Linux From Scratch

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 we need

Dedication

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

Table of Contents

Preface

I would like to express my thanks to all the people on the lfs−discuss@linuxfromscratch.org mailinglist. Without all the suggestion they made and all the questions they asked this book would never have become the book it is today.

Thanks you all!

Chapter 1. Introduction

What's this all about?

Having used a number of different Linux distributions, I was never fully satisfied with either 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. No 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 sorted out all kinds of dependency problems, compilation problems, etcetera, a manually Linux system was created and fully operational. I called this system and LFS system which stands for LinuxFromScratch.

Co−authors

- Michael Peters − Author of the Apple PowerPC parts in this document

New versions

The latest version of this document can always be found at http://www.linuxfromscratch.org

Changelog

j.3.1 – April 12th, 2000

- Modified parts of the document so the document can be used for people who want to install LFS on an Apple PowerPC
- 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

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

- lfs−discuss
- 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 devleopments regarding this book

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. This list was created originally to stop the high volume of off−topic messages to the lfs−discuss list. Although some of the messages posted to the linux are somewhat related to this book, the list is also used for anything else that isn't related to this book at all. Feel free to join this list if you have non−LFS questions or just want to discuss a subject.

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−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 a 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−announce or linux.

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

Contact information

Direct all your emails to the lfs−discuss mailinglist preferably. If you need to reach me personally, send an email to gerard@linuxfromscratch.org

Chapter 2. Conventions used in this book

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/hda5. If I read this document myself and I see $LFS somewhere, I will pretend that I read /mnt/hda5. If I read that I have to run this command: cp inittab $LFS/etc I actually will run this: cp inittab /mnt/hda5/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 in stead of replacing it by something like /mnt/hda5. This is accomplished by running: export LFS=/mnt/hda5

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/hda5/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 is 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/hda5. If you make a typo while entering /mnt/hda5, 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 myself have 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 chapter 4.

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 that once, in this paragraph. There is also the possibility that you have the possibility 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.

- Change to the $LFS/usr/src directory by running **`cd $LFS/usr/src`**

- When you have a file that is tar'ed and gzip'ed, you unpack it by running **`tar xvfz filename.tar.gz`** or **`tar xvfz filename.tgz`**. If you don't keep your files in the $LFS/usr/src directory but in some subdirectory, prepend the correct directory name to the filename.

- When you have a file that is tar'ed and bzip'ed, you unpack it by running **`tar --use-compress-prog=bzip2 -xvf filename.tar.bz2`**. If you don't keep your files in the $LFS/usr/src directory but in some subdirectory, prepend the correct directory name to the filename.

- When you have a file that is tar'ed, you unpack it by running **`tar xvf filename.tar`**. If you don't keep your files in the $LFS/usr/src directory but in some subdirectory, prepend the correct directory name to the filename.

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. All the above will be summarized as 'Unpack the xxx archive'. So, when you read that, you run the tar program to ungzip/unbzip and untar it, then you enter the directory that was created and then you read the next line of the installation instructions.

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 (all software up to chapter 7.2 will be re−installed in chapter 7.3) 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 apply anymore when you have restarted your computer into the LFS system). Doing a simple make clean does not always guarantee a totally clean source tree. The configure script also has files lying around in various subdirectories which are rarely removed by the make clean process.

Chapter 3. Packages you need to download

Below is a list of all the software that you need to download for use in this document. I display the sites and directories where you can download the software, but it is up to you to make sure you download the source archive and the latest version. The version numbers correspondent to versions of the software that is known to work and which this document is going to be based on. If you experience problems which you can't solve yourself, download the version that is assumed in this document (in case you download a newer version). The lists are split up in two parts: Mandatory & optional packages for Intel systems and Mandatory & optional packages for PPC based system.

Mandatory packages for Intel systems

- Sysvinit (2.78): ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/
- Bash (2.03): ftp://ftp.gnu.org/gnu/bash
- Linux Kernel (2.2.14): ftp://ftp.kernel.org/pub/linux/kernel/"
- Binutils (2.9.1): ftp://ftp.gnu.org/gnu/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): ftp://ftp.gnu.org/gnu/grep/
- Gzip (1.2.4): ftp://ftp.gnu.org/gnu/gzip/
- Make (3.78.1): ftp://ftp.gnu.org/gnu/make/
- 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 (2.10f): 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 (5.005_03): ftp://ftp.gnu.org/gnu/perl/
- 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 (21): 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): ftp://sunsite.unc.edu/pub/linux/system/boot/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−lfs.tar.gz
- Sysklogd (1.3.31): ftp://sunsite.unc.edu/pub/Linux/system/daemons/
- Vim (5.6): ftp://ftp.vim.org/pub/editors/vim/unix/

Mandatory packages for PPC systems

- Sysvinit (2.78): ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/
- Bash (2.03): ftp://ftp.gnu.org/gnu/bash
- Linux Kernel (2.2.14): ftp://ftp.kernel.org/pub/linux/kernel/"
- USB PowerPC Kernel Patch: ftp://216.22.163.20/usb−2.3.50−1−for−2.2.14.diff.gz
- Binutils (2.9.5.0.29): 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−PPC−Patch: ftp://216.22.163.20/glibc−2.1.3−ctype.patch
- Grep (2.4): ftp://ftp.gnu.org/gnu/grep/
- Gzip (1.2.4): ftp://ftp.gnu.org/gnu/gzip/
- Make (3.78.1): ftp://ftp.gnu.org/gnu/make/
-

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 (2.10f): ftp://ftp.win.tue.nl/pub/linux/utils/util−linux/
- Pmac Linux (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 (5.0): ftp://ftp.gnu.org/gnu/ncurses/
- Less (340): ftp://ftp.gnu.org/gnu/less/
- Perl (5.005_03): ftp://ftp.gnu.org/gnu/perl/
- 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 (21): 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): ftp://sunsite.unc.edu/pub/linux/system/boot/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−lfs.tar.gz
- Sysklogd (1.3.31): ftp://sunsite.unc.edu/pub/Linux/system/daemons/
- Vim (5.6): ftp://ftp.vim.org/pub/editors/vim/unix/

Optional packages for Intel systems

All software below is used in chapters 13 and above and are not strictly necessary. You have to determine for yourself if you want to install certain packages. If, for example, you don't intend to go online with the LFS system, you might not want to install the email, telnet, ftp, www, etc. utilities.

- Netkit−base (0.17): ftp://ftp.uk.linux.org/pub/linux/Networking/netkit−devel/
- Net−tools (1.54): http://www.tazenda.demon.co.uk/phil/net−tools/
- Procmail (3.14): http://ftp://ftp.procmail.org/pub/procmail/
- Sendmail (8.9.3): ftp://ftp.sendmail.org/pub/sendmail/
- Mailx (8.1.1): http://www.linuxfromscratch.org/download/mailx−8.1.1−lfs.tar.gz
- Mutt (1.01i): ftp://ftp.mutt.org/pub/mutt/
- Fetchmail (5.2.0): http://www.tuxedo.org/~esr/fetchmail/
- Netkit−telnet (0.17): ftp://ftp.uk.linux.org/linux/Networking/netkit−devel/
- Proftpd (1.2.0pre9): ftp://ftp.tos.net/pub/proftpd/
- Netkit−ftp (0.17): ftp://ftp.uk.linux.org/linux/Networking/netkit−devel/
- Apache (1.3.11): http://www.apache.org/dist/
- Zlib (1.1.3): http://www.cdrom.com/pub/infozip/zlib/
- Lynx (2.8.2): http://www.slcc.edu/lynx/release/
- PPP (2.3.11): ftp://cs.anu.edu.au/pub/software/ppp/
- XFree86 (3.3.5): ftp://ftp..xfree86.org/pub/XFree86/
-

libPropList (0.9.1): ftp://ftp.windowmaker.org/pub/libs/

- libXpm (4.7): ftp://sunsite.unc.edu/pub/Linux/libs/X/
- libpng (1.0.3): http://www.cdrom.com/pub/png/
- libtiff (3.4): ftp://ftp.sgi.com/graphics/tiff/
- libjpeg (6b): http://www.ijg.org/
- libungif (4.1.0): ftp://prtr−13.ucsc.edu/pub/libungif/
- WindowMaker (0.61.1): ftp://ftp.windowmaker.org/pub/release/

Optional packages for PPC systems

All software below is used in chapters 13 and above and are not strictly necessary. You have to determine for yourself if you want to install certain packages. If, for example, you don't intend to go online with the LFS system, you might not want to install the email, telnet, ftp, www, etc. utilities.

- Patch (2.5.4): ftp://ftp.gnu.org/gnu/patch/
- Netkit−base (0.17): ftp://ftp.uk.linux.org/pub/linux/Networking/netkit−devel/
- Net−tools (1.54): http://www.tazenda.demon.co.uk/phil/net−tools/
- Procmail (3.14): http://ftp://ftp.procmail.org/pub/procmail/
- Sendmail (8.9.3): ftp://ftp.sendmail.org/pub/sendmail/
- Mailx (8.1.1): http://www.linuxfromscratch.org/download/mailx−8.1.1−lfs.tar.gz
- Mutt (1.01i): ftp://ftp.mutt.org/pub/mutt/
- Fetchmail (5.2.0): http://www.tuxedo.org/~esr/fetchmail/
- Netkit−telnet (0.17): ftp://ftp.uk.linux.org/linux/Networking/netkit−devel/
- Proftpd (1.2.0pre9): ftp://ftp.tos.net/pub/proftpd/
- Netkit−ftp (0.17): ftp://ftp.uk.linux.org/linux/Networking/netkit−devel/
- Apache (1.3.11): http://www.apache.org/dist/
- Zlib (1.1.3): http://www.cdrom.com/pub/infozip/zlib/
- Lynx (2.8.2): http://www.slcc.edu/lynx/release/
- PPP (2.3.11): ftp://cs.anu.edu.au/pub/software/ppp/
-

XFree86 (3.3.5): ftp://ftp..xfree86.org/pub/XFree86/

- Xpmac (rev9): ftp://216.22.163.20/Xpmac.rage128.usb.rev9.gz
- libPropList (0.9.1): ftp://ftp.windowmaker.org/pub/libs/
- libXpm (4.7): ftp://sunsite.unc.edu/pub/Linux/libs/X/
- libpng (1.0.3): http://www.cdrom.com/pub/png/
- libtiff (3.4): ftp://ftp.sgi.com/graphics/tiff/
- libjpeg (6b): http://www.ijg.org/
- libungif (4.1.0): ftp://prtr−13.ucsc.edu/pub/libungif/
- WindowMaker (0.61.1): ftp://ftp.windowmaker.org/pub/release/

Chapter 4. Preparing the new system

How we are going to do things

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 document 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'll 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 document, though it isn't impossible to use a libc5 system if you have no choice.

Creating a new partition on Intel systems

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 whistles and bells (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 if 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 document.

Creating a new partition on PPC systems

For PPC start the pfdisk program with the appropriate harddisk option. The pdisk syntax is a little bit different than fdisk syntax, if you are familiar with fdisk.

To create a linux capable partition using the available free space, type c followed by the partition designation of the free space pX, 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`**

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 the command as **`mke2fs /dev/hda5`**

Mounting the new partition

Once 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/hda5, you can access this partition by going to the /mnt/hda5 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 subdirectory you choose (or you can use just the /mnt directory as the mounting point), but a good practice is to create a directory with the same name as the partition's designation. In my case the LFS partition is called hda5 and therefore I mount it on /mnt/hda5

- Create the /mnt directory if it doesn't exist yet
- Create the /mnt/xxx directory where xxx is to be replaced by your LFS partition's designation.
- Mount the LFS partition by running: mount /dev/xxx /mnt/xxx and replace xxx by your LFS partition's designation.

This directory (/mnt/xxx) is the $LFS you have read about earlier. So if you read somewhere to "cp inittab $LFS/etc" you actually will type "cp inittab /mnt/xxx/etc" where xxx is replaced by your partition's designation.

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:

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

```
cp –av /dev $LFS
chown root.root $LFS/dev/*
```

Chapter 5. Making the LFS system bootable

Installing Sysvinit

Under normal circumstances, after the kernel is done loading and initializing various system components, it attempts to load a program called init which will finalize the system boot process. The package found on most if not every single Linux system is called Sysvinit and that's the program we're going to install on our LFS system.

- Unpack the Sysvinit archive
- Enter the src directory
- Open the Makefile file in a text editor
- Somwhere in this file but before the rule all: put this line: ROOT = $LFS
- Preceed every /dev on the last four lines by $(ROOT)

After applying the $(ROOT) parts to the last four lines, they should look like this:

```
@if [! -p $(ROOT)/dev/initctl ]; then \
echo "Creating $(ROOT)/dev/initctl"; \
rm -f $(ROOT)/dev/initctl; \
mknod -m 600 $(ROOT)/dev/initctl p; fi
```

- Install the package by running:

```
make -e LDFLAGS=-static; make install
```

Configuring Sysvinit

In order for Sysvinit to work, we need to create it's configuration file. Create the $LFS/etc/inittab file containing the following:

```
# Begin /etc/inittab

id:2:initdefault:

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

1:2345:respawn:/sbin/sulogin

# End /etc/inittab
```

Creating passwd & group files

As you can see from the inittab file, when we boot the system, init will start the sulogin program and sulogin will ask you for user root's password. This means we need to have at least a passwd file present on the LFS system.

- Create the $LFS/etc/passwd file containing the following:

  ```
  root:s394ul1Bkvmq2:0:0:root:/root:/bin/bash
  ```

- Create the $LFS/etc/group file containing the following:

  ```
  root::0:
  ```

The encoded password above is: *lfs123*

When you logon to your LFS system, enter *lfs123* when asked to enter user root's password.

Installing the Bash shell

When sulogin asks you for the root password and you've entered the password, a shell needs to be started. Usually this is the bash shell. Since there are no libraries installed yet, we need to link bash statically, just like we did with Sysvinit.

- Unpack the Bash archive
- Install Bash by running:

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

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=<currently used image>
  label=<label>
  root=<partition>
  read−only
```

Replace <currently used image> by the kernel image file that you are using to boot your normal Linux system. <label> can be anything you want it to be. I named the label "lfs" What you enter as <label> is what you enter at the LILO−prompt when you choose with system to boot. <partition> must be replaced by your partition's designation (which would be /dev/hda5 in my case).

Now run the lilo program to update the boot loader.

Testing the system

After you've completed this section, we can test the system by rebooting into LFS and see if we can log on to it. When you reboot and are at the LILO prompt, enter the label you have entered in the lilo.conf file to start the LFS system. Then enter root's password and you should be on the bash−prompt now. You won't be able to shutdown the system with a program like shutdown. Although the program is present, it will give you the following error: "You don't exist. Go away." when you try to use the program. The meaning of this error is that the system isn't able to locate the password file. Although the shutdown program is statically linked against the libraries it needs, it still depends on the NSS Library (Name Server Switch) which is part of the GNU C Library, which will be installed in a later chapter. This NSS library passes on information where (in this case) the passwd file can be found.

For now you can reboot the system using the reboot −f command. This will bypass shutting down the system using the shutdown program and reboot immediately. Since the file system is mounted read−only this will not harm our system in any way (though you might get a warning next time you try to mount the system that it wasn't unmounted cleanly the last time and that you should run e2fsck to make sure the file system is still intact).

Chapter 6. Installing a kernel

Note on ftp.kernel.org

In section 2 above I mentioned you can download a new kernel from ftp://ftp.kernel.org/ However, this site is often too busy to get through and the maintainers of this site encourage you to download the kernel from a location near you. You can access a mirror site by going to ftp://ftp.<country code>.kernel.org/ (like ftp.ca.kernel.org).

Configuring the kernel on Intel systems

- Rename the current /usr/src/linux directory to something else (/usr/src/linux can be a symlink rather than an actual directory. Either way, rename it) by running:

```
mv /usr/src/linux /usr/src/linux-old
```

- Remove /usr/include/linux and /usr/include/asm by running:

```
rm -r /usr/include/linux /usr/include/asm
```

- Unpack the Kernel archive in the /usr/src/ directory (this will create a new /usr/src/linux directory)
- Create the /usr/include/linux and /usr/include/asm symlinks by running:

```
cd /usr/include
ln -s ../src/linux/include/linux linux
ln -s ../src/linux/include/asm-<cpu> asm
```

Look in the /usr/src/linux/include directory and see which asm-* directories are present. Choose the correct one for your platform. If you're on an Intel platform, you'd run ln -s ../src/linux/include/asm-i386 asm

- Choose a method to configure the kernel (see the README file for more details on configuration methods) and make sure you don't configure anything as modules at this point. This is because we won't have the necessary software available to load kernel modules for a while.
- After you're done with your kernel configuration, check and set the dependencies by running:

```
make dep
```

- Compile the kernel by running:

```
make bzImage
```

- Copy the bzImage file to the /boot directory by running:

```
cp arch/<cpu>/boot/bzImage /boot/lfskernel
```

Replace <cpu> by the CPU type you are using like i386 on Intel system.s

- Copy the entire kernel source tree to the LFS partition by running:

```
cp –av /usr/src/linux $LFS/usr/src
```

- Create the $LFS/usr/include/linux and $LFS/usr/include/asm symlinks by running:

```
cd $LFS/usr/include
ln –s ../src/linux/include/linux linux
ln –s ../src/linux/include/asm asm
```

Configuring the kernel on PPC systems

- Rename the current /usr/src/linux directory to something else (/usr/src/linux can be a symlink rather than an actual directory. Either way, rename it) by running:

  ```
  mv /usr/src/linux /usr/src/linux−old
  ```

- Remove /usr/include/linux and /usr/include/asm by running:

  ```
  rm −r /usr/include/linux /usr/include/asm
  ```

- Unpack the Kernel archive in the /usr/src/ directory (this will create a new /usr/src/linux directory)

- Create the /usr/include/linux and /usr/include/asm symlinks by running:

  ```
  cd /usr/include
  ln −s ../src/linux/include/linux linux
  ln −s ../src/linux/include/asm−<cpu> asm
  ```

Look in the /usr/src/linux/include directory and see which asm−* directories are present. Choose the correct one for your platform. If you're on an PPC platform, you'd run ln −s ../src/linux/include/asm−ppc asm

- For USB PowerPC systems you need to apply the USP PPC Patch. The patch should be located in /usr/src and you can keep it gzipped:

  ```
  cd /usr/src/linux
  gzip −cd ../usb−2.3.50−1−for−2.2.14.diff.gz | patch −p1
  make distclean
  ```

- Before you configure the actual kernel you need to load the proper defaults for PowerPC. For USB Mac's, the USB you need to choose is OHCI. Do not enable UHCI. You will need to select USB Human Interface Device, the USB keyboard and the USB mouse. You load these defaults by running:

```
pmac_config
```

- Choose a method to configure the kernel (see the README file for more details on configuration methods) and make sure you don't configure anything as modules at this point. This is because we won't have the necessary software available to load kernel modules for a while.

- After you're done with your kernel configuration, check and set the dependencies by running:

```
make dep
```

- Compile the kernel by running:

```
make vmlinux
```

- Copy the vmlinux file to the /boot directory by running:

```
cp vmlinux /boot/lfskernel
```

- Copy the entire kernel source tree to the LFS partition by running:

```
cp -av /usr/src/linux $LFS/usr/src
```

- Create the $LFS/usr/include/linux and $LFS/usr/include/asm symlinks by running:

```
cd $LFS/usr/include
ln -s ../src/linux/include/linux linux
ln -s ../src/linux/include/asm asm
```

Updating LILO

- Edit the /etc/lilo.conf file and go to the LFS section
- Change the imagename to lfskernel
- To update the boot loader, run:

```
lilo
```

Testing the system

Reboot your system and start your LFS system. Verify that the newly installed kernel doesn't perform out−of−the−ordinary actions (such as crashing).

Chapter 7. Installing basic system software

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 deals 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.

There are a number of packages that need to be already installed before we can start installing all the basic system software. A typical configure scripts needs programs like rm, grep, sed, mv, cat, cp, diff. You need to be able to ungzip and untar archives, you need to link programs after you have compiled the objects files. All these (and a few more) programs needs to be available before we can install anything else. These programs are going to be linked statically. The reasoning behind this is that your normal Linux system may have a different C Library version than the LFS system is going to have. The programs you install in this section will be linked against the C Library of your normal Linux system. This may cause library conflicts if you run those programs on the LFS system. Therefore we have to link those programs statically. During the installation of the basic system software set, we will re−install the statically linked software so that they are linked dynamically against the C library on the LFS system.

About debugging symbols

Every program and library is 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 binary significantly. This document will not install software without debugging symbols (as I don't know if the majority of readers do or don't 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 has been used and which C library version is used to link dynamic programs against, but your results will be very 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 Binutils

- Unpack the Binutils archive

- Install the package by running:

```
./configure
make −e LDFLAGS=−all−static
make −e prefix=$LFS/usr install
```

Installing Bzip2

- Unpack the Bzip2 archive

- Open the Makefile file in an editor

- Find the lines that start with $(CC) $(CFLAGS) −o

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

- Install the package by running:

```
make −e LDFLAGS=−static
make −e PRFIX=$LFS/usr install
cd $LFS/usr/bin
mv bunzip2 bzip2 $LFS/bin
```

Installing Diffutils

- Unpack the Diffutils archive
- Install the package by running:

```
./configure
make −e LDFLAGS=−static
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 fixed package from http://www.linuxfromscratch.org/download/diffutils−2.7−fixed.tar.gz

---

Installing Fileutils

- Unpack the Fileutils archive
- Install the package by running:

```
./configure −−disable−nls
make −e LDFLAGS=−static
make −e prefix=$LFS/usr install
cd $LFS/usr/bin
mv chgrp chmod chown cp dd df ln $LFS/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 you need to have gcc−2.95.2 installed. Although any version above 2.8 would do, 2.95.2 is the highlyl 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 The reason is still unknown).

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 in stead we will install gcc in a separate directory

(/usr/local/gcc2952). This way no binaries or header files will be replaced.

- Unpack the GCC archive
- Install the package by running:

```
mkdir $LFS/usr/src/gcc−build
cd $LFS/usr/src/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++
make bootstrap
make install
```

Installing GCC on the LFS system

- Unpack the GCC archive
- Install the package by running:

```
mkdir $LFS/usr/src/gcc−build
cd $LFS/usr/src/gcc−build
../gcc−2.95.2/configure −−enable−languages=c,++ −−disable−nls
make −e LDFLAGS=−static bootstrap
make −e prefix=$LFS/usr local_prefix=$LFS/usr 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:

```
cd $LFS/lib
ln −s ../usr/lib/gcc−lib/<host>/2.95.2/cpp cpp
cd $LFS/usr/lib
ln −s gcc−lib/<host>/2.95.2/cpp cpp
cd $LFS/usr/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

- Unpack the Glibc archive
- 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.
- Create a new file configparms containing:

```
# Begin configparms
slibdir=/lib
sysconfdir=/etc
# End configparms
```

- For PPC only: Copy the glibc−2.1.3−ctype.patch file to the $LFS/usr/src directory and apply the patch by running:

```
cd $LFS/usr/src/glibc−2.1.3
patch −p1 < ../glibc−2.3.1−ctype.patch
```

- If your normal already had a gcc version suitable to compile glibc with, install the package by running:

```
mkdir $LFS/usr/src/glib−build
cd $LFS/usr/src/glibc−build
../glibc−2.1.3/configure −−enable−add−ons
make
make install_root=$LFS install
```

- If your normal system didn'd had a suitable gcc version, install the package by running:

```
mkdir $LFS/usr/src/glibc−build
cd $LFS/usr/src/glibc−build
CC=/usr/gcc2952/bin/gcc \
        ../glibc−2.1.3/configure −−enable−add−ons
make
make install_root=$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:

```
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:

```
cp −av /lib/*nss* $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

- Unpack the Grep archive
- Install the package by running:

```
./configure −−disable−nls
make −e LDFLAGS=−static
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 fixed package from http://www.linuxfromscratch.org/download/grep−2.4−fixed.tar.gz

---

Installing Gzip

- Unpack the Gzip archive
- Install the package by running:

```
./configure
make −e LDFLAGS=−static
make −e prefix=$LFS/usr install
cd $LFS/usr/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−1.2.4−fixed.tar.gz

---

Installing Make

- Unpack the Make archive
- Install the package by running:

```
./configure
make −e LDFLAGS=−static
make −e prefix=$LFS/usr install
```

Installing Sed

- Unpack the Sed archive
- Install the package by running:

```
./configure
make −e LDFLAGS=−static
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 fixed package from http://www.linuxfromscratch.org/download/sed−3.03−fixed.tar.gz

---

Installing Shell Utils

- Unpack the Shell Utils archive
- Install the package by running:

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

Installing Tar

- Unpack the Tar archive
- Install the package by running:

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

Installing Textutils

- Unpack the Textutils archive
- Install the package by running:

```
./configure --disable-nls
make -e LDFLAGS=-static
```

```
make -e prefix=$LFS/usr install
mv $LFS/usr/bin/cat $LFS/bin
```

Installing Util−Linux

- Unpack the Util−Linux archive
- Install the package by running:

```
./configure
cd lib
make
cd ../mount
make -e LDFLAGS=-static
cp mount umount $LFS/bin
cp swapon $LFS/sbin
```

Installing Pmac−utils (PPC systems only)

- Unpack the Pmac−utils archive
- Install the package by running:

```
make clock
cp clock $LFS/sbin
```

- Create a new file hwclock containing:

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

/sbin/clock -s
```

```
# End /sbin/hwclock
```

- Install this file by running:

```
chmod 755 hwclock
mv hwclock $LFS/sbin
```

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 this aspect, I, however, choose to give the full instructions for each and every package. This is simply to avoid any possible confusion and errors. Before you continue with this document you have to restart your system and boot into the LFS system. But before you do that, you need to determine which partition is used as your swap partition. This information can usually be found in the /etc/fstab file. Check this file for a line similar to this one: /dev/hda6 none swap sw 0 0 A different variation is a line simlar to this one: /dev/hda6 swap swap defaults 0 0

The 4th field in a line must contain 'sw'. Or the 2nd and/or 3rd field in a line must contain 'swap'. Keep in mind that there are more variantions possible, though it shouldn't be hard to recognize which line represents a swap partition (because it either contains 'sw' or 'swap' in it). All you need to remember is it's designation (which is /dev/hda6 in my case but this will probably be different on your system). When you have determined which partition is the swap partition, you can reboot your computer now and continue from here.

After you have rebooted you need to run /sbin/hwclock if you're on a PPC system before you execute anything else.

Remounting partition and activating swap

Before the software can be installed we need to remount the partition in read−write mode. Also, we need to activate the swap partition so that we won't risk running out of memory during large compilation processes (such as compiling gcc):

```
mount −n −o remount,rw / /
/sbin/swapon <swap device>
```

Replace <swap device> with the designation that you have determined before you rebooted the system (which is /dev/hda6 in my case).

Installing GCC

- Unpack the GCC archive and install it by running:

```
mkdir /usr/src/gcc−build
cd /usr/src/gcc−build
../gcc−2.95.2/configure −−with−gxx−include−dir=/usr/include/g++ \
```

```
    --enable-shared --enable-languages==c,c++
make bootstrap
make install
```

Installing Bison

- Unpack the Bison archive and install it by running:

```
./configure --datadir=/usr/share/bison
make
make install
```

Installing Mawk

- Unpack the Mawk archive and install it by running:

```
./configure
make
make install
cd /usr/bin
ln -s mawk awk
```

Installing Findutils

- Unpack the Findutils archive and install it by running:

```
./configure
make
make install
```

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

Installing Termcap

- Unpack the Termpcap archive and install it by running:

```
./configure
make
make install
```

Installing Ncurses

- Unpack the Findutils archive and install it by running:

```
./configure --with-shared
make
make install
```

Installing Less

- Unpack the Less archive and install it by running:

```
./configure
make
make install
mv /usr/bin/less /bin
```

Installing Perl

- Unpack the Perl archive and install it by running:

```
./Configure
make
make install
```

Note that we skip the make test step. This is because at the moment the system isn't ready yet for running the perl test suite. So we'll trust that perl compiled properly. If you want to be 100% sure, you have to reinstall Perl youself using the above instructions after you have created the bootscripts and have rebooted the system.

Installing M4

- Unpack the M4 archive and install it by running:

```
./configure
make
make install
```

Installing M4

- Unpack the M4 archive and install it by running:

```
./configure
make
make install
```


Installing Texinfo

- Unpack the Texinfo archive and install it by running:

```
./configure
make
make install
```

Installing Autoconf

- Unpack the Autoconf archive and install it by running:

```
./configure
make
make install
```

Installing Automakme

- Unpack the Automake archive and install it by running:

```
./configure
make install
```

Installing Bash

- Unpack the Bash archive and install it by running:

```
./configure
make
make install
mv /usr/bin/bash /bin
```

Installing Flex

- Unpack the Flex archive and install it by running:

```
./configure
make
make install
```

Installing Binutils

- Unpack the Binutils archive and install it by running:

```
./configure
make
make install
```

Installing Bzip2

- Unpack the Bzip2 archive and install it by running:

```
make
make install
cd /usr/bin
mv bunzip2 bzip2 /bin
```

Installing Diffuitls

- Unpack the Diffutils archive and install it by running:

```
./configure
make
make install
```

Installing E2fsprogs

- Unpack the E2fsprogs archive and install it by running:

```
./configure
make
make install
mv /usr/sbin/mklost+found /sbin
```

Installing File

- Unpack the File archive and install it by running:

```
./configure
make
make install
```

Installing Fileutils

- Unpack the Fileutils archive and install it by running:

```
./configure
make
make install
cd /usr/bin
mv chgrp chmod chown cp dd df ln /bin
mv ls mkdir mknod mv rm rmdir sync /bin
```

Installing Grep

- Unpack the Grep archive and install it by running:

```
./configure
make
make install
```

Installing Groff

- Unpack the Groff archive and install it by running:

```
./configure
make
make install
```

Installing Gzip

- Unpack the Gzip archive and install it by running:

```
./configure
make
make install
cd /usr/bin
mv *z /bin
```

Installing Ld.so

- Unpack the Ld.soarchive and install it by running:

```
cd util
make ldd ldconfig
cp ldd /bin
cp ldconfig /sbin
rm /usr/bin/ldd
```

Installing Libtool

- Unpack the Libtool archive and install it by running:

```
./configure
make
make install
```

Installing Linux86

- Unpack the Linux86 archive and install it by running:

```
cd as
make
make install
cd ../ld
make ld86
make install
```

Installing Lilo

- Unpack the Lilo archive and install it by running:

```
make
make install
```

Installing Make

- Unpack the Make archive and install it by running:

```
./configure
make
make install
```

Installing Shell Utils

- Unpack the Shell Utils archive and install it by running:

```
./configure
make
make install
cd /usr/bin
mv date echo false pwd stty /bin
mv su true uname hostname /bin
```

Installing Shadow Password Suite

- Unpack the Shadow Password Suite archive and install it by running:

```
./configure
make
make install
cd etc
cp limits login.access login.defs.linux shells suauth /etc
mv /etc/login.defs.linux /etc/login.defs
cd /usr/bin
mv chpasswd dpasswd groupadd groupdel groupmod logoutd /sbin
mv mkpasswd newusers useradd userdel usermod grpchk /sbin
mv pwck vipw grpconv grpunconv pwconv pwunconv /sbin
```

Installing Man

- Unpack the Man archive and install it by running:

```
./configure --default
make
make install
```

Installing Modutils

- Unpack the Modutils archive and install it by running:

```
./configure --default
make
make install
```

Installing Procinfo

- Unpack the Procinfo archive and install it by running:

```
make
make install
```

Installing Procps

- Unpack the Procps archive and install it by running:

```
gcc -O3 -Wall -Wno-unused -c watch.c
make
make -e XSCPT="" install
mv /usr/bin/kill /bin
```

Installing Psmisc

- Unpack the Psmisc archive and install it by running:

```
make
make install
```

Installing Sed

- Unpack the Sed archive and install it by running:

```
./configure
make
make install
mv /usr/bin/sed /bin
```

Installing Start−stop−daemon

- Unpack the Start−stop−daemon archive and install it by running:

```
make start−stop−daemon
cp start−stop−daemon /sbin
cp start−stop−daemon.8 /usr/share/man/man8
```

Installing Sysklogd

- Unpack the Sysklogd archive and install it by running:

```
make
```

```
make install
```

Installing Sysvinit

- Unpack the Sysvinit archive and install it by running:

```
cd src
make
make install
```

Installing Tar

- Unpack the Tar archive and install it by running:

```
./configure
make
make install
mv /usr/bin/tar /bin
```

Installing Textutils

- Unpack the Texutils archive and install it by running:

```
./configure
make
make install
mv /usr/bin/cat /bin
```

Installing Vim

- Unpack the Vim−rt and Vim−src archives and install them by running:

```
./configure
make
make install
cd /usr/bin
ln −s vim vi
```

Installing Util−Linux

- Unpack the Util−Linux archive
- Edit the MCONFIG file, find and modify the following variables as follows:

```
HAVE_PASSWD=yes
HAVE_SLN=yes
HAVE_TSORT=yes
```

- Install the package by running:

```
groupadd −g 5 tty
./configure
make
make install
```

- This installation will overwrite the existing /sbin/hwclock if you're using a PPC system. You need to re−create this file. Create a new file hwclock containing the following:

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

/sbin/clock −s

# End /sbin/hwclock
```

- Install this file by running:

```
chmod 755 hwclock
mv hwclock /sbin
```

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:

```
rm /lib/libnss*.so.1 /lib/libnss*2.0*
```

Configuring the 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:

```
cd /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 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.

- Create the /mnt/original directory by running:

```
mkdir /mnt/original
```

- Mount your normal Linux system on this mount point by running:

```
mount /dev/xxx /mnt/original
```

Replace /dev/xxx with your normal partition's designation.

- Copy the Lilo configuration file and kernel images that Lilo uses by running:

```
cp /mnt/original/etc/lilo.conf /etc
cp /mnt/original/boot/* /boot
```

If your normal Linux system does not have (all of) it's kernel images in /mnt/original/boot, then check your /etc/lilo.conf file for the location of those files and copy those as well to the location where /etc/lilo.conf expects them to be. Or you can copy them to /boot regardless and modify the /etc/lilo.conf file so it contains the new paths for the images as you have them on the LFS system. Either way works fine, it's up to you how you want to do it.

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

After you have made the following modification to the /etc/inittab file, you will be able to logon to it as you are used to (using the agetty and login programs). Sulogin won't be used anymore for normal logins.

- Edit the /etc/inittab file and modify it so that it contains 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 file by running:

```
touch /var/run/utmp
```

- Give it the proper file permissions by running:

```
chmod 644 /var/run/utmp
```

Chapter 8. Creating system boot scripts

These bootscripts are started at system boot time. The scripts are responsible for mounting the root file system in read−write mode, activating swap, setting up some system settings and starting the various daemons that our system needs.

Preparing the directories and master files

You need the Sysvinit package again for this section.

- Create the necessary directories by running:

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

- Go to the unpacked Sysvinit source directory
- Copy the Debian/etc/init.d/rc file to: /etc/init.d
- Go to the /etc/init.d directory
- Create a new file rcS containing:

```
#!/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:

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

echo -n "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:

```
#!/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:

```
#!/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:

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

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

echo "Deactivating swap..."
/bin/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:

```
#!/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:

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

echo "Activating swap..."
/sbin/swapon −av

if [ −f /fastboot ]
then
  echo "Fast boot, no file system check"
else
  /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 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."
      /sbin/sulogin
      /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:

```
#!/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
```

Creating the setclock script (PPC systems only)

- Create a new file /etc/init.d/setclock containing:

```
#!/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
```

Setting up symlinks and permissions

- Set the proper file permissions and symlinks by running:

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

You only need to enter setclock in the chmod command and setclock in the ln command if you're on a PPC system. Otherwise pretend it's not there ;)

Creating the /etc/fstab file

- Create a new file /etc/fstab containing:

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

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

Chapter 9. Setting up basic networking

Installing Netkit−base

- Unpack the netkit−base archive and install it by running:

```
./configure
make
make install
cd etc.sample
cp services protocols /etc
mv /usr/bin/ping /bin
```

Installing Net−tools

- Unpack the Net−tools archive and install it by running:

```
make
make install
mv /usr/bin/netstat /bin
cd /usr/sbin
mv ifconfig route /sbin
```

Creating the /etc/init.d/localnet bootscript

- Create a new file /etc/init.d/localnet containing:

```
#!/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 permissions by running:

```
chmod 755 /etc/init.d/localnet
```

- Create the proper symlinks by running:

```
cd /etc/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.

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:

```
<myip> 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 me.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 me.lfs.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 me.lfs.org <contents of /etc/hostname>
# End /etc/hosts (network card version)
```

Of course, change the 192.168.1.1 and me.lfs.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 not, skip this section.

- Create a new file /etc/init.d/ethnet containing:

```
#!/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
NETWORK="209.83.245.0" # Replace with your own Network address
```

```
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 "Setting up route..."
/sbin/route add −net $(NETWORK) netmask $(NETMASK) eth0
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 permissions by running:

```
chmod 755 /etc/init.d/ethnet
```

- Create the proper symlinks by running:

```
cd /etc/init.d/rc2.d
ln −s ../init.d/ethnet S10ethnet
```

Testing the network setup

- Start the just created localnet script by running:

```
/etc/init.d/localnet
```

-

Start the just created ethnet script, if you have one, by running:

```
/etc/init.d/ethnet
```

- Test if the /etc/hosts file and devices are working by running:

```
ping <FQDN>
ping <contents of /etc/hostname>
ping localhost
ping 127.0.0.1
ping <IP address of network card if you have one>
```

All these ping command should work without failure. If so, the basic networking is working.

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).

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 keep in 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.

Chapter 10. Installing network daemons

Setting up SMTP

Creating groups and users

- Create the groups and users needed by Sendmail by running:

```
groupadd –g 1 bin
groupadd –g 2 kmem
groupadd –g 3 mail
useradd –u 1 –g bin –d /bin –s /bin/sh bin
```

Creating directories

Outgoing mail processed by Sendmail is put in the /var/spool/mqueue directory. Incoming mail is forwarded to Procmail by Sendmail so we need to have an incoming mail directory as well which is /var/mail. We'll create these directories and give them the proper permissions:

```
mkdir /var/spool/mqueue
mkdir /var/mail
cd /var/spool
ln –s ../mail mail
chmod 700 /var/spool/mqueue
chmod 755 /var/mail
chgrp mail /var/mail
chmod 1777 /tmp
```

Installing Sendmail

- Unpack the Sendmail archive and install it by running:

```
cd src
./Build
./Build install
```

Configuring Sendmail

Configuring Sendmail isn't as easily said as done. There are a lot of things you need to consider while configuring Sendmail and I can't take everything into account. That's why at this time we'll create a very basic and standard setup. If you want to tweak Sendmail to your own liking, go right ahead. You could always use your existing /etc/sendmail.cf (or /etc/mail/sendmail.cf) file if you need to use certain features.

- Go to the ../cf directory
- Create a new file cf/lfs.mc containing:

```
OSTYPE(LFS)
FEATURE(`local_procmail')
FEATURE(`nouucp')
define(`confDEF_USER_ID',nobody:daemon)
define(`confERROR_MODE',m)
define(`confSAFE_QUEUE')
MAILER(local)
MAILER(smtp)
```

- Create an empty file ostype/LFS.m4 by running:

```
touch ostype/LFS.m4
```

- Compile and install the lfs.mc file by running:

```
m4 m4/cf.m4 cf/lfs.mc > cf/lfs.cf
cp cf/lfs.cf /etc/sendmail.cf
```

- Create an emtpy file /etc/aliases by running:

```
touch /etc/aliases
```

Per RFC822 it's required that every site be set up so that mail addressed to postmaster is always delivered successfully. It's up to you to either add the postmaster alias and possibly other aliases (like aliassing mail for user root to a normal user)

- Initialize the aliases database by running:

```
sendmail -bi
```

Installing Procmail

- Unpack the Procmail archive and install it by running:

```
make
make install
make install-suid
```

Creating the /etc/init.d/sendmail script

- Create a new file /etc/init.d/sendmail containing:

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

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

case "$1" in
  start)
```

```
    echo -n "Starting Sendmail..."
    start-stop-daemon -S -q -o -x /usr/sbin/sendmail -- -bd
    check_status
    ;;

  stop)
    echo -n "Stopping Sendmail..."
    start-stop-daemon -K -q -o -p /var/run/sendmail.pid
    check_status
    ;;

  reload)
    echo -n "Reloading Sendmail configuration file..."
    start-stop-daemon -K -q -s 1 -p /var/run/sendmail.pid
    check_status
    ;;

  restart)
    echo -n "Stopping Sendmail..."
    start-stop-daemon -K -q -o -p /var/run/sendmail.pid
    check_status

    sleep 1

    echo -n "Starting Sendmail..."
    start-stop-daemon -S -q -o -x /usr/sbin/sendmail -- -bd
    check_status
    ;;

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

esac

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

Setting up permissions and symlinks

- Set the proper permissions by running:

```
cd /etc/rc2.d
ln -s ../init.d/sendmail S20sendmail
cd ../rc0.d
```

```
ln −s ../init.d/sendmail K20sendmail
cd ../rc6.d
ln −s ../init.d/sendmail K20sendmail
```

Setting up FTP

Creating groups and users

- Create the goups and users used by ProFTPD by running:

```
groupadd −g 65534 nogroup
groupadd −g 4 ftp
useradd −u 65535 −g nogroup −d /home nobody
useradd −u 4 −g ftp −s /bin/sh −m ftp
```

Installing ProFTPD

- Unpack the ProFTPD archive and install it by running:

```
./configure
make
make install
```

Creating the /etc/init.d/proftpd script

- Create a new file /etc/init.d/proftpd containing:

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

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

```
}

case "$1" in
  start)
    echo -n "Starting Pro FTP daemon..."
    start-stop-daemon -S -q -o -x /usr/sbin/proftpd
    check_status
    ;;

  stop)
    echo -n "Stopping Pro FTP daemon..."
    start-stop-daemon -K -q -o -x /usr/sbin/proftpd
    check_status
    ;;

  restart)
    echo -n "Stopping Pro FTP daemon..."
    start-stop-daemon -K -q -o -x /usr/sbin/proftpd
    check_status

    sleep 1

    echo -n "Starting Pro FTP daemon..."
    start-stop-daemon -S -q -o -x /usr/sbin/proftpd
    check_status
    ;;

  *)
    echo "Usage: $0 {start|stop|restart}"
    ;;

esac

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

Setting up permissions and symlinks

- Set the proper permissions by running:

```
chmod 755 /etc/init.d/proftpd
```

- Create the proper symlinks by running:

```
cd /etc/rc2.d
ln -s ../proftpd S30proftpd
cd ../rc0.d
ln -s ../proftpd S30proftpd
cd ../rc6.d
ln -s ../proftpd S30proftpd
```

Setting up Telnet

Installing telnet daemon + client

- Unpack the Netkit−telnet archive and install it by running:

```
./configure
cd telnetd
make
make install
```

Creating the /etc/inetd.conf configuration file

- Create a new file /etc/inetd.conf containing:

```
# Begin /etc/inetd.conf

telnet stream tcp nowait root /usr/sbin/in.telnetd

# End /etc/inetd.conf
```

Creating the /etc/init.d/inetd script

- Create a new file /etc/init.d/inetd containing:

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

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

```
    echo "FAILED"
  fi
}

case "$1" in
  start)
    echo −n "Starting Internet Server daemon..."
    start−stop−daemon −S −q −o −x /usr/sbin/inetd
    check_status
    ;;

  stop)
    echo −n "Stopping Internet Server daemon..."
    start−stop−daemon −K −q −o −p /var/run/inetd.pid
    check_status
    ;;

  reload)
    echo −n "Reloading Internet Server configuration file..."
    start−stop−daemon −K −q −s 1 −p /var/run/inetd.pid
    check_status
    ;;

  restart)
    echo −n "Stopping Internet Server daemon..."
    start−stop−daemon −K −q −o −p /var/run/inetd.pid
    check_status

    sleep 1

    echo −n "Starting Internet Server daemon..."
    start−stop−daemon −S −q −o −x /usr/sbin/inetd
    check_status
    ;;

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

esac

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

Setting up permissions and symlinks

- Set the proper permissions by running:

```
chmod 755 /etc/init.d/inetd
```

- Create the proper symlinks by running:

```
cd /etc/rc2.d
ln -s ../init.d/inetd S40inetd
cd ../rc0.d
ln -s ../init.d/inetd K40inetd
cd ../rc6.d
ln -s ../init.d/inetd K40inetd
```

Setting up PPP

Configuring the kernel

Before you can logon to the Internet, the kernel must be ppp−aware. You can accomplish this by compiling ppp−support directly into the kernel, or compiling the ppp drivers are modules which you load when you need them. Whatever you prefer, do it now by re−configuring the kernel if necessary. If your LFS kernel is already ppp−aware than you don't have to re−configure the kernel.

Creating group

- Create the group used by PPP by running:

```
groupadd −g 7 daemon
```

Installing PPP

- Unpack the PPP archive and install it by running:

```
./configure
make
make install
```

Creating /etc/resolv.conf

- Create a new file /etc/resolv.conf containing:

```
# Begin /etc/resolv.conf

nameserver <IP address of your ISP's primary DNS server>
nameserver <IP address of your ISP's secundary DNS server>
```

```
# End /etc/resolv.conf
```

Creating /etc/ppp/peers/provider

- Create the /etc/ppp/peers directory by running:

```
mkdir /etc/ppp/peers
```

- Create a new file /etc/ppp/peers/provider containing:

```
# Begin /etc/ppp/peers/provider

noauth
connect "/usr/sbin/chat -v -f /etc/chatscripts/provider"
/dev/<device>
115200
defaultroute
noipdefault

# End /etc/ppp/peers/provider
```

Replace <device> with the device that correspondents with your modem.

Creating the /etc/chatscripts/provider file

- Create the /etc/chatscripts directory by running:

```
mkdir /etc/chatscripts
```

- Create a new file /etc/chatscripts/provider containing:

```
# Begin /etc/chatscripts/provider

ABORT BUSY
ABORT "NO CARRIER"
ABORT VOICE
ABORT "NO DIALTONE"
ABORT "NO ANSWER"
"" ATZ
OK ATDT <ISP's phonenumber>
TIMEOUT 35
CONNECT ''
TIMEOUT 10
ogin: \q<username>
TIMEOUT 10
assword: \q<mysecretpassword>

# End /etc/chatscripts/provider
```

Chapter 11. Installing network clients

Installing Email clients

Installing Mailx

- Unpack the Mailx archive and install it by running:

```
make
make install
```

Installing Mutt

My favorite email client is Mutt, so that's why we're installing this one. Feel free to skip the installation of Mutt and install your own favorite client. After all, this is going to be your system. Not mine.

If your favorite client is an X Window client (such as Netscape Mail) then you'll have to sit tight a little while till we've installed X.

- Unpack the Mutt archive and install it by running:

```
./configure
make
make install
```

Installing Fetchmail

- Unpack the Fetchmail archive and install it by running:

```
./configure
make
make install
```

Installing FTP client

- Unpack the netkit−ftp archive and install it by running:

```
./configure
make
make install
```

Installing HTTP client

Installing Zlib

- Unpack the Zlib archive and install it by running:

```
./configure --shared
make
make install
```

Installing Lynx

- Unpack the Lynx archive and install it by running:

```
./configure --libdir=/etc --with-zlib
make
make install
make install-help
make install-doc
```

Installing Telnet client

- Unpack the Netkit−telnet archive and install it by running:

```
./configure
cd telnet
make
make install
```

Installing PPP clients

- Create a new file /usr/bin/connect containing:

```
#!/bin/sh
# Begin /usr/bin/connect

/usr/sbin/pppd call provider

# End /usr/bin/connect
```

- Create a new file /usr/bin/disconnect containing:

```
    #!/bin/sh
    # Begin /usr/bin/disconnect

set -- `cat /var/run/ppp*.pid`

case $# in
  0)
    kill -15 `ps axw|grep "pppd call [[allnum:]]+" \
    grep -v grep|awk '{print $1}'`
    exit 0
    ;;
  1)
    kill -15 $1
    exit 0
    ;;
esac

    # End /usr/bin/connect
```

Chapter 12. Installing X Window System

Installing X on Intel systems

- Unpack the X archive and install it by running:

```
make World
make install
make install.man
```

Installing X on PPC systems

- Unpack the X archive and install it by running:

```
make World
make install
make install.man
```

If make World fails with a gcc complaint (which it probably will) you will need to edit the xc/config/cf/linux.cf file.

- Open the xc/config/cf/linux.cf file in a text editor and go down to line 287 which says *#define StandardDefines −Dlinux LinuxMachineDefines LinuxSourceDefines*

- Below that line entere these five lines:

```
#define DefaultCCOptions -fsigned-char
#define OptimzedDebugFlags -O2
#define LinuxMachineDefines -D__powerpc__
#define ServerOSDefines XFree86ServerOSDefines -DDDXTIME -DPART_NET
#define ServerExtraDefines -DGCCUSESGAS XFree86ServerDefines
```

- Now install X by running the same commands as before:

```
make World
make install
make install.man
```

Installing the Xpmac server (PPC systems only)

Future plans will include building the Xpmac server from scratch. However, that has not yet been implemented. You don't need an ATI Rage 128 card nor do you need USB to use this server.

- Unpack the Xpmac archive and install it by running:

```
cp Xpmac.rage128.usb.rev9 /usr/X11R6/bin
cd /usr/X11R6/bin
chmod 755 Xpmac.rage128.usb.rev9
ln -s Xpmac.rage128.usb.rev9 X
```

Creating /etc/ld.so.conf

- Create a new file /etc/ld.so.conf containing:

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

/lib
/usr/lib
/usr/X11R6/lib

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

- Update the dynamic loader cache by running:

```
ldconfig
```

Creating necessary symlinks

The following symlinks are necessary for the pre−processor and software to find the X11 files.

- Create the /usr/include/X11 symlink by running:

```
cd /usr/include
ln −s ../X11R6/include/X11 X11
```

- Create the /usr/X11 symlink by running:

```
cd /usr
ln −s X11R6 X11
```

Adding /usr/X11/bin to the $PATH environment variable

There are a few ways to add the /usr/X11/bin to the $PATH variable. One of the ways of doing so is as follows:

```
echo export PATH=\$PATH:/usr/X11/bin >> /etc/profile
```

Configuring X on Intel systems

- Configure the X server by running:

```
xf86config
```

Configuring X on PPC systems

You don't need to worry about a XF86Config file when using the Xpmac server. This server doesn't use this configuration file, so that means you don't need to configure X.

Installing WindowMaker

I choose to install Window Maker as the Window Manager. This is because I've used WindowMaker for quite a while now and I'm very satisfied with it. As usual, you don't have to do what I'm doing; install whatever you want. As you might know, you can install several Window Managers simultaneously and choose which one to start by specifying it in the $HOME/.xinitrc (or $HOME/.xsession in case you decide to use xdm) file.

I was meaning to use KDE instead of WindowMaker but I can't get KDE to compile using gcc−2.95.2 (at least I think it's to be blamed on gcc−2.95.2). I'll wait for KDE2 to be released and hope that version does compile.

Installing libPropList

- Unpack the libPropList archive and install it by running:

```
./configure
make
make install
```

Installing libXpm

- Unpack the libXpm archive and install it by running:

```
xmkmf
make Makefiles
make includes
make depend
cd lib
make
make install
cd ..
make
make install
```

Installing libpng

- Unpack the libpng archive and install it by running:

```
./configure
make
make install
```

Installing libtiff

- Unpack the libtiff archive and install it by running:

```
./configure
make
make install
```

Installing libjpeg

- Unpack the libjpeg archive and install it by running:

```
./configure --enable-shared --enable-static
make
make install
```

Installing libungif

- Unpack the libungif archive and install it by running:

```
./configure
```

```
make
make install
```

Installing WindowMaker

- Unpack the WindowMaker archive and install it by running:

```
./configure
make
make install
```

Updating dynamic loader cache

- Update the dynamic loader cache by running:

```
ldconfig
```

Configuring WindowMaker

Every user who wishes to use WindowMaker has to run the wmaker.inst script before he or she can use it. This script will copy the necessary files into the user's home directory and modify the $HOME/.xinitrc file (or create it if it's not there yet).

- Setup WindowMaker for yourself by running:

```
wmaker.inst
```

Appendix A. Resources

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

- Linux Network Administrator's Guide online at http://www.linuxdoc.org
- ISP−Hookup−HOWTO online at http://www.linuxdoc.org

Other

- The various man and info pages that come with the packages