /dev/sda1 by your Mac partition's designation.

If you can't mount the Mac partition for some reason (for example because it's a HFS+ partition) you'll have to email the kernel to yourself. Use a shell on your normal Linux's system (not the the chroot'ed environment) to obtain the kernel image. Compress it with gzip and attach it to an email. Boot into your MacOS and download the email. You can use the MacGzip application to ungzip the kernel image and move it to the "Linux Kernels" folder under "System Folder". If you don't have MacGzip installed, you can download it from http://macinsearch.com/infomac/cmp/mac−gzip−111.html

Of course, if the kernel is small enough to fit on a floppy disk, and your Mac has a floppy drive, you can transfer it that way. Of if you have a ZIP drive at your disposal, you can transfer it on that medium.

Testing the system

Now that all software has been installed, bootscripts have been written and the local network is setup, it's time for you to reboot your computer and test these new scripts to verify that they actually work. You first want to execute them manually from the /etc/init.d directory so you can fix the most obvious problems (typos, wrong paths and such). When those scripts seem to work just fine manually they should also work during a system start or shutdown. There's only one way to test that. Shutdown your system with shutdown –r now and reboot into LFS. After the reboot you will have a normal login prompt like you have on your normal Linux system (unless you use XDM or some sort of other Display Manger (like KDM – KDE's version of XDM).

When you are at the login prompt, login as user root and when asked for a password just press enter. The first thing you want to do is set a password for user root by running the following command:

```
:root:~#   passwd
```

At this point your basic LFS system is ready for use. Everything else that follows now is optional, so you can skip packages at your own discretion. But do keepein mind that if you skip packages (especially libraries) you can break dependencies of other packages. For example, when the Lynx browser is installed, the zlib library is installed as well. You can decide to skip the zlib library, but this library isn't used by Lynx alone. Other packages require this library too. The same may apply to other libraries and programs.

IV. Part IV – Appendixes

Table of Contents

Appendix A. Package descriptions

Introduction

This appendix describes the following aspect of each and every package that is installed in this book:

- What every package contains
- What every program from a package does

The packages are listed in the same order as they are installed in chapter 5 (Intel system) or chapter 11 (PPC systems).

Most information about these packages (especially the descriptions of it) come from the man pages from those packages. I'm not going to print the entire man page, just the core elements to make you understand what a program does. If you want to know full details on a program, I suggest you start by reading the complete man page in addition to this appendix.

You will also find that certain packages are documented more in depth than others. The reason is that I just happen to know more about certain packages than I know about others. If you have anything to add on the following descriptions, please don't hesitate to email me. This list is going to contain an in depth description of every package installed, but I can't do this on my own. I have had help from various people but more help is needed.

Please note that currently only what a package does is described and not why you need to install it. That will be added later.

Glibc

Contents

The Glibc package contains the GNU C Library.

Description

The C Library is a collection of commonly used functions in programs. This way a programmer doens't need to create his own functions for every single task. The most common things like writing a string to your screen are already present and at the disposal of the programmer.

The C library (actually almost every library) come in two flavours: dynamic ones and static ones. In short when a program uses a static C library, the code from the C library will be copied into the executable file. When a program uses a dynamic library, that executable will not contain the code from the C library, but instead a routine that loads the functions from the library at the time the program is run. This means a significant decrease in the file size of a program. If you don't understand this concept, you better read the documentation that comes with the C Library as it is too complicated to explain here in one or two lines.

Ed

Contents

The Ed package contains the ed program.

Description

Ed is a line−oriented text editor. It is used to create, display, modify and otherwise manipulate text files.

Patch

Contents

The Patch package contains the patch program.

Description

The patch program modifies a file according to a patch file. A patch file usually is a list created by the diff program that contains instructions on how an original file needs to be modified. Patch is used a lot for source code patches since it saves time and space. Imagine you have a package that is 1MB in size. The next version of that package only has changes in two files of the first version. You can ship an entirely new package of 1MB or provide a patch file of 1KB which will update the first version to make it identical to the second version. So if you have downloaded the first version already, a patch file can save you a second large download.

GCC

Contents

The GCC package contains compilers, preprocessors and the GNU C++ Library.

Description

Compiler

A compiler translates source code in text format to a format that a computer understands. After a source code file is compiled into an object file, a linker will create an executable file from one or more of these compiler generated object files.

Pre−processor

A pre−processor pre−processes a source file, such as including the contents of header files into the source file. You generally don't do this yourself to save yourself a lot of time. You just insert a line like #include <filename>. The pre−processor file insert the contents of that file into the source file. That's one of the things a pre−processor does.

C++ Library

The C++ library is used by C++ programs. The C++ library contains functions that are frequently used in C++ programs. This way the programmer doens't have to write certain functions (such as writing a string of text to the screen) from scratch every time he creates a program.

Bison

Contents

The Bison package contains the bison program.

Description

Bison is a parser generator, a replacement for YACC. YACC stands for Yet Another Compiler Compiler. What is Bison then? It is a program that generates a program that analyses the structure of a textfile. Instead of writing the actual program you specify how things should be connected and with those rules a program is constructed that analyses the textfile.

There are alot of examples where structure is needed and one of them is the calculator.

Given the string :

```
1 + 2 * 3
```

You can easily come to the result 7. Why ? Because of the structure. You know how to interpretet the string. The computer doesn't know that and Bison is a tool to help it understand by presenting the string in the following way to the compiler:

```
  +
 / \
1   *
   / \
  2   3
```

You start at the bottom of a tree and you come across the numbers 2 and 3 which are joined by the multiplication symbol, so the computers multiplies 2 and 3. The result of that multiplication is remembered and the next thing that the computer sees is the result of 2*3 and the number 1 which are joined by the add symbol. Adding 1 to the previous result makes 7. In calculating the most complex calculations can be broken down in this tree format and the computer just starts at the bottom and works it's way up to the top and comes with the correct answer. Of course, Bison isn't only used for calculators alone.

Mawk

Contents

The Mawk package contains the mawk program.

Description

Mawk is an interpreter for the AWK Programming Language. The AWK language is useful for manipulation of data files, text retrieval and processing, and for prototyping and experimenting with algorithms.

Findutils

Contents

The Findutils package contains the find, locate, updatedb and xargs programs.

Description

Find

The find program searches for files in a directory hierarchy which match a certain criteria. If no criteria is given, it lists all files in the current directory and it's subdirectories.

Locate

Locate scans a database which contain all files and directories on a filesystem. This program lists the files and directories in this database matching a certain criteria. If you're looking for a file this program will scan the database and tell you exactly where the files you requested are located. This only makes sense if your locate database is fairly up–to–date else it will provide you with out–of–date information.

Updatedb

The updatedb program updates the locate database. It scans the entire file system (including other file system that are currently mounted unless you specify it not to) and puts every directory and file it finds into the database that's used by the locate program which retrieves this information. It's a good practice to update this database once a day so that you are ensured of a database that is up–to–date.

Xargs

The xargs command applies a command to a list of files. If you need to perform the same command on multiple files, you can create a file that contains all these files (one per line) and use xargs to perform that command on the list.

Termcap

Contents

The Termcap package contains the termcap library.

Description

The termcap library contains C functions that enable programs to send control strings to terminals in a way independent of the terminal type. The GNU termcap library does not place an arbitrary limit on the size of termcap entries, unlike most other termcap libraries.

The use of termcap is discouraged. Termcap is being phased out in favor of the terminfo−based ncurses library, which contains an emulation of the termcap library routines in addition to an excellent curses implementation. The reason for having Termcap installed is there are one or two programs that specifically need this library and don't know about the NCurses library (yet).

Ncurses

Contents

The Ncurses package contains the ncurses, panel, menu and form libraries. It also contains the tic, infocmp, clear, tput, toe and tset programs.

Description

The libraries

The libraries that make up the Ncurses library are used to display text (often in a fancy way) on your screen. An example where ncurses is used is in the kernel's "make menuconfig" process. The libraries contain routines to create panels, menu's, form and general text display routines.

Tic

Tic is the terminfo entry−description compiler. The program translates a terminfo file from source format into the binary format for use with the ncurses library routines. Terminfo files contain information about the capabilities of your terminal.

Infocmp

The infocmp program can be used to compare a binary terminfo entry with other terminfo entries, rewrite a terminfo description to take advantage of the use= terminfo field, or print out a terminfo description from the binary file (term) in a variety of formats (the opposite of what tic does).

clear

The clear program clears your screen if this is possible. It looks in the environment for the terminal type and then in the terminfo database to figure out how to clear the screen.

tput

The tput program uses the terminfo database to make the values of terminal−dependent capabilities and information available to the shell, to initialize or reset the terminal, or return the long name of the requested terminal type.

toe

The toe program lists all available terminal types by primary name with descriptions.

---

tset

The Tset program initializes terminals so they can be used, but it's not widely used anymore. It's provided for 4.4BSD compatibility.

Less

Contents

The Less package contains the less program

Description

The less program is a file pager (or text viewer). It displays the contents of a file with the ability to scroll. Less is an improvement on the common pager called "more". Less has the ability to scroll backwards through files as well and it doesn't need to read the entire file when it starts, which makes it faster when you are reading large files.

Perl

Contents

The Perl package contains Perl – Practical Extraction and Report Language

Description

Perl combines the features and capabilities of C, awk, sed and sh into one powerful programming language.

M4

Contents

The M4 package contains the M4 processor

Description

M4 is a macro processor. It copies input to output expanding macros as it goes. Macros are either builtin or user−defined and can take any number of arguments. Besides just doing macro expansion m4 has builtin functions for including named files, running UNIX commands, doing integer arithmetic, manipulating text in various ways, recursion, etc. M4 can be used either as a front−end to a compiler or as a macro processor in its own right.

Texinfo

Contents

The Texinfo package contains the info, install–info, makeinfo, texi2dvi and texindex programs

---

Description

info

The info program reads Info documents, usually contained in your /usr/doc/info directory. Info documents are like man(ual) pages, but they tend to be more in depth than just explaining the options to a program.

---

install–info

The install–info program updates the info entries. When you run the info program a list with available topics (ie: available info documents) will be presented. The install–info program is used to maintain this list of available topics. If you decice to remove info files manually, you need to delete the topic in the index file as well. This program is used for that. It also works the other way around when you add info documents.

---

makeinfo

The makeinfo program translates Texinfo source documents into various formats. Available formats are: info files, plain text and HTML.

---

texi2dvi

The texi2dvi program prints Texinfo documents

---

texindex

The texindex program is used to sort Texinfo index files.

Autoconf

Contents

The Autoconf package contains the autoconf, autoheader, autoreconf, autoscan, autoupdate and ifnames programs

---

Description

autoconf

Autoconf is a tool for producing shell scripts that automatically configure software source code packages to adapt to many kinds of UNIX−like systems. The configuration scripts produced by Autoconf are independent of Autoconf when they are run, so their users do not need to have Autoconf.

---

autoheader

The autoheader program can create a template file of C #define statements for configure to use

---

autoreconf

If you have a lot of Autoconf−generated configure scripts, the autoreconf program can save you some work. It runs autoconf (and autoheader, where appropriate) repeatedly to remake the Autoconf configure scripts and configuration header templates in the directory tree rooted at the current directory.

---

autoscan

The autoscan program can help you create a configure.in file for a software package. autoscan examines source files in the directory tree rooted at a directory given as a command line argument, or the current directory if none is given. It searches the source files for common portability problems and creates a file configure.scan which is a preliminary configure.in for that package.

---

autoupdate

The autoupdate program updates a configure.in file that calls Autoconf macros by their old names to use the current macro names.

---

ifnames

ifnames can help when writing a configure.in for a software package. It prints the identifiers that the package already uses in C preprocessor conditionals. If a package has already been set up to have some portability, this program can help you figure out what its configure needs to check for. It may help fill in some gaps in a configure.in generated by autoscan.

Automake

Contents

The Autoconf package contains the aclocal and automake programs

Description

aclocal

Automake includes a number of Autoconf macros which can be used in your package; some of them are actually required by Automake in certain situations. These macros must be defined in your aclocal.m4; otherwise they will not be seen by autoconf.

The aclocal program will automatically generate aclocal.m4 files based on the contents of configure.in. This provides a convenient way to get Automake–provided macros, without having to search around. Also, the aclocal mechanism is extensible for use by other packages.

automake

To create all the Makefile.in's for a package, run the automake program in the top level directory, with no arguments. automake will automatically find each appropriate Makefile.am (by scanning configure.in) and generate the corresponding Makefile.in.

Bash

Contents

The Bash package contains the bash program

Description

Bash is the Bourne−Again SHell, which is a widely used command interpreter on Unix systems. Bash is a program that reads from standard input, the keyboard. You type something and the program will evaluate what you have typed and do something with it, like running a program.

Flex

Contents

The Flex package contains the flex program

Description

Flex is a tool for generating programs which regognize patterns in text. Pattern recognition is very useful in many applications. You set up rules what to look for and flex will make a program that looks for those patterns. The reason people use flex is that it is much easier to set up rules for what to look for than to write the actual program that finds the text.

Binutils

Description

The Binutils package contains the ld, as, ar, nm, objcopy, objdump, ranlib, size, strings, strip, c++filt, addr2line and nlmconv programs

---

Description

ld

ld combines a number of object and archive files, relocates their data and ties up symbol references. Often the last step in building a new compiled program to run is a call to ld.

---

as

as is primarily intended to assemble the output of the GNU C compiler gcc for use by the linker ld.

---

ar

The ar program creates, modifies, and extracts from archives. An archive is a single file holding a collection of other files in a structure that makes it possible to retrieve the original individual files (called members of the archive).

---

nm

nm lists the symbols from object files.

---

objcopy

objcopy utility copies the contents of an object file to another. objcopy uses the GNU BFD Library to read and write the object files. It can write the destination object file in a format different from that of the source object file.

---

objdump

objdump displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work.

---

ranlib

ranlib generates an index to the contents of an archive, and stores it in the archive. The index lists each symbol defined by a member of an archive that is a relocatable object file.

---

size

size lists the section sizes −−and the total size−− for each of the object files objfile in its argument list. By default, one line of output is generated for each object file or each module in an archive.

---

strings

For each file given, strings prints the printable character sequences that are at least 4 characters long (or the number specified with an option to the program) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file.

strings is mainly useful for determining the contents of non−text files.

---

strip

strip discards all or specific symbols from object files. The list of object files may include archives. At least one object file must be given. strip modifies the files named in its argument, rather than writing modified copies under different names.

---

c++filt

The C++ language provides function overloading, which means that you can write many functions with the same name (providing each takes parameters of different types). All C++ function names are encoded into a low−level assembly label (this process is known as mangling). The c++filt program does the inverse mapping: it decodes (demangles) low−level names into user−level names so that the linker can keep these overloaded functions from clashing.

---

addr2line

addr2line translates program addresses into file names and line numbers. Given an address and an executable, it uses the debugging information in the executable to figure out which file name and line number are associated with a given address.

---

nlmconv

nlmconv converts relocatable object files into the NetWare Loadable Module files, optionally reading header files for NLM header information.

Bzip2

Contents

The Bzip2 packages contains the bzip2, bunzip2, bzcat and bzip2recover programs.

Description

Bzip2

bzip2 compresses files using the Burrows−Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78−based compressors, and approaches the performance of the PPM family of statistical compressors.

Bunzip2

Bunzip2 decompresses files that are compressed with bzip2.

bzcat

bzcat (or bzip2 −dc) decompresses all specified files to the standard output.

bzip2recover

bzip2recover recovers data from damaged bzip2 files.

Diffutils

Contents

The Diffutils packagec contains the cmp, diff, diff3 and sdiff programs.

Description

cmp and diff

cmp and diff both compare two files and report their differences. Both programs have extra options which compare files in different situations.

diff3

The difference between diff and diff3 is that diff comprares 2 files, diff3 compares 3 files.

sdiff

sdiff merges two two files and interactively outputs the results.

Linux kernel

Contents

The Linux kernel package contains the Linux kernel.

Description

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

E2fsprogs

Contents

The e2fsprogs package contains the chattr, lsattr, uuidgen, badblocks, debugfs, dumpe2fs, e2fsck, e2label, fsck, fsck.ext2, mke2fs, mkfs.ext2, mklost+found and tune2fs programs.

Description

chattr

chattr changes the file attributes on a Linux second extended file system.

---

lsattr

lsattr lists the file attributes on a second extended file system.

---

uuidgen

The uuidgen program creates a new universally unique identifier (UUID) using the libuuid library. The new UUID can reasonably be considered unique among all UUIDs created on the local system, and among UUIDs created on other systems in the past and in the future.

---

badblocks

badblocks is used to search for bad blocks on a device (usually a disk partition).

---

debugfs

The debugfs program is a file system debugger. It can be used to examine and change the state of an ext2 file system.

---

dumpe2fs

dumpe2fs prints the super block and blocks group information for the filesystem present on a specified device.

---

e2fsck and fsck.ext2

e2fsck is used to check a Linux second extended file system. fsck.ext2 does the same as e2fsck.

---

e2label

e2label will display or change the filesystem label on the ext2 filesystem located on the specified device.

---

fsck

fsck is used to check and optionally repair a Linux file system.

---

mke2fs and mkfs.ext2

mke2fs is used to create a Linux second extended file system on a device (usually a disk partition). mkfs.ext2 does the same as mke2fs.

---

mklost+found

mklost+found is used to create a lost+found directory in the current working directory on a Linux second extended file system. mklost+found pre−allocates disk blocks to the directory to make it usable by e2fsck.

---

tune2fs

tune2fs adjusts tunable filesystem parameters on a Linux second extended filesystem.

File

Contents

The File package contains the file program.

Description

File tests each specified file in an attempt to classify it. There are three sets of tests, performed in this order: filesystem tests, magic number tests, and language tests. The first test that succeeds causes the file type to be printed.

Fileutils

Contents

The Fileutils package contains the chgrp, chmod, chown, cp, dd, df, dir, dircolors, du, install, ln, ls, mkdir, mkfifo, mknod, mv, rm, rmdir, sync, touch and vdir programs.

Description

chgrp

chgrp changes the group ownership of each given file to the named group, which can be either a group name or a numeric group ID.

---

chmod

chmod changes the permissions of each given file according to mode, which can be either a symbolic representation of changes to make, or an octal number representing the bit pattern for the new permissions.

---

chown

chown changes the user and/or group ownership of each given file.

---

cp

cp copies files from one place to another.

---

dd

dd copies a file (from the standard input to the standard output, by default) with a user−selectable blocksize, while optionally performing conversions on it.

---

df

df displays the amount of disk space available on the filesystem containing each file name argument. If no file name is given, the space available on all currently mounted filesystems is shown.

---

ls, dir and vdir

dir and vdir are versions of ls with different default output formats. These programs list each given file or directory name. Directory contents are sorted alphabetically. For ls, files are by default listed in columns, sorted vertically, if the standard output is a terminal; otherwise they are listed one per line. For dir, files are by default listed in columns, sorted vertically. For vdir, files are by default listed in long format.

---

dircolors

dircolors outputs commands to set the LS_COLOR environment variable. The LS_COLOR variable is use to change the default color scheme used by ls and related utilities.

---

du

du displays the amount of disk space used by each argument and for each subdirectory of directory arguments.

---

install

install copies files and sets their permission modes and, if possible, their owner and group.

---

ln

ln makes hard or soft (symbolic) links between files.

---

mkdir

mkdir creates directories with a given name.

---

mkfifo

mkfifo creates a FIFO with each given name.

---

mknod

mknod creates a FIFO, character special file, or block special file with the given file name.

---

mv

mv moves files from one directory to another or renames files, depending on the arguments given to mv.

rm

rm removes files or directories.

rmdir

rmdir removes directories, if they are empty.

sync

sync forces changed blocks to disk and updates the super block.

touch

touch changes the access and modification times of each given file to the current time. Files that do not exist are created empty.

Grep

Contents

The grep package contains the egrep, fgrep and grep programs.

Description

egrep

egrep prints lines from files matching an extended regular expression pattern.

---

fgrep

fgrep prints lines from files matching a list of fixed strings, separated by newlines, any of which is to be matched.

---

grep

grep prints lines from files matching a basic regular expression pattern.

Groff

Contents

The Groff packages contains the addftinfo, afmtodit, eqn, grodvi, groff, grog, grohtml, grolj4, grops, grotty, hpftodit, indxbib, lkbib, lookbib, neqn, nroff, pfbtops, pic, psbb, refer, soelim, tbl, tfmtodit and troff programs.

Description

addftinfo

addftinfo reads a troff font file and adds some additional font−metric information that is used by the groff system.

---

afmtodit

afmtodit creates a font file for use with groff and grops.

---

eqn

eqn compiles descriptions of equations embedded within troff input files into commands that are understood by troff.

---

grodvi

grodvi is a driver for groff that produces TeX dvi format.

---

groff

groff is a front−end to the groff document formatting system. Normally it runs the troff program and a postprocessor appropriate for the selected device.

---

grog

grog reads files and guesses which of the groff options −e, −man, −me, −mm, −ms, −p, −s, and −t are required for printing files, and prints the groff command including those options on the standard output.

---

grohtml

grohtml translates the output of GNU troff to html

grolj4

grolj4 is a driver for groff that produces output in PCL5 format suitable for an HP Laserjet 4 printer.

grops

grops translates the output of GNU troff to PostScript.

grotty

grotty translates the output of GNU troff into a form suitable for typewriter–like devices.

hpftodit

hpftodit creates a font file for use with groff –Tlj4 from an HP tagged font metric file.

indxbib

indxbib makes an inverted index for the bibliographic databases a specified file for use with refer, lookbib, and lkbib.

lkbib

lkbib searches bibliographic databases for references that contain specified keys and prints any references found on the standard output.

lookbib

lookbib prints a prompt on the standard error (unless the standard input is not a terminal), reads from the standard input a line containing a set of keywords, searches the bibliographic databases in a specified file for references containing those keywords, prints any references found on the standard output, and repeats this process until the end of input.

neqn

It is currently not known what neqn is and what it does.

---

nroff

The nroff script emulates the nroff command using groff.

---

pfbtops

pfbtops translates a PostScript font in .pfb format to ASCII.

---

pic

pic compiles descriptions of pictures embedded within troff or TeX input files into commands that are understood by TeX or troff.

---

psbb

psbb reads a file which should be a PostScript document conforming to the Document Structuring conventions and looks for a %%BoundingBox comment.

---

refer

refer copies the contents of a file to the standard output, except that lines between .[and .] are interpreted as citations, and lines between .R1 and .R2 are interpreted as commands about how citations are to be processed.

---

soelim

soelim reads files and replaces lines of the form *.so file* by the contents of *file*.

---

tbl

tbl compiles descriptions of tables embedded within troff input files into commands that are understood by troff.

---

tfmtodit

tfmtodit creates a font file for use with **`groff -Tdvi`**

---

troff

troff is highly compatible with Unix troff. Usually it should be invoked using the groff command, which will also run preprocessors and postprocessors in the appropriate order and with the appropriate options.

Gzip

Contents

The Gzip package contains the gunzip, gzexe, gzip, zcat, zcmp, zdiff, zforece, zgrep, zmore and znew programs.

Description

gunzip

gunzip decompresses files that are compressed with gzip.

---

gzexe

gzexe allows you to compress executables in place and have them automatically uncompress and execute when you run them (at a penalty in performance).

---

gzip

gzip reduces the size of the named files using Lempel−Ziv coding (LZ77).

---

zcat

zcat uncompresses either a list of files on the command line or its standard input and writes the uncompressed data on standard output

---

zcmp

zcmp invokes the cmp program on compressed files.

---

zdiff

zdiff invokes the diff program on compressed files.

---

zforce

zforce forces a .gz extension on all gzip files so that gzip will not compress them twice. This can be useful for files with names truncated after a file transfer.

zgrep

zgrep invokes the grep program on compressed files.

zmore

Zmore is a filter which allows examination of compressed or plain text files one screenful at a time on a soft–copy terminal (similar to the more program).

znew

Znew recompresses files from .Z (compress) format to .gz (gzip) format.

Ld.so

Contents

From the Ld.so package we're using the ldconfig and ldd programs.

Description

ldconfig

ldconfig creates the necessary links and cache (for use by the run−time linker, ld.so) to the most recent shared libraries found in the directories specified on the command line, in the file /etc/ld.so.conf, and in the trusted directories (/usr/lib and /lib). ldconfig checks the header and file names of the libraries it encounters when determining which versions should have their links updated.

---

ldd

ldd prints the shared libraries required by each program or shared library specified on the command line.

Libtool

Contents

The Libtool package contains the libtool and libtoolize programs. It also contains the ltdl library.

Description

libtool

Libtool provides generalized library−building support services.

libtoolize

libtoolize provides a standard way to add libtool support to your package.

ltdl library

Libtool provides a small library, called `libltdl', that aims at hiding the various difficulties of dlopening libraries from programmers.

Linux86

Contents

From the Linux86 package we're using the as86 and ld86 programs.

Description

as86

as86 is an assembler for the 8086..80386 processors.

ld86

ld86 understands only the object files produced by the as86 assembler, it can link them into either an impure or a separate I&D executable.

Lilo

Contents

The Lilo package contains the lilo program.

Description

lilo installs the Linux boot loader which is used to start a Linux system.

Make

Contents

The Make package contains the make program.

Description

make determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them.

Shellutils

Contents

The Shellutils package contains the basename, chroot, date, dirname, echo, env, expr, factor, false, groups, hostid, hostname, id, logname, nice, nohup, pathchk, pinky, printenv, printf, pwd, seq, sleep, stty, su, tee, test, true, tty, uname, uptime, users, who, whoami and yes programs.

Description

basename

basename strips directory and suffixes from filenames.

---

chroot

chroot runs a command or interactive shell with special root directory.

---

date

date displays the current time in a specified format, or sets the system date.

---

dirname

dirname strips non–directory suffixes from file name.

---

echo

echo displays a line of text.

---

env

env runs a program in a modified environment.

---

expr

expr evaluates expressions.

---

factor

factor prints the prime factors of all specified integer numbers.

false

false always exits with a status code indicating failure.

groups

groups prints the groups a user is in.

hostid

hostid prints the numeric identifier (in hexadecimal) for the current host.

hostname

hostname sets or prints the name of the current host system

id

id prints the real and effective UIDs and GIDs of a user or the current user.

logname

logname prints the current user's login name.

nice

nice runs a program with modified scheduling priority.

nohup

nohup runs a command immune to hangups, with output to a non−tty

pathchk

pathchk checks whether file names are valid or portable.

pinky

pinky is a lightweight finger utility which retrieves information about a certain user

printenv

printenv prints all or part of the environment.

printf

printf formats and print data (the same as the printf C function).

pwd

pwd prints the name of the current/working directory

seq

seq prints numbers in a certain range with a certain increment.

sleep

sleep delays for a specified amount of time.

stty

stty changes and prints terminal line settings.

su

su runs a shell with substitute user and group IDs

tee

tee reads from standard input and write to standard output and files.

---

test

test checks file types and compares values.

---

true

True always exitx with a status code indicating success.

---

tty

tty prints the file name of the terminal connected to standard input.

---

uname

uname prints system information.

---

uptime

uptime tells how long the system has been running.

---

users

users prints the user names of users currently logged in to the current host.

---

who

who shows who is logged on.

---

whoami

whoami prints your effective userid.

---

yes

yes outputs a string repeatedly until killed.

Shadow Password Suite

Contents

The Shadow Password Suite contains the chage, chfn, chsh, expiry, faillog, gpasswd, lastlog, login, newgrp, passwd, sg, su, chpasswd, dpasswd, groupadd, groupdel, groupmod, grpck, grpconv, grpunconv, logoutd, mkpasswd, newusers, pwck, pwconv, pwunconv, useradd, userdel, usermod and vipw programs.

Description

chage

chage changes the number of days between password changes and the date of the last password change.

---

chfn

chfn changes user fullname, office number, office extension, and home phone number information for a user's account.

---

chsh

chsh changes the user login shell.

---

expiry

It's currently unknown what this program is for.

---

faillog

faillog formats the contents of the failure log,/var/log/faillog, and maintains failure counts and limits.

---

gpasswd

gpasswd is used to administer the /etc/group file

---

lastlog

lastlog formats and prints the contents of the last login log, /var/log/lastlog. The login−name, port, and last login time will be printed.

login

login is used to establish a new session with the system.

newgrp

newgrp is used to change the current group ID during a login session.

passwd

passwd changes passwords for user and group accounts.

sg

sg executes command as a different group ID.

su

Change the effective user id and group id to that of a user. This replaces the su programs that's installed from the Shellutils package.

chpasswd

chpasswd reads a file of user name and password pairs from standard input and uses this information to update a group of existing users.

dpasswd

dpasswd adds, deletes, and updates dialup passwords for user login shells.

groupadd

The groupadd command creates a new group account using the values specified on the command line and the default values from the system.

groupdel

The groupdel command modifies the system account files, deleting all entries that refer to group.

---

groupmod

The groupmod command modifies the system account files to reflect the changes that are specified on the command line.

---

grpck

grpck verifies the integrity of the system authentication information.

---

grpconv

grpunconv converts to shadow group files from normal group files.

---

grpunconv

grpunconv converts from shadow group files to normal group files.

---

logoutd

logoutd enforces the login time and port restrictions specified in /etc/porttime.

---

mkpasswd

mkpasswd reads a file in the format given by the flags and converts it to the corresponding database file format.

---

newusers

newusers reads a file of user name and cleartext password pairs and uses this information to update a group of existing users or to create new users.

---

pwck

pwck verifies the integrity of the system authentication information.

---

pwconv

pwconv converts to shadow passwd files from normal passwd files.

---

pwunconv

pwunconv converts from shadow passwd files to normal files.

---

useradd

useradd creates a new user or update default new user information.

---

userdel

userdel modifies the system account files, deleting all entries that refer to a specified login name.

---

usermod

usermod modifies the system account files to reflect the changes that are specified on the command line.

---

vipw and vigr

vipw and vigr will edit the files /etc/passwd and /etc/group, respectively. With the −s flag, they will edit the shadow versions of those files, /etc/shadow and /etc/gshadow, respectively.

Man

Contents

The Man package contains the man, apropos whatis and makewhatis programs.

Description

man

man formats and displays the on−line manual pages.

---

apropos

apropos searches a set of database files containing short descriptions of system commands for keywords and displays the result on the standard output.

---

whatis

whatis searches a set of database files containing short descriptions of system commands for keywords and displays the result on the standard output. Only complete word matches are displayed.

---

makewhatis

makewhatis reads all the manual pages contained in given sections of manpath or the preformatted pages contained in the given sections of catpath. For each page, it writes a line in the whatis database; each line consists of the name of the page and a short description, separated by a dash. The description is extracted using the content of the NAME section of the manual page.

Modutils

Contents

The Modutils package contains the depmod, genksyms, insmod, insmod_ksymoops_clean, kerneld, kernelversion, ksyms, lsmod, modinfo, modprobe and rmmod programs.

Description

depmod

depmod handles dependency descriptions for loadable kernel modules.

---

genksyms

genksyms reads (on standard input) the output from gcc −E source.c and generates a file containing version information.

---

insmod

insmod installs a loadable module in the running kernel.

---

insmod_ksymoops_clean

insmod_ksymoops_clean deletes saved ksyms and modules not accessed in 2 days.

---

kerneld

kerneld performs kernel action in user space (such as on−demand loading of modules)

---

kernelversion

kernelversion reports the major version of the running kernel.

---

ksyms

ksyms displays exported kernel symbols.

---

lsmod

lsmod shows information about all loaded modules.

---

modinfo

modinfo examines an object file associated with a kernel module and displays any information that it can glean.

---

modprobe

Modprobe uses a Makefile–like dependency file, created by depmod, to automatically load the relevant module(s) from the set of modules available in predefined directory trees.

---

rmmod

rmmod unloads loadable modules from the running kernel.

Procinfo

Contents

The Procinfo package contains the procinfo program.

Description

procinfo gathers some system data from the /proc directory and prints it nicely formatted on the standard output device.

Procps

Contents

The Procps package contains the free, kill, oldps, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w and watch programs.

Description

free

free displays the total amount of free and used physical and swap memory in the system, as well as the shared memory and buffers used by the kernel.

kill

kills sends signals to processes.

oldps and ps

ps gives a snapshot of the current processes.

skill

skill sends signals to process matching a criteria.

snice

snice changes the scheduling priority for process matching a criteria.

sysctl

sysctl modifies kernel parameters at runtime.

tload

tload prints a graph of the current system load average to the specified tty (or the tty of the tload process if none is specified).

top

top provides an ongoing look at processor activity in real time.

---

uptime

uptime gives a one line display of the following information: the current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.

---

vmstat

vmstat reports information about processes, memory, paging, block IO, traps, and cpu activity.

---

w

w displays information about the users currently on the machine, and their processes.

---

watch

watch runs command repeatedly, displaying its output (the first screenfull).

Psmisc

Contents

The Psmisc package contains the fuser, killall and pstree programs.

Description

fuser

fuser displays the PIDs of processes using the specified files or file systems.

killall

killall sends a signal to all processes running any of the specified commands.

pstree

pstree shows running processes as a tree.

Sed

Contents

The Sed package contains the sed program.

Description

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

Start−stop−daemon

Contents

The Start−stop−daemon contains the start−stop−daemon program.

Description

start−stop−daemon is used to control the creation and termination of system−level processes, usually the ones started during the startup of the system.

Appendix B. Resources

Introduction

A list of books, HOWTOs and other documents you might find useful to download or buy follows. This list is just a small list to start with. We hope to be able to expand this list in time as we come across more useful documents or books.

Books

- Sendmail published by O'Reilly. ISBN: 1−56592−222−0
- Linux Network Administrator's Guide published by O'Reilly. ISBN: 1−56502−087−2
- Running Linux published by O'Reilly. ISBN: 1−56592−151−8

HOWTOs and Guides

All of the following HOWTOs can be downloaded from the Linux Documentation Project site at http://www.linuxdoc.org

- Linux Network Administrator's Guide
- ISP−Hookup−HOWTO
- Powerup2Bash−HOWTO

Other

- The various man and info pages that come with the packages